



Clasificador de sonidos urbanos mediante redes neuronales en dispositivos de bajo coste

David Salvo Gutiérrez

Tutor: Maria Gemma Piñero Sipan

Cotutor: Alberto González Salvador

Trabajo Fin de Máster presentado en la Escuela Técnica Superior de Ingenieros de Telecomunicación de la Universitat Politècnica de València, para la obtención del Máster Universitario en Ingeniería de la Telecomunicación

Curso 2019-20

Valencia, 7 de septiembre de 2020

Resumen

En este trabajo se propone un clasificador de eventos acústicos para espacios urbanos “ruidosos”, por medio de una red de sensores que monitorizan eventos acústicos y los clasifica con una *Convolutional Neural Network*, haciendo uso de componentes de bajo coste.

La arquitectura propuesta, emplea una Raspberry Pi a la que se conecta un micrófono que hace de nodo sensor. A través del micrófono, el nodo detecta un evento acústico y lo clasifica entre 10 clases sonoras posibles. La etiqueta del evento clasificado es enviado al *broker* para su futura distribución y visualización, a través del *framework* de FIWARE por la que se ha creado el flujo de comunicación entre nodos y la base de datos.

El objetivo de este proyecto es caracterizar y analizar el comportamiento y funcionamiento de un sistema de edge computing implementando técnicas de machine learning e inteligencia artificial en nodos de bajo coste. Analizando la precisión, eficiencia energética y caracterización de los mismos.

Palabras clave: aprendizaje automático, inteligencia artificial, edge computing, redes neuronales convolucionales, detección de eventos acústicos, FIWARE, redes IoT, broker, raspberry pi, node-red.

Resum

En aquest treball es proposa un classificador d'esdeveniments acústics per a espais urbans "sorollosos", per mitjà d'una xarxa de sensors que monitoritzen esdeveniments acústics i els classifica amb una *Convolutional Neural Network*, fent ús de components de baix cost.

L'arquitectura proposta, empra una Raspberry Pi a la qual es connecta un micròfon que fa de node sensor. Mitjançant el micròfon, el node detecta un esdeveniment acústic i el classifica entre 10 classes sonores possibles. La etiqueta de l'esdeveniment classificat és enviat al *broker* per a la seva futura distribució i visualització, a través del *framework* de FIWARE per la qual s'ha creat el flux de comunicació entre nodes i la base de dades.

L'objectiu d'aquest projecte és caracteritzar i analitzar el comportament i funcionament d'un sistema de edge computing implementant tècniques de machine learning i intel·ligència artificial en nodes de baix cost. Analitzant la precisió, eficiència energètica i caracterització dels mateixos.

Paraules clau: aprenentatge automàtic, intel·ligència artificial, edge computing, xarxes neuronals convolucionals, detecció d'esdeveniments acústics, FIWARE, xarxes IOT, broker, raspberry pi, node-xarxa.

Abstract

In this work, a classifier of acoustic events is proposed for “noisy” urban spaces, by means of a network of sensors that monitor acoustic events and classify them with a *Convolutional Neural Network*, making use of low-cost components.

The proposed architecture uses a Raspberry Pi to which a microphone is connected to act as a sensor node. Through the microphone, the node detects an acoustic event and classifies it among 10 possible sound classes. The tag of the classified event is sent to the *broker* for future distribution and visualization, through the *FIWARE framework* by which the communication flow between nodes and the database has been created.

The objective of this project is to characterize and analyze the behavior and operation of an edge computing system by implementing machine learning and artificial intelligence techniques in low-cost nodes. Analyzing the precision, energy efficiency and characterization of the same.

Keywords: machine learning, artificial intelligence, edge computing, convolutional neural networks, acoustic event detection, FIWARE, IoT networks, broker, raspberry pi, node-red.

Índice general

I Memoria

1. Introducción	1
1.1. Motivación	1
1.2. Objetivos	2
1.2.1. Clasificador de eventos acústicos	3
1.2.2. Implementación en Raspberry Pi	3
1.2.3. Red IoT	4
2. Contexto y estado del arte	5
2.1. La Era Urbana Inteligente	5
2.1.1. Horizonte Europa	6
2.1.2. Exposición del problema	8
2.1.3. Sounds of New York City	8
2.2. Aprendizaje Automático	10
2.2.1. Procedimiento de Aprendizaje	10
2.2.2. Arquitectura y técnicas del Aprendizaje Automático	11
2.2.3. Redes Neuronales	13
2.2.4. Redes Neuronales Feed-Forward	17
2.2.4.1. Convolutional Neural Networks (CNN)	18
3. Componentes	21
3.1. Entorno de programación	21
3.2. Hardware	25
3.3. Dataset	27
4. Demostrador	29
4.1. Arquitectura	29
4.2. Extracción de Características del Sonido Urbano	32
4.3. Clasificador CNN	35
4.3.1. Extracción de Características	36
4.3.2. Estructura de la CNN	36
4.3.3. Entrenamiento del Clasificador	38
4.4. Red IoT FIWARE	44
4.4.1. Arquitectura	44
4.4.1.1. Modelo Suscripción	45
4.4.1.2. Orion Context-Broker	46

4.4.1.3.	Entidad	47
4.4.1.4.	MongoDB	50
4.4.1.5.	QuantumLeap	50
4.4.1.6.	CrateDB	50
4.4.1.7.	Grafana	51
4.4.2.	Despliegue de RED	51
4.4.2.1.	Paso 1: Configuración imagen Docker-Compose servidor FI- WARE	51
4.4.2.2.	Paso 2: Despliegue del servicio	53
4.4.2.3.	Paso 3: Comprobación de servicios con Postman	56
4.4.2.4.	Paso 4: Crear una Entidad en Orion	56
4.4.2.5.	Paso 5: Crear suscripción de QuantumLeap a Orion para esa En- tidad o conjunto de Entidades	62
4.4.2.6.	Paso 6: Crear Dashboard en Grafana	64
4.4.2.7.	Esquema Funcional de la red	71
4.5.	Raspberry Pi	72
4.5.1.	Configuración Raspberry Pi	72
4.5.2.	Rendimiento	78
5.	Conclusiones	81
5.1.	Conclusiones	81
5.2.	Líneas futuras	82
6.	Anexo I	83
6.1.	Extracción de Características por Clase	83
7.	Anexo II	89
7.1.	Código fuente Github	89
	Bibliografía	91

Índice de figuras

1.1. Esquema del Proyecto	3
2.1. Esquema Proceso de Aprendizaje	10
2.2. Esquema Aprendizaje Automático	12
2.3. Esquema Aprendizaje Supervisado	13
2.4. Esquema Aprendizaje No Supervisado	13
2.5. Esquema Neurona Biológica	14
2.6. Esquema Neurona Artificial	14
2.7. Ejemplos Funciones de Activación	15
2.8. Esquema Red Neuronal	16
2.9. Red Neuronal Perceptrón Multicapa (MLP) [19]	18
2.10. Red Neuronal Convolutacional (CNN) [19]	19
2.11. Ejemplo de cómo se aplica la convolución en imagenes usando "kernel filters"[20]	20
3.1. Entorno de desarrollo Jupyter Notebook	23
3.2. Imagen de una Raspberry Pi 3 Model B	27
3.3. Imagen de un Micrófono mini DSP	27
4.1. Esquema global del proyecto	30
4.2. Imagen descriptiva del nodo sensor de este proyecto I	30
4.3. Imagen descriptiva del nodo sensor de este proyecto II	31
4.4. Caracterización acústica de perros ladrando	33
4.5. Esquema Función de Extracción de Características de un fragmento de Audio	34
4.6. Esquema Implementación Clasificador de sonidos urbanos con CNN	35
4.7. Esquema Arquitectura CNN	36
4.8. Extracción Vector Características (features)	37
4.9. Esquema Learning Rate	38
4.10. Resultado Fase Entrenamiento Clasificador	39
4.11. Matriz de Confusión test-dataset 1	39
4.12. Matriz de Confusión test-dataset 1	40
4.13. Matriz de Confusión test-dataset 2	40
4.14. Matriz de Confusión test-dataset 2	41
4.15. Datos de salida del Clasificador propuesto.	42
4.16. Arquitectura del modelo DS-CNN	44
4.17. Arquitectura de Red IoT del proyecto	45
4.18. Ejemplo Protocolo de Publicación/Suscripción (MQTT)	46
4.19. Esquema agentes red IoT FIWARE	46
4.20. Estructura modelo de datos Device, empleado en el proyecto	48

4.21. Atributos extraordinarios modelos de datos Device	49
4.22. Esquema de redes del servicio FIWARE	51
4.23. Configuración de redes y volúmenes en la imagen de docker-compose.	52
4.24. Imagen docker-compose Orion	53
4.25. Imagen docker-compose QuantumLeap	53
4.26. Imagen docker-compose CrateDB	53
4.27. Imagen docker-compose MongoDB	54
4.28. Imagen docker-compose Grafana	54
4.29. Comando para lanzar los servicios de la imagen docker-compose	54
4.30. Comando para comprobar el estado de los contenedores instanciados en Docker	55
4.31. Comando para finalizar la ejecución de los servicios especificados en la imagen docker-compose	55
4.32. Comprobación versión Orion Context-Broker	57
4.33. Comprobación versión QuantumLeap	58
4.34. Comprobación de Entidades disponibles en Orion	59
4.35. Cabecera empleada en el contexto FIWARE de este proyecto	60
4.36. Creación entidad en Python I	60
4.37. Creación entidad en Python II	61
4.38. Estructura suscripción de QuantumLeap a Orion	63
4.39. Información almacenada por QuantumLeap de la suscripción anterior	65
4.40. Tabla asociada a la suscripción Fig. 4.38, aplicación CrateDB	66
4.41. Aplicación de Grafana	66
4.42. Asignación de nuestro Data Source para crear el Dashboard	67
4.43. Dashboard de la red creado en Grafana	68
4.44. Panel que indica la cantidad de eventos detectados por cada clase clasificada.	68
4.45. Panel para visualizar la localización de los sensores en el mapa.	69
4.46. Panel para visualizar el porcentaje de clasificación por cada clase.	69
4.47. Panel que indica el número total de clases detectadas por la red.	69
4.48. Panel en el que se visualiza en tiempo real cada evento clasificado y el sensor responsable de esa clasificación.	70
4.49. Esquema Funcional de la red al completo.	71
4.50. Raspberry clasificando el evento acústico Cláxon	76
4.51. Raspberry clasificando el evento acústico Sirena	76
4.52. Raspberry clasificando el evento acústico Pistola	77
4.53. Esquema funcionamiento del Clasificador en la Raspberry Pi	78
4.54. Arquitectura de Red implementada en el proyecto	78
6.1. Caracterización acústica del aire acondicionado	83
6.2. Caracterización acústica del cláxon	84
6.3. Caracterización acústica de niños jugando	84
6.4. Caracterización acústica de perros ladrando	85
6.5. Caracterización acústica del ruido por Perforación	85
6.6. Caracterización acústica del ruido de motor	86
6.7. Caracterización acústica de los disparos	86
6.8. Caracterización acústica del martillo neumático	87
6.9. Caracterización acústica de la Sirena	87
6.10. Caracterización acústica de la música callejera	88

7.1. Muestra del repositorio en el que se ha alojado el proyecto completo.	90
7.2. Fragmento de código subido al repositorio	90

Índice de tablas

4.1. Resultado clasificación test-dataset 1	42
4.2. Resultado clasificación test-dataset 2	42
4.3. Comparación de precisión de otros modelos publicados sobre el dataset Urban- sound8k	43
4.4. Especificaciones de hardware y tiempo total de procesamiento	79
4.5. Tiempo de procesamiento por tarea específica	79
4.6. Consumo de Recursos del clasificador en la Raspberry Pi	80

Parte I

Memoria

Capítulo 1

Introducción

1.1. Motivación

De los 7,5 billones de personas que viven en el mundo actualmente, el 55 % se encuentra en zonas urbanas, porcentaje que se estima crecerá hasta el 68 % para el 2050 de acuerdo a las Naciones Unidas [1]. A la vez que el mundo se va urbanizando, la densidad de población y el consumo de recursos se va acelerando, convirtiendo en necesidad el desarrollo sostenible y eficiente de las zonas urbanas para garantizar la calidad de vida de sus habitantes. De esta forma, en la última década ha supuesto un reto crear sendas estrategias y planificaciones urbanas teniendo en cuenta las necesidades económicas, sociales y medio ambientales de cada zona. Este fenómeno ha supuesto un cambio de paradigma en la gestión de las ciudades, en el que la tecnología de la información se ha convertido en el principal aliado en el desarrollo de las mismas, con lo que se conoce como “smart cities”. Desde entonces, se han creado diferentes puntos de vista sobre lo que se entiende como ciudad inteligente [2].

En esta línea, crear sistemas de información capaces de obtener información del medio en tiempo real y establecer marcos contextuales que faciliten la toma de decisiones, se ha convertido en una ventaja competitiva a la hora de crear soluciones y estrategias de desarrollo urbano inteligente. De todos los indicadores y eventos urbanos que se pueden sensorizar y monitorizar en una ciudad, en este proyecto se ha elegido el “sonido urbano” debido a su papel en la polución urbana y desgaste de la calidad de vida de los ciudadanos. La polución acústica supone una importante amenaza del bienestar de los ciudadanos. Se estima que alrededor del 90 % de los residentes en la ciudad de Nueva York están expuestos a niveles de sonido que superan el umbral establecido, como perjudicial para la salud, por la Agencia de Protección Medio Ambiental (EPA). Además, se ha demostrado que esta exposición puede provocar trastornos del sueño, pérdida parcial del sonido, hipertensión y enfermedades del corazón. La lucha contra este problema no es trivial y se requiere de gran capacidad de monitorización y actuación para poder cubrir cualquier foco sonoro en la urbe [3, 4].

Con objeto de mejorar la calidad de vida de los ciudadanos y ofrecer un sistema que permita establecer un marco de aprendizaje continuo y desarrollo urbano, en este proyecto se ha propuesto un método de monitorización y clasificación de sonidos urbanos, que opera de forma autónoma, capaz de capturar sonido del entorno urbano clasificarlo en función de la fuente sonora que lo produzca y poder crear así un contexto en la red basado en datos obtenidos directamente de lugar.

Este sistema IoT puede ofrecer una perspectiva global de lo que ocurre en una zona de la ciudad, y poder ayudar en la toma de decisiones. Este tipo de soluciones presenta una gran ventaja debidas a su versatilidad, ya que se puede desplegar en zonas sin acceso a la red eléctrica teniendo en cuenta las características de bajo coste de los sensores. Además de poder realizar tareas de clasificación automáticas sin necesidad de supervisar el proceso, por lo que es una solución que cuenta con las ventajas de un sistema inteligente sobre una red de sensores de bajo coste.

Este proyecto, parte del trabajo realizado por los investigadores del proyecto *Sounds of New York City, SONYC*. A raíz de sus publicaciones hemos establecido un punto de partida, empleando su base de datos como referencia y creando modelos de aprendizaje basados en estos datos. Unas publicaciones que han servido de base para la confección de este proyecto, ofreciendo un punto de partida para definir la arquitectura y las bases de desarrollo del mismo.

Por consiguiente, parte de la naturaleza del desarrollo de este proyecto viene del estudio y análisis de diversas propuestas en el ámbito de desarrollo en smart cities. Con objeto de poder establecer estrategias y soluciones que mejoren la calidad de vida de los ciudadanos en tres aspectos: eficiencia, sostenibilidad y paz social. Estableciendo así un marco de desarrollo orientado al uso de la tecnología como medio para mejorar la capacidad de cambio de los núcleos urbanos y adaptarlos a los nuevos retos y necesidades de la humanidad y el planeta.

1.2. Objetivos

Inicialmente, se estableció como punto de partida replicar y testear el mismo proyecto que SONYC con objeto de crear un clasificador de sonidos urbanos basado en aprendizaje máquina (en inglés *machine learning, ML*) e inteligencia artificial. Con el estudio de sus publicaciones, se determinó la base y estructura del proyecto. En ese momento, se estableció la creación de un clasificador empleando con técnicas de *machine learning* que se pueda implementar sobre dispositivos de bajo coste.

De esta manera, se ha creado un sistema funcional cuyo objetivo es caracterizar y monitorizar de forma autónoma el sonido urbano de diferentes localizaciones dentro de una ciudad de forma totalmente automática, generando información de contexto y facilitando la creación de informes de situación para organismos de control y gestión urbana.

Los objetivos a alcanzar en este proyecto son los siguientes, divididos en tres bloques funcionales: diseño de un sistema de clasificación empleando una CNN, implementación del clasificador en una Raspberry Pi y diseño de la red de IoT (*Internet of Thing*) basado en el framework de Fiware, tal y como se puede apreciar en la Fig. 1.1.

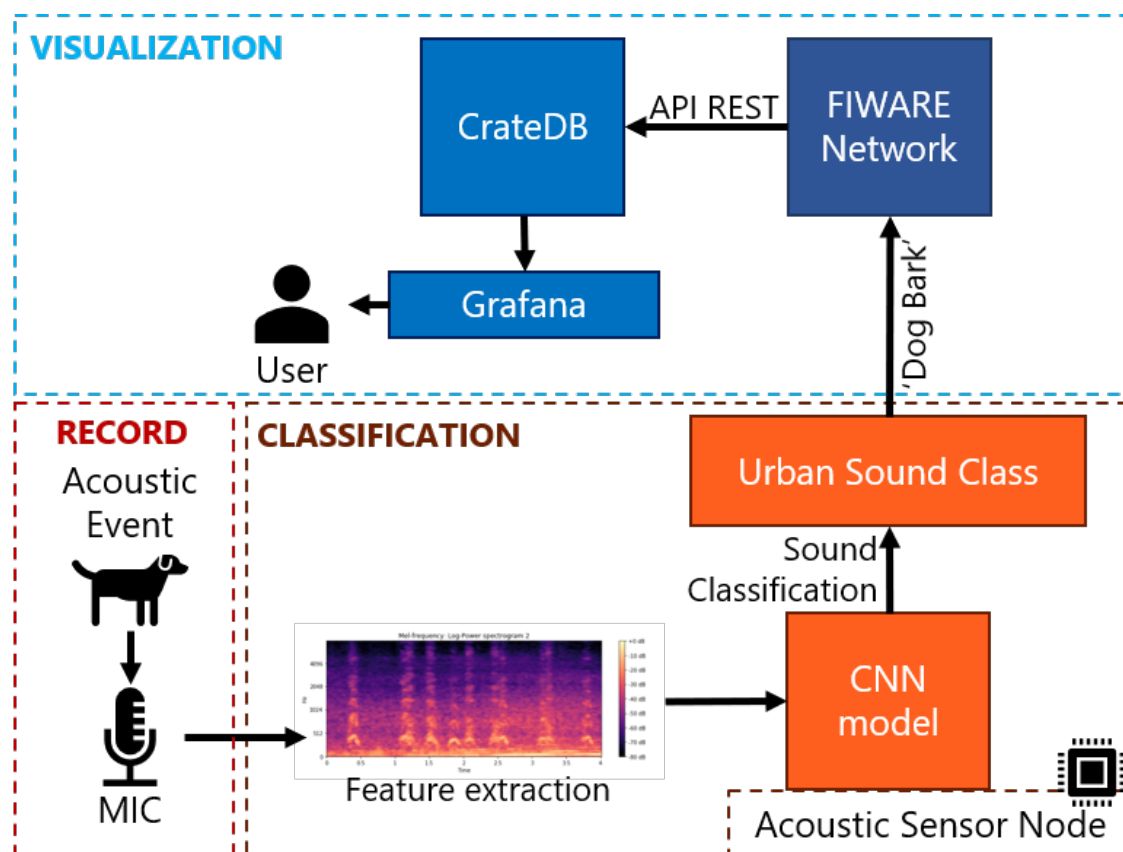


Figura 1.1: Esquema del Proyecto

1.2.1. Clasificador de eventos acústicos

Teniendo en cuenta la tipología y características del sonido urbano, observando las clases del clasificador y determinando los métodos adecuados para caracterizar cada fragmento de sonido, se requiere el diseño de una red neuronal capaz de identificar y discernir entre diferentes eventos acústicos, pudiendo así clasificar y registrar de forma autónoma la fuente sonora del evento acústico captado por el micrófono.

Observando publicaciones relacionadas con la clasificación de eventos acústicos, se espera obtener un margen de precisión en la clasificación alrededor del 71 %, y un framework de trabajo ágil y de fácil implementación en dispositivos de bajo coste.

1.2.2. Implementación en Raspberry Pi

Como objetivo de este bloque se prioriza la implementación del clasificador una vez entrenado en la Raspberry Pi, siendo capaz de ser ejecutado y cuyo funcionamiento sea óptimo ante nuevos eventos acústicos. Además de buscar alternativas para que el consumo de recursos por el clasificador, sea moderado y en tiempo real.

Partiendo de las limitaciones en hardware de la Raspberry Pi, se busca un consumo moderado de la CPU y de la memoria RAM optimizando la fase de clasificación, empleando una fase previa de

pre-detección que sirva de activador del clasificador. De esta manera se evita la ejecución continua del clasificador, y se filtra el contenido que se clasifica y que se está vertiendo en la red. Esta fase de pre-detección tiene como objetivo filtrar los eventos acústicos, según las necesidades de cada aplicación, teniendo en cuenta la energía del evento. De esta forma se clasificará aquellos eventos que cumplan una serie de características

1.2.3. Red IoT

Cualquier propuesta de sensores inteligentes, debe estar acompañada de una correcta configuración de red que gestione y coordine las funciones y datos generados por cada nodo de la misma. Por ello, se requiere el diseño de una arquitectura de red para sensores de monitorización que se adapte a un escenario actual de ciudad inteligente, capaz de sincronizar y gestionar el flujo de datos generados por los nodos de la red y hacerlos accesibles en todo momento. En esta línea, se optará por implementar esta característica haciendo uso de herramientas y frameworks que se emplean en la actualidad, como es el caso de Fiware.

Capítulo 2

Contexto y estado del arte

2.1. La Era Urbana Inteligente

Cuando hablamos de “urbano” nos referimos a la idea de un espacio de encuentro e intercambio en una comunidad de personas con una densidad de población elevada. Estos espacios se caracterizan principalmente por su capacidad de adaptación o resiliencia, con objeto de mantener un ritmo constante de circulación y flujo de actividad.

El origen urbano se remonta a la revolución del neolítico, en el que por primera vez nuestra especie se agrupa en comunidades numerosas con objeto de ser más fuertes y capaces de adaptarse a cualquier situación. Desde entonces, como seres sociales que somos, el intercambio y la comunicación ha sido nuestra moneda de cambio para vivir en conjunto y formar lo que conocemos actualmente como sociedad. El volumen de estas agrupaciones ha ido creciendo a la par junto con nuestros sistemas de comunicación y capacidad de intercambio. Hasta llegar, en la actualidad, a lo que llamamos sociedad globalizada.

La tecnología a lo largo de los años nos ha permitido alcanzar cualquier parte del mundo, permitiendo así conectar, comunicar e intercambiar conocimiento a través de todas las culturas y comunidades que habitan este planeta. Desde la posibilidad de desplazarse por tierra, mar y aire, hasta la capacidad de comunicarse a través de una red de comunicación mundial, Internet. Estos avances, nos han llevado hoy a lo que conocemos como sociedad digital, una sociedad que se caracteriza por su acceso a sistemas de comunicación inmediatos y capaces de alcanzar cualquier parte del mundo.

De esta forma, los nuevos sistemas de información han hecho posible mejorar y aumentar la capacidad de resiliencia de cualquier persona, comunidad o ciudad. Ya que la capacidad de acceso a la información, por lo consiguiente, al conocimiento es cada vez mayor. Es aquí, donde definimos y referenciamos al que identificamos como factor fundamental de desarrollo, y que rige las bases de cualquier sociedad y del ser humano como tal: la información como base del conocimiento mundial.

El motivo de esta introducción en el ámbito urbano actual, era asentar la idea y concepto de la información y el papel que juega sobre los conocidos procesos de aprendizaje. Ya que estos son la base de desarrollo de cualquier sistema social, así como para cualquier principio de aprendizaje máquina. Y es que en las últimas dos décadas, la sociedad ha crecido con la idea de crear nuevos

asentamientos sociales, en los que se tenga en cuenta la eficiencia, sostenibilidad y economía social. A raíz de esta necesidad, los sistemas de monitorización y sensorización están jugando un papel fundamental en los procesos de aprendizaje, ya que son la principal fuente de información que nos permite crear un contexto sobre cualquier escenario y situación.

Teniendo en cuenta lo anterior, introducimos dos conceptos que han sido fundamentales en el desarrollo de este proyecto. La idea de crear comunidades inteligentes o lo que se conoce actualmente como “smart cities” parte de la necesidad, mencionada anteriormente, de crear un nuevo marco de comunicación y desarrollo urbano hacia nuevos núcleos de convivencia que tengan en cuenta los tres componentes esenciales de nuestra sociedad: el medio ambiente, las personas y la tecnología. Estos tres aspectos de nuestra sociedad, cuando se cohesionan a través de sendas estrategias y planificaciones urbanas, permiten crear nuevos asentamientos sociales más eficientes, sostenibles y sobre todo más humanos. De forma paralela, con la idea de “Big Data” a través del cual se reconoce la capacidad de la red para el intercambio de grandes cantidad de datos desde cualquier fuente, ha hecho posible la viabilidad de sensorizar y monitorizar cualquier sistema o comunidad. De esta forma, partiendo de la cantidad de datos disponibles en la red más aquellos que se generan de forma puntual se puede crear soluciones que integren dichos datos y los conviertan de forma autónoma en fuentes de información y conocimiento, facilitando así la toma de decisiones hacia cualquier nueva situación o problema. Empleando la tecnología como medio para gestionar la transformación social y resiliencia de las comunidades.

En este contexto surge la necesidad de fusionar ambas ideas, con objeto de usar los datos como fuente de información para mejorar la calidad de vida de los espacios urbanos. Pudiendo así establecer un marco de desarrollo versátil y capaz de adaptarse al cambio, basado en el conocimiento y aprendizaje. Integrando la tecnología dentro del proceso de aprendizaje y desarrollo de la sociedad, facilitando y agilizando nuestra capacidad de adaptarnos al cambio. A esta idea se la conoce como, aprendizaje colectivo, cuyo término es acuñado por el proceso a través del cual un conjunto de personas que conviven en sociedad comparten información y conocimiento entre ellos para poder solventar y cubrir cada una de sus necesidades.

Con lo dicho, observamos que actualmente la sociedad ha sufrido una transformación que ha venido de mano de la información, en concreto debido a las nuevas tecnologías de la información que permiten acceder al conocimiento de forma inmediata y desde cualquier parte del mundo. Por este motivo, en este proyecto se han desarrollado dos áreas que juegan un papel fundamental en el desarrollo evolutivo de nuestra sociedad: por un lado está el papel de la información y los sistemas de comunicación, y por otro lado el papel del aprendizaje y conocimiento. De esta manera, este proyecto tiene el propósito de clarificar y mostrar con un ejemplo práctico, el papel de la tecnología en la actualidad para poder crear sistemas de información precisos y capaces de adaptarse a cualquier situación, que ofrezcan datos y con ellos conocimientos a aquellos que los utilicen.

2.1.1. Horizonte Europa

Tal y como se ha mencionado, la importancia de establecer una agenda de objetivos en común entre todos los países invoca la unión y alianza con un mismo fin, construyendo una comunidad más allá de las fronteras territoriales, y por lo que los ciudadanos de cada rincón del planeta se convierten en ciudadanos del mundo.

Este tipo de agendas implican un compromiso común universal, en los que cada país se enfrenta a retos específicos en su búsqueda del desarrollo sostenible dónde cada uno establecerá sus propias

metas nacionales siguiendo los Objetivos de Desarrollo Sostenible (ODS). Es en 2010 cuando la Unión Europea (UE) ya había establecido un marco de referencia con una agenda de desarrollo sostenible para el 2020. Esta estrategia se conoce como “Horizonte 2020”. En ella se fijan los objetivos y la agenda para el crecimiento y empleo para los países miembros de la UE. Esta agenda “H2020” se caracteriza porque apuesta por un crecimiento “inteligente”, sostenible e integrador como método para superar las deficientes estructura económica europea, mejorar su competitividad y productividad, así como sustentar una economía social de mercado sostenible. Actualmente, la sucesora de susodicha agenda se conoce como “Horizon Europe”, la cual se ha definido en el periodo de 2021 – 2027. Esta agenda tiene como objetivo mantener la línea de trabajo de “H2020” además de situar el foco en desafíos globales así como asegurar la competitividad industrial del entramado europeo.

En este marco de referencia, las ciudades han sido el objetivo principal, a nivel mundial, en las estrategias nacionales para el desarrollo sostenible. En España se cuenta con una red nacional de ciudades inteligentes (RECI), que se posiciona como una asociación de territorios locales cuyas entidades, son representativas de cada territorio y lideran los sistemas de innovación en su propio ámbito fomentando su propia red local de agentes vinculados al desarrollo de estos núcleos. Esta red, que surgió en 2014 junto con el plan de ciudades inteligentes, diseñó 5 grupos de trabajo por medio de los cuales cada territorio debía de desarrollar e investigar. Los grupos temáticos abarcan: el gobierno, economía y negocios, también se habla de la movilidad urbana, medio ambiente, infraestructuras y habitabilidad urbana, así como del sector energético y la innovación social. Esta asociación nace a raíz del Plan Nacional de Ciudades Inteligentes impulsado por el Ministerio de Energía, Turismo y Agenda Digital. Ambos grupos, de los que participa la secretaria de estado para la “Sociedad de la Información y la Agenda Digital y Red.es”, nace con el objeto de potenciar el empleo de las TIC en el desarrollo de las ciudades y territorios turísticos, para impulsar su transformación en destinos turísticos inteligentes y mejorar la sostenibilidad energética y medioambiental de la actividad turística de los mismos. Gracias a esta hoja de ruta nacional, en el caso de España, diversas ciudades han diseñado su plan estratégico para el desarrollo urbano inteligente de sus ciudades. En los que Barcelona, Bilbao, Vitoria, Madrid y Valencia que destacan como ciudades que promueven la creación de escenarios urbanos amables, atractivos para el turismo y favorables para el emprendimiento, donde la tecnología es protagonista en la innovación social abriendo nuevos horizontes en accesibilidad y participación ciudadana.

Ahora, la necesidad de conseguir un cambio socioeconómico en busca del bienestar social y medioambiental ha establecido un objetivo concreto. Desarrollar hoy, las ciudades del mañana. Como resultado del periodo del plan “H2020” (2015 – 2020), el 65 % de los países europeos involucrados piensan que “H2020” ha jugado un papel clave en la implementación de la estrategia europea hacia 2020 para el trabajo y el desarrollo inteligente, sostenible e inclusivo. Estos mismos indican que esta estrategia ha ayudado notablemente en el desarrollo de una sociedad y economía basada en el conocimiento y la innovación.

Se puede apreciar entonces, el papel que juega esta estrategia europea en el desarrollo de soluciones innovadoras para la mejora del bienestar social. Empleando el conocimiento y la innovación como medios de transformación, con marcos de desarrollo como en el que se enmarca este trabajo. [2]

2.1.2. Exposición del problema

Como se ha mencionado anteriormente, según las Naciones Unidas, el 55 % de la población mundial vive en áreas urbanas, pero se espera que esta proporción aumente a 68 % para 2050. Las ciudades enfrentarán cambios importantes en el futuro cercano, por lo que el desarrollo de las soluciones de ciudad inteligente serán un factor clave para llevar a cabo estrategias exitosas. En este sentido, comprender su entorno en cualquier momento del día o de la noche interpretando los datos obtenidos de los sensores desplegados sobre la ciudad será fundamental. Una fuente importante de información para conocer lo que ocurre en una ciudad son los sonidos, cuyo reconocimiento es el objetivo de las técnicas llamadas *environmental sound classification* (ESC). ESC es un área de investigación muy activa y sus aplicaciones son numerosas, la mayoría de ellas centradas en entornos urbanos, naturales y domésticos [5, 6, 7, 8].

Las redes de sensores acústicos inalámbricos (WASN) generalmente se implementan para capturar los sonidos cualquier sonido ambiente, clasificarlos y transmitirlos [9, 10]. En general, parte del procesamiento que realizan los sensores se realiza *in situ* en el nodo, pero las tareas de decisión e inteligencia suelen ser realizadas por la red mediante *cloud* (procesamiento remoto, en la nube) o *edge computing* (procesamiento en el nodo) [10, 11, 12]. En este proyecto, presentamos una red formada por sensores inteligentes capaces de verter datos contextualizados en la red [13, 14, 15], que es diferente de la visión de Internet-of-Things (IoT) donde los nodos graban el sonido y los retrasmiten a la red y la inteligencia es proporcionada por completo por la red.

Los nodos acústicos de la red presentada en este trabajo de fin de master son dispositivos Raspberry Pi cuyo costo es muy bajo, pero cuya unidad de procesamiento es tan potente como para reconocer los sonidos urbanos mediante una red neuronal convolucional profunda (CNN) [16, 17]. Una vez que se obtiene la salida de la clasificación, el nodo la transmite a través de una red basada en el estándar abierto FIWARE [18]. En el apartado 4 describiremos las diferentes partes de la red de bajo coste cuya principal novedad es realizar casi en tiempo real la clasificación de sonido en el dispositivo (Raspberry Pi).

2.1.3. Sounds of New York City

SONYC es el principal proyecto que ha motivado la realización y ejecución de este mismo proyecto. Todo empezó cuando el equipo detrás del proyecto SONYC se encontró con que la contaminación acústica es uno de los principales problemas de calidad de vida de los residentes urbanos de los Estados Unidos. Se ha estimado que 9 de cada 10 adultos en la ciudad de Nueva York (NYC) están expuestos a niveles de ruido excesivos, es decir, más allá del límite de lo que la EPA considera perjudicial. Cuando se aplica a ciudades estadounidenses de más de 4 millones de habitantes, tales estimaciones se extienden a más de 72 millones de residentes urbanos.

Apuntando a la mayor queja cívica de los neoyorquinos, *noise*, un equipo de científicos de la Universidad de Nueva York, que trabaja con colaboradores de la Universidad Estatal de Ohio, ha lanzado una iniciativa de investigación integral, la primera en su tipo, para comprender y abordar la contaminación acústica en Nueva York y más allá.

El proyecto, que involucra el monitoreo del ruido a gran escala, aprovecha lo último en tecnología de aprendizaje automático, análisis de big data y reportes de ciencia ciudadana para monitorear, analizar y mitigar de manera más efectiva la contaminación acústica urbana. Conocido como *Sounds of New York City* (SONYC), este proyecto de varios años ha recibido una subvención de

4,6\$ millones de la National Science Foundation y cuenta con el apoyo de las agencias de salud y ambientales de la Ciudad.

Los objetivos de SONYC son crear soluciones tecnológicas para: (1) el monitoreo sistemático y constante de la contaminación acústica a escala de ciudad; (2) la descripción precisa de entornos acústicos en términos de sus fuentes de composición; (3) ampliar la participación ciudadana en la notificación y mitigación del ruido; y (4) permitir que las agencias de la ciudad tomen acciones efectivas basadas en información para la mitigación del ruido.

Han planeado desde 2014, lograr estos objetivos implementando SONYC (Sonidos de la ciudad de Nueva York), el novedoso sistema ciberfísico que se describe arriba. Este sistema incluye una red híbrida distribuida de sensores y ciudadanos para informes de ruido a gran escala. En esta red, los ciudadanos cuentan con aplicaciones que les ayudan a conectarse con la ciudad y entre ellos de una manera eficaz y receptiva, mientras que los sensores utilizan métodos de escucha de máquinas de vanguardia para proporcionar constantemente una descripción detallada de su entorno acústico. La información de la red fluye a través de una ciberinfraestructura que analiza, recupera y visualiza datos para facilitar la identificación de patrones importantes de contaminación acústica: una especie de centro de control de misión”de ruido, destinado a los responsables de la toma de decisiones en las agencias de la ciudad para desplegar estratégicamente el recursos humanos a su disposición para actuar en el mundo físico.

Inspirado en el trabajo realizado en el proyecto SONYC [3, 13, 8, 9], se ha propuesto en este trabajo fin de master el desarrollo de un sistema de clasificación de sonidos urbanos empleando redes CNN, implementado en una Raspberry Pi como parte de una red de sensores de clasificación de sonidos urbanos autónomos.

2.2. Aprendizaje Automático

2.2.1. Procedimiento de Aprendizaje

La idea de aprendizaje máquina engloba un conjunto de técnicas y algoritmos para la extracción de información de datos, o para realizar una estimación en un sistema desconocido, haciendo uso de un número determinado de observaciones de entrada y salida.

La extracción de datos forma parte de la base de cualquier investigación científica y análisis. Las personas acumulamos información de nuestro entorno, la cual se convierte en conocimiento, organizamos esta información y buscamos patrones y explicaciones coherentes a las regularidades que hay en ella. Mediante un razonamiento inductivo, el científico determina una regla o teoría que explica las relaciones entre las observaciones y los patrones que la definen. Entre la inmensidad de posibles hechos y observaciones el científico selecciona aquellos que considera más relevante con cierto conocimiento subjetivo, o a priori. De este razonamiento, podemos observar que la ciencia evoluciona gracias a que se tienen en cuenta observaciones que previamente no se habían considerado y se descartan aquellas que se han considerado como irrelevantes. La clasificación en la ciencia no es un hecho estático que precede al descubrimiento de una teoría, sino que está íntimamente entrelazado con ella. Además, las clasificaciones correctas hacen posible descubrimientos y, a su vez, los descubrimientos hacen cambiar la forma de clasificar las cosas que estudiamos.

La cuestión del aprendizaje es sólo una parte del procedimiento que se emplea en todos los campos del conocimiento para extraer conclusiones ante cualquier situación o problema. El esquema clásico del procedimiento general para estimar o aprender un modelo de relación entre los datos observados es el siguiente:

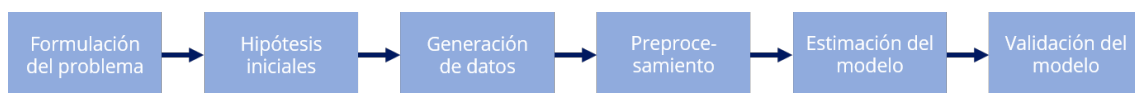


Figura 2.1: Esquema Proceso de Aprendizaje

A modo resumen, analizamos cada una de las partes de este proceso, ya que es la base por la que se define y construye el aprendizaje automático. Todo conocimiento surge de la observación e identificación de un problema. Este problema suele partir de una serie de hipótesis conocidas acerca del modelo que se desea encontrar. Las hipótesis se establecen a través de un conjunto de datos y observaciones desconocidas, o de las que se tiene algún conocimiento parcial, que se desea estimar a través de datos experimentales. Estos datos se suelen obtener de forma experimental o pueden ser recopilado tras observar el medio que se está estudiando. De esta forma el conjunto de datos en un aprendizaje se divide en dos subconjuntos: aquellos que se emplean para el entrenamiento o estimación del modelo, y los otros que se emplean para probar y testear el modelo determinado. En el aprendizaje los datos deben generarse de forma aleatoria de acuerdo a una función de probabilidad que debe ser desconocida, asegurando así la espontaneidad del sistema.

Las fuentes de información, así como aquellos datos que se usan para definir cualquier experiencia o conocimiento deben ser codificados de la forma más conveniente para su procesamiento. Es decir, cualquier fuente de información o conjunto de datos que se empleen para realizar observaciones y estudiar un problema, se deben elegir de forma precisa determinando aquellas características del conjunto de datos que más información aporte para nuestra experimentación. Este proceso se conoce como extracción de características y es un aspecto tan importante, que muchas veces el

éxito de una solución o aplicación se debe a la elección adecuada de las características que se emplean para codificar los datos. Una buena praxis para extraer características, es tener en cuenta que sean sencillas de extraer, que se invariante ante transformaciones irrelevantes para el problema, tengan insensibilidad al ruido y que contengan información relevante o discriminatoria. Dentro de un patrón, la extracción de características representa los procedimientos para obtener números o valores de los patrones que en cierta forma representan la información relevante contenida en los patrones.

Otra tarea usual en el preprocesamiento es la detección, esta tarea se centra en buscar características dentro del entorno monitorizado que cumplan una serie de restricciones y aspectos. Definiendo una serie de umbrales y aspectos a seguir, se puede crear un detector que obtenga la información del medio que cumpla con esas restricciones. La implementación de esta técnica representa un refuerzo en el proceso de extracción de características y en muchas ocasiones hace de filtro de información, dejando pasar aquello que cumpla con las características establecidas y pudiendo hacer el problema más pequeño. En el caso de este proyecto, la detección se ha empleado para clasificar aquellos eventos acústicos que cumplan con unas características acústicas particulares, y de esta forma mejorar el funcionamiento del clasificador, aumentando su precisión y su tiempo de procesado.

Una vez se haya especificado las características de los datos que son de relevancia en la correcta observación del estudio planteado, el siguiente paso consiste en establecer el modelo o también conocido como aprendizaje. En esta parte, se establece la forma en la que se relacionan cada aspecto observable buscando los patrones que los relacionan. De esta forma, se busca los algoritmos que mejor aproximen y se ajusten a los datos empleados en el estudio. En el proceso de selección del mejor modelo, siempre se realiza un problema de optimización, que ajusten dicho modelo a los datos empleados en el estudio. De esta forma, se obtiene un modelo que predice y aproxima cada datos de entrada a su semejante de salida, siguiendo los patrones establecidos en la fase de entrenamiento.

Finalmente, todo aprendizaje tiene una fase de validación o testeo, en la cual se determina el correcto funcionamiento del modelo. Tal y como se ha mencionado antes, los datos se separan en dos subconjuntos, los cuales no se pueden reutilizar en ambas partes de entrenamiento y validación, ya que puede provocar una situación de modelo “sobreajustado”. Es decir, que el modelo se ha ajustado específicamente a los patrones de comportamiento de los datos empleados en el entrenamiento, lo cual hace que el modelo no funcione correctamente ante nuevas entradas de datos o nuevos eventos.

2.2.2. Arquitectura y técnicas del Aprendizaje Automático

El aprendizaje automático es el estudio de técnicas que permiten que los ordenadores aprendan, basado en el desarrollo de algoritmos y modelos estadísticos que guían el aprendizaje del mismo. Esta disciplina forma parte de la rama de la inteligencia artificial, basada en la idea de que los sistemas computerizados pueden aprender a identificar patrones a partir de datos y observaciones cuantificables.

El principio del aprendizaje automático, se encuentra en 1949 de mano de Hebb, con la primera propuesta con la que se intenta aprender siguiendo el mecanismo biológico. Este dice que cuando dos neuronas están conectadas y están activas simultáneamente el peso sináptico que las une debería reforzarse, pero que si cuando una neurona está activa, la otra está inhibida, su peso

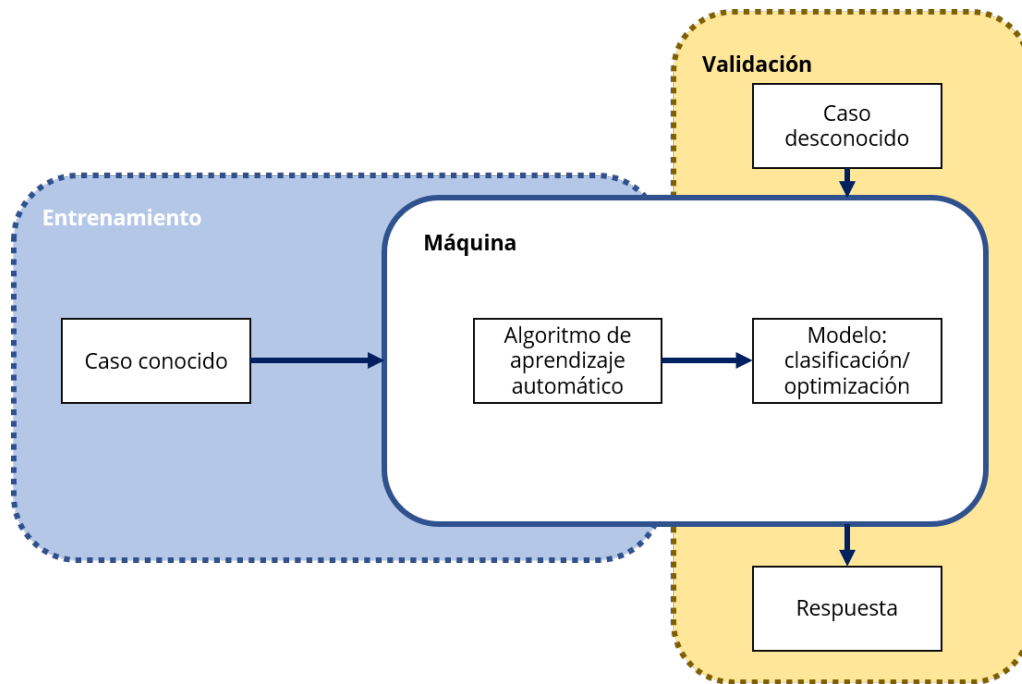


Figura 2.2: Esquema Aprendizaje Automático

debería disminuir. Esta formulación se resume matemáticamente como la regla de aprendizaje de Hebb, en la que se encuentra una ecuación que nos marque la variación del peso de conexión w_i dependiendo de los valores de las señales pre y postsinápticas x_i e y_i :

$$\Delta w_{ij} = \alpha \times x_i(k) \times y_j(k) \quad (2.1)$$

Dada la expresión, supongamos que valores positivos representan activación y valores negativos inhibición. Cuando la entrada x_i de la neurona está activa al igual que su salida y_i , se refuerza la conexión, se incrementa w_i , al igual que si las dos estuviesen inhibidas ya que están de acuerdo sus salidas. En resumen, si las dos magnitudes son del mismo signo (ya sea positivo o negativo) se refuerza la conexión y si son de signo contrario se debilita. Esta misma expresión se puede resumir de la siguiente manera: se dice que un programa aprende de una experiencia E con respecto a la tarea T con rendimiento P , si su rendimiento sobre la tarea T , medida por P , mejora la experiencia E .

Una tarea T describe cómo un sistema debe procesar un caso. El caso puede representar como un vector x en P^n , donde cada entrada x_i del vector es una característica, como los pixels generados por una imagen que representa el sonido basados en la percepción auditiva humana, también conocido como MFCCs. La medida del rendimiento P es específica para cada tipo de tarea T . Este se suele medir por medio de la validación del modelo con datos que no han formado parte del entrenamiento, o también conocidos como nuevos casos.

Existen diversas técnicas de aprendizaje automático según el tipo de experiencia E que emplean, de las cuales vamos a destacar principalmente el aprendizaje supervisado y no supervisado. El aprendizaje supervisado crea el modelo de aprendizaje en función a experiencias y patrones

ya conocidos, indicando en la base de dato lo que debería de generar la red a la salida, también conocida como salida deseada. Con esta información la red calcula cuál debe ser el cambio en sus parámetros para ajustar la salida a la esperada. Por otra parte el aprendizaje no supervisado crea un modelo de aprendizaje basado en un conjunto de datos de entrada desconocidos, tratados como variables aleatorias, por lo que la red no recibe información sobre si su respuesta es correcta o no a una entrada dada.

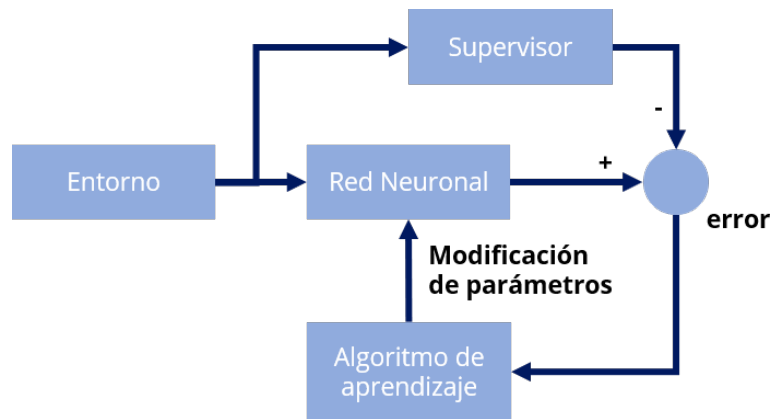


Figura 2.3: Esquema Aprendizaje Supervisado



Figura 2.4: Esquema Aprendizaje No Supervisado

2.2.3. Redes Neuronales

Las redes neuronales artificiales son estructuras paralelas inspiradas en las neuronas biológicas. Una red neuronal está compuesta por multitud de elementos simples, neuronas, interconectados de una forma más o menos densa y cuyo funcionamiento en conjunto puede dar lugar a un procesamiento no lineal complejo. Las redes son capaces de adaptarse y ajustar su comportamiento a partir de los datos y experiencias facilitados. Por este motivo, las redes neuronales son muy útiles en situaciones dónde la información es escasa o varía en el tiempo.

Para entender mejor qué es una red neuronal, analizamos la estructura básica de una neurona. Una neurona biológica consta de tres partes: las dendritas, el soma y el axión. Las dendritas se encargan de las transmisiones de señales eléctricas al cuerpo de la neurona, en la cual la suma de las entradas multiplicadas por sus pesos asociados determina el “impulso nervioso” que recibe la neurona. Este estímulo se procesa en el interior de la neurona mediante una función de activación que devuelve el impulso de salida hacia la siguiente neurona. Estas conexiones entre neuronas se realizan a través de los axones. Las redes neuronales artificiales siguen el mismo patrón de funcionamiento que las biológicas, las neuronas artificiales vienen representadas por nodos conectados entre sí, los cuales suman los valores de sus entradas teniendo en cuenta el peso de cada uno y en función del valor de entrada, se obtendrá un valor de salida que será utilizado para generar una señal que se transmitirá a otra neurona.

En general, una red neuronal puede ser considerada como un grafo dirigido y ponderado. En el que cada uno de sus nodos representa una neurona artificial. Si el arco es de entrada, este llevará asociado al mismo un valor o peso, grafo ponderado. Teniendo en cuenta estas características, existe una gran variedad de redes neuronales artificiales atendiendo a la forma en la que se conectan las neuronas, sus pesos y las capas que forman la red, entre otras cosas.

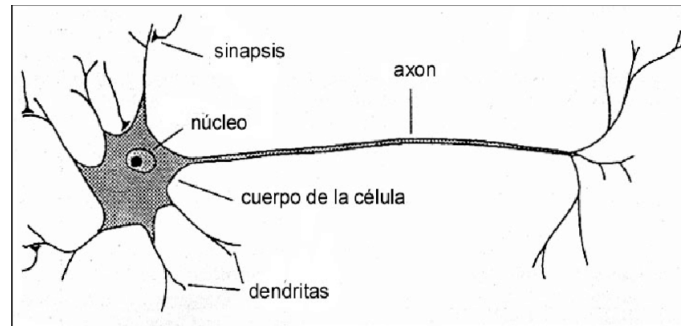


Figura 2.5: Esquema Neurona Biológica

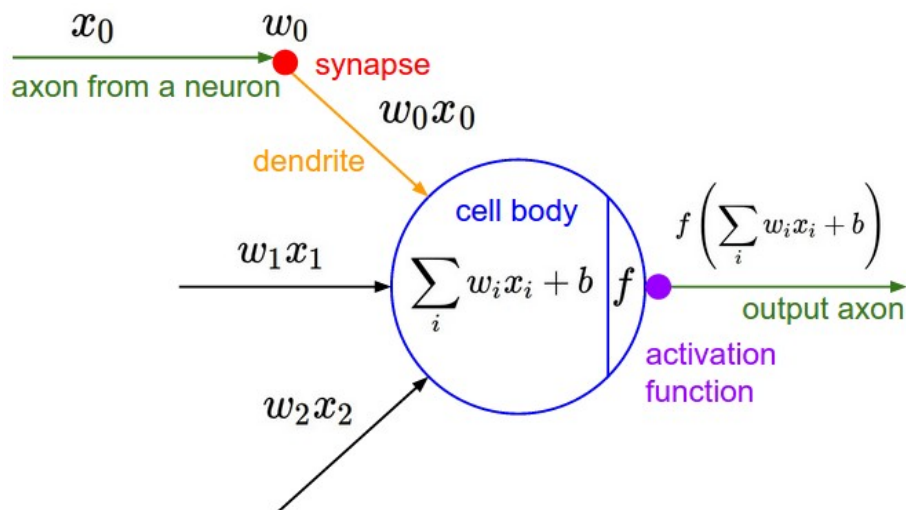


Figura 2.6: Esquema Neurona Artificial

Como hemos comentado, existen gran variedad de elementos y parámetros que caracterizan una red neuronal: el número de neuronas por capa, el grado, tipo de algoritmo de aprendizaje, función activación, función salida, umbral... A continuación explicamos sus características esenciales:

Conexiones entre neuronas

Las conexiones entre neuronas, son el equivalente a los axones, están definidas por los pesos o importancia de las conexiones entre dos neuronas i y j , y viene dado por w_{ij} . Estos mismos pueden tratarse de enlaces excitadores o inhibidores en función de si el peso es mayor o menor que 0 respectivamente. El potencial de entrada de una neurona, viene dado por la suma del conjunto de sus entradas. Una neurona estará activada cuando el potencial sináptico alcanza un determinado umbral.

$$cellbody = \sum_{i=1}^n w_{ij} \times x_i \quad (2.2)$$

Estado de activación

Una neurona puede estar en dos estados diferentes: reposo o excitación. Teniendo en cuenta que no se tiene en cuenta el estado anterior, el estado de activación de una neurona viene dado por la entrada actual. El estado de activación puede ser discreto o continuo, encontrándose abitualmente en los rangos $[-1,1]$ o $[0,1]$ respectivamente.

Función de activación

Teniendo en cuenta que una neurona solo se activa cuando se supera el umbral de entrada, permite evaluar el valor de entrada en la neurona para comprobar si su salida debe activarse o no. Esta tarea la realiza lo que se conoce como función de activación, que dependiendo del estado de activación de la neurona y de su entrada, modificará su salida. En la figura siguiente se puede observar las funciones de activación más comunes, algunas de las cuales se han empleado en el desarrollo del proyecto.

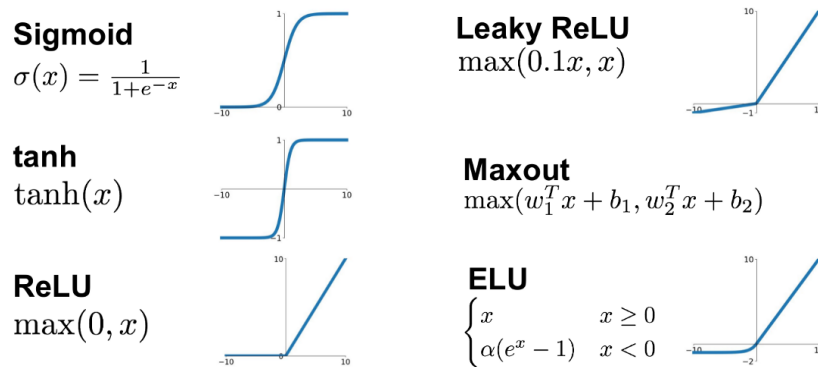


Figura 2.7: Ejemplos Funciones de Activación

Función salida

El último elemento del esquema, es la función de salida. Esta función se determina teniendo en cuenta la entrada a la misma y de si del estado de la neurona es activación o no. Con ella se determina el valor de salida de la neurona, con la cual se conectará con las siguientes neuronas.

Capas y conectividad neuronal

La organización de una red neuronal se basa en niveles o capas. Existen redes monocapa y redes multicapa. Las conexiones entre neuronas pueden realizarse en capas anteriores, posteriores, dentro de la misma capa o hacia si misma. De esta forma se organiza la red neuronal en tres niveles, en los cuales encontramos la capa de entrada, la capa oculta y la capa de salida.

- **Capa de entrada:** es la capa que recibe la información de entrada, en la cual se ajusta el vector de entrada a la red, con objeto de poder manipularlo correctamente.
- **Capa oculta:** es el conjunto de capas que forma parte de la red interna, no tiene contacto con el exterior y en las cuales se realiza las conexiones profundas entre las neuronas de la red.

- **Capa de salida:** esta capa ajusta la información generada por la red neuronal y la transfiere a la salida de la neurona.

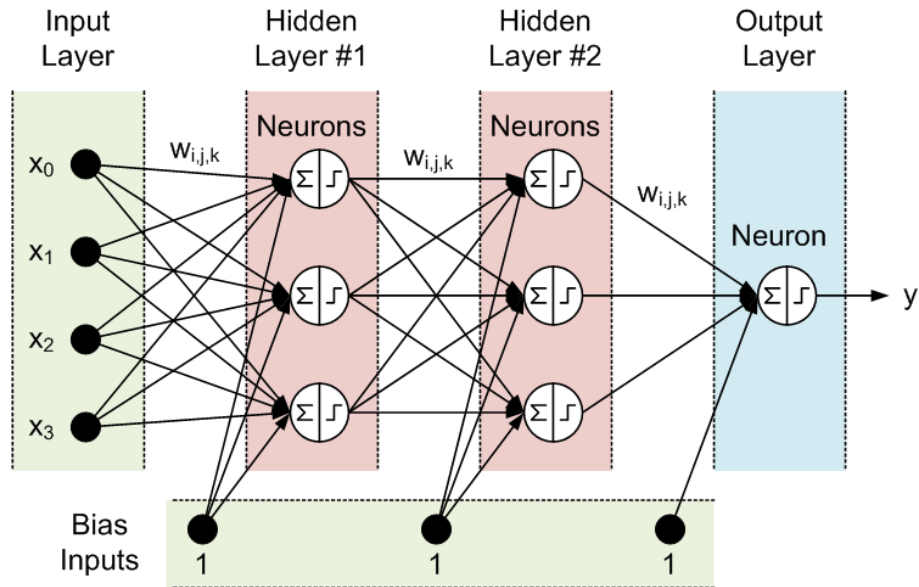


Figura 2.8: Esquema Red Neuronal

Existe una gran variedad de tipos de redes, y arquitecturas posibles. Pero principalmente podemos distinguir dos tipos de redes neuronales, que no cuentan con realimentación: las redes monocapa y multicapa.

- **Redes monocapa:** se establecen conexiones laterales entre las neuronas de una misma capa. Suele emplearse en tareas relacionales, o también conocido como auto-asociación. Por ejemplo, en caso de que la información esté incompleta, este tipo de redes completa la información que se ha recibido de forma distorsionada observando el contenido de cada neurona de la misma capa.
- **Redes multicapa:** disponen de neuronas agrupadas en distintos niveles. Cada capa se diferencia una de la otra, por las señales que recibe. Por lo que se establece un punto de partida, en la capa de entrada y así seguimos el camino por las diferentes capas hasta la capa de salida. Distinguimos así dos tipos de capa por las señales que reciben: “feedforward”, que sólo establece conexiones hacia delante y no existe conexiones hacia atrás, ni autorrecurrentes, ni laterales, como es el caso del Perceptrón multicapa. Por otro lado, se encuentra las redes “feedback” las cuales establecen conexiones hacia atrás, resultando útil para relacionar la información de entrada con la información de salida.

Ventajas de las redes neuronales

Las ventajas de las redes neuronales artificiales, son innumerables. La capacidad de gestión, predicción y adaptación al cambio es muy grande. A modo resumen, vamos a nombrar los aspectos más significativos de esta tecnología:

- **Aprendizaje adaptativo:** con la capacidad de aprender a realizar trabajos o tareas basándose en experiencias y datos conocidos.
- **Autoorganización:** son capaces de autogestionar su funcionamiento y comportamiento, debido a la realimentación y aprendizaje continuo al que se someten, adaptando en todo momento su capacidad a las necesidades.
- **Tolerancia a fallos:** debido a que se encuentra en constante aprendizaje, en caso de que se produzca un error del sistema o se genere un informe erróneo que inhabilite parte del sistema, la red podría seguir funcionando adaptándose a la capacidad actual del sistema, pero obteniendo el mismo resultado.
- **No linealidad:** debido a que es un sistema que se basa en la estadística y la capacidad de correlación entre diferentes datos, trabaja con mayor holgura y capacidad de predicción ante escenarios totalmente distintos, ya que no viene guiado por un sistema lineal que converge toda entrada a la misma salida.
- **Inteligente:** este tipo de redes, permite en la actualidad generar y diseñar operaciones con gran cantidad de datos para predecir y automatizar muchos procesos y tareas que agilizan la producción en cualquier ámbito de desarrollo.

2.2.4. Redes Neuronales Feed-Forward

Este tipo de redes son las más comunes, suelen emplear aprendizaje supervisado y como algoritmo de corrección del error utilizan la retropropagación o BackPropagation. Entre las redes de esta tipología más habituales vamos a comparar y mencionar las redes Multilayer Perceptron (MLP) y las Convolutional Neural Networks (CNN).

Estas redes se caracterizan por no tener ciclos en las capas, es decir, que todas sus conexiones se establecen entre neuronas de una capa a la siguiente. Si contase con ciclos de realimentación sería una red recurrente. A modo de ejemplo, nos ponemos en el caso de que una red tiene tres capas donde la capa 1 es la capa de entrada y la capa 3 la de salida. Una red de alimentación directa estaría estructurada de la siguiente forma: la capa 1 toma las entradas a la red, a su vez esta capa alimenta a la capa 2 y la capa 2 alimenta a la 3 donde se sitúan las salidas. Si fuese el caso de una red recurrente, la capa 1 alimenta a la 2, pero la capa 2 puede alimentar a la capa 1 y a la capa 3 al mismo tiempo, dado que la capa “inferior” alimenta sus salidas en una capa “superior”, crea un ciclo dentro de la red neuronal.

Como aclaración, las redes Feed Forward se estructuran hacia adelante, cosa que no tiene nada que ver con que empleen un algoritmo de aprendizaje que se llame “BackPropagation”. La propagación hacia atrás es el método por el cual se entrena una red neuronal. No tiene nada que ver con la estructura de la red, sino que actúa sobre la forma en la que se modifican los pesos de las conexiones entre las neuronas. Al entrenar una red Feed Forward, la información pasa a la red y la clasificación resultante se compara con la muestra de entrenamiento conocida. Si la clasificación de la red es incorrecta, los pesos se ajustan hacia atrás a través de la red en la dirección que le daría la clasificación correcta. Es este el momento en el que la red se propaga hacia atrás como parte de su entrenamiento, cosa que no tiene nada que ver con la tipología de la red neuronal. Por este motivo, tanto la MLP como la CNN son redes Feed-Forward, pero que se entrenan a través de la propagación inversa o “BackPropagation”.

El proceso de aprendizaje se realiza en tres fases:

- **Feed-Forward:** en la primera fase la capa de entrada se alimenta con datos etiquetados y supervisados, y se recoge la información emitida por la capa de salida para el reajustes de pesos de los enlaces.
- **Error Function:** en segundo lugar se calcula el error obtenidos entre el valor calculado por la red neuronal y el valor deseado.
- **BackPropagation:** finalmente, según el valor obtenido en la función de error, se ajusta los pesos de aquellas neuronas que han dado pie a una respuesta incorrecta.

2.2.4.1. Convolutional Neural Networks (CNN)

Cuando hablamos del procesado de imagen, las redes neuronales como la MLP presentan muchos inconvenientes. Estas redes emplean un perceptrón por cada entrada (e.j un píxel de una imagen, multiplicado por 3 en el caso de RGB). La cantidad de pesos de los nodos de la red rápidamente conlleva una gran carga de cómputo para imágenes de gran tamaño. Para una imagen de tamaño $224 \times 224 \times 3$ hay un total de 150,000 pesos que deben ser entrenados. Como resultado, surgen dificultades durante el entrenamiento pudiendo producir casos de *overfitting*.

Otro problema común en las redes MLP es que reaccionan de forma diferentes a cambios en las imágenes, por ejemplo, si un objeto se sitúa en la parte superior de la imagen y en otra imagen el mismo objeto se sitúa en la parte inferior, la red neuronal intentará corregir este error en vez de identificar los patrones de igualdad en ambas imágenes. Claramente, este tipo de redes no son la mejor idea para el procesado de imagen. Uno de los problemas principales es que se pierde información espacial cuando las imágenes se introducen en una red MLP. Los nodos que se encuentran próximo los unos a los otros son fundamentales para ayudar a caracterizar las imágenes que se introducen como entrada a la red. Por este motivo, se necesita un método para aprovechar la correlación espacial de los píxeles de la imagen de tal manera que se pueda identificar el objeto de la imagen sin importar dónde se sitúa espacialmente sobre la imagen.

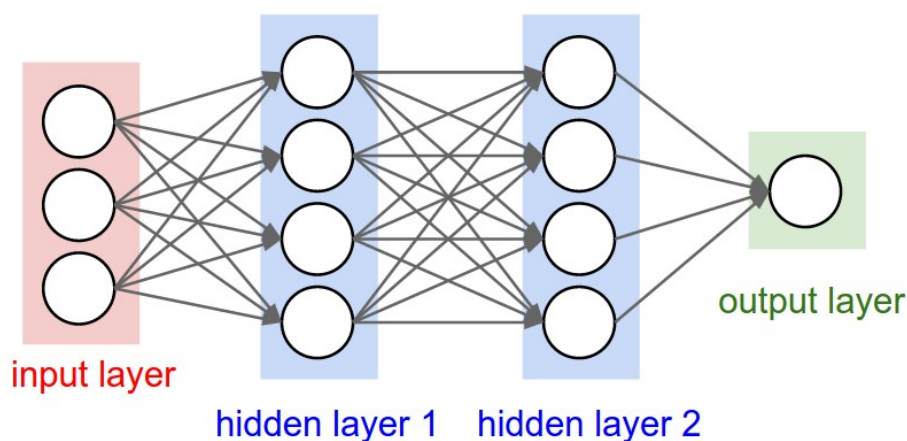


Figura 2.9: Red Neuronal Perceptrón Multicapa (MLP) [19]

Es en este caso dónde se introduce la utilidad y funcionalidad de las redes CNN. Los filtros han jugado un papel fundamental en el procesado de imágenes, para poder obtener la influencia que

generan los píxeles uno a los otros. Tal y como indica la palabra, el filtro se encarga de analizar un conjunto de elementos que se encuentran próximos unos a los otros. Se define un tamaño del filtro y se desplaza este desde la parte superior izquierda a la parte inferior derecha por toda la imagen para caracterizarla. Para cada punto de la imagen se obtiene un valor calculado por el filtro usando una operación convolucional.

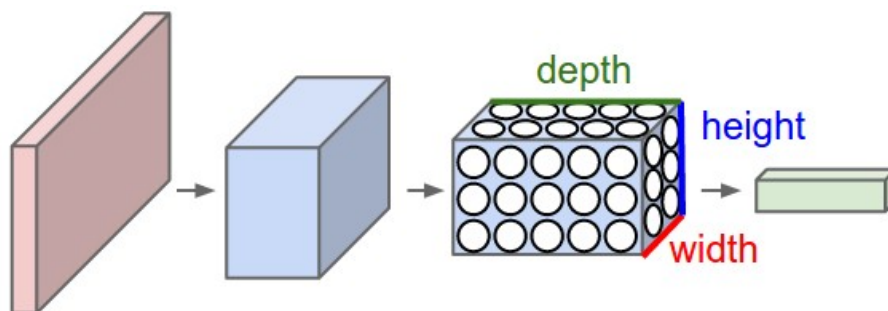


Figura 2.10: Red Neuronal Convolutiva (CNN) [19]

Después de que se usa el filtro sobre la imagen, se crea un mapa de características de la propia imagen, también conocido como “feature map”. Estos son tomados por la función de activación, la cual se encargará de situar cada característica de forma espacial sobre la imagen. En el caso de que se emplee una capa *pooling* esta se encargará de seleccionar los valores más altos dentro del *feature map* y usar estos como salida de esa capa neuronal. De esta forma se podrán identificar fácilmente aquellos valores que sean atípicos o *outliers*.

A pesar de sus similitudes, las CNNs y MLPs se diferencian en la forma en la que están conectadas sus neuronas en las diferentes capas. Mientras que las redes MLPs se tratan de redes *fully connected*, las redes CNN son redes *sparse connected*. Es decir, las redes Perceptrón Multicapa presentan 3 niveles, al igual que muchas CNNs, con la diferencia de que las neuronas de cada capa se encuentran totalmente conectadas una respecto a las otras, mientras que en la CNN las capas están parcialmente conectadas. Es decir, las CNNs emplean filtros, conjunto de pesos en forma de cubos que se aplican sobre toda la imagen. Cada segmento 2D de los filtros se conoce como *kernels*. Estos filtros introducen la invariancia de la imagen y la similitud de parámetros.

En resumidas cuentas la arquitectura CNN, es una lista de capas que transforman el volumen de la imagen en un volumen de salida de una forma más simple y optimizada teniendo en cuenta toda la información de la imagen.

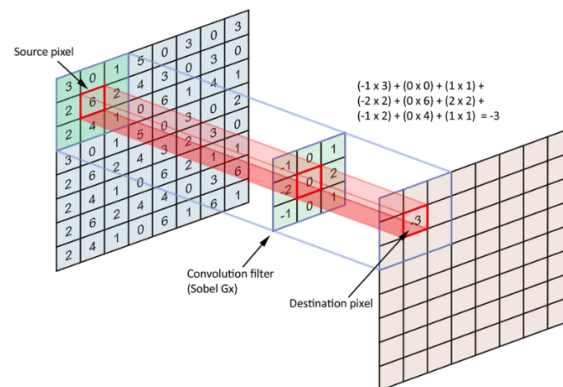


Figura 2.11: Ejemplo de cómo se aplica la convolución en imágenes usando "kernel filters"[20]

Capítulo 3

Componentes

3.1. Entorno de programación

En este proyecto se han implementado técnicas para clasificar el sonido urbano mediante el aprendizaje automático. La clasificación del sonido es bastante diferente a otras fuentes de datos. En este trabajo, primero veremos qué características se pueden extraer de los datos de sonido y qué tan fácil es extraer tales características en Python usando una librería de código abierto llamada Librosa principalmente.

Para el desarrollo de este proyecto, se han utilizado las siguiente herramientas:

- Tensorflow version 2.14, 1.14 (Raspberry Pi)
- Keras
- Python version 3.6
- Anaconda version 3.6
- Jupyter Notebooks
- Librosa
- Numpy
- Matplotlib
- glob
- os
- pandas
- scikit-learn

La programación de la red neuronal se ha realizado en un XPS 15 con Windows 10. Todo el proyecto se ha realizado empleando el entorno de programación **Python**. Para ello, se ha utilizado

el framework de *Anaconda* que instalar todo el entorno necesario para operar correctamente en Python. En primer lugar, en este entorno de programación se han optado por los entornos virtuales, para poder gestionar de manera más cómoda la instalación de las librerías necesarias para cada aplicación.

Entorno Virtual:

Desde el terminal de Anaconda, se crea un entorno virtual para TensorFlow. En nuestro caso se ha empleado la versión 2 de tensorflow y se ha empleado la CPU en vez de una tarjeta gráfica para su procesado.

Crear entorno virtual e instalar dependencias

```
# Crear un entorno virtual en anaconda
conda create -n tensorflow pip python=3.8

# Se llama tensorflow el entorno virtual , lo activamos
conda activate tensorflow

# Dentro del entorno virtual instalamos las librerías necesarias

(tensorflow) C:\Users\sglvladi> pip install --ignore-installed --upgrade
tensorflow==2.2.0

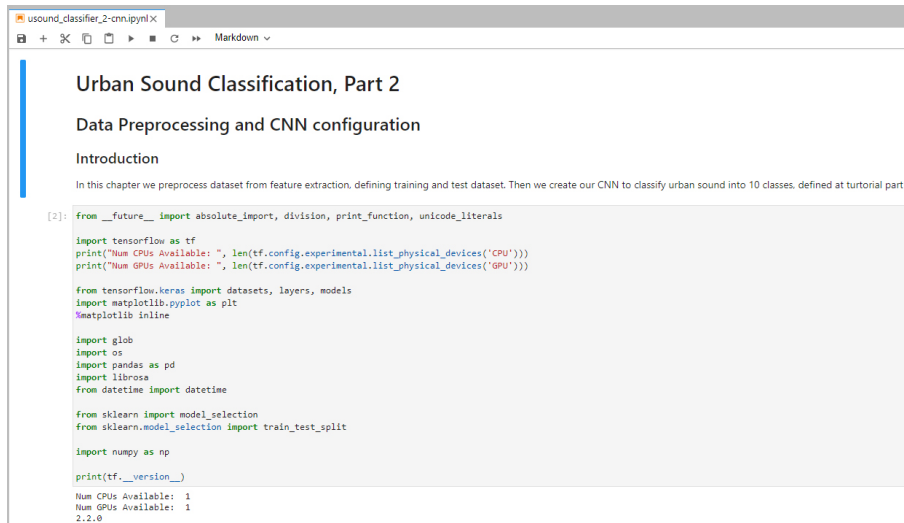
# Instalar Jupyter Notebooks
(tensorflow) C:\Users\sglvladi> pip install jupyterlab
```

Tensorflow: TensorFlow es una biblioteca de software gratuita y de código abierto para el flujo de datos y la programación diferenciable en una variedad de tareas. Es una biblioteca matemática simbólica y también se utiliza para aplicaciones de aprendizaje automático como las redes neuronales. En este proyecto se han empleado dos versiones diferentes del mismo software. Para la creación de la red neuronal y entrenamiento del modelo que se ha realizado en el PC, se ha empleado la versión 2,2. Por otra parte, en arm y para Raspberry Pi 3.2 la versión 2,2 no es compatible, por lo que se ha empleado la versión 1,14 que sí es compatible con esa distribución. Tras la comprobación y ejecución del proyecto en ambos dispositivos se ha comprobado que no ha habido un problema de compatibilidad en los servicios que se han empleado en cada versión. Ya que en el caso de la Raspberry Pi sólo se emplea Tensorflow para cargar el modelo ya entrenado en el PC.

Keras: Keras is an open-source neural-network library written in Python. It is capable of running on top of TensorFlow, Microsoft Cognitive Toolkit, R, Theano, or PlaidML. Designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible.

En Python existe un framework de trabajo que permite ejecutar código en tiempo real y poder crear libretas de trabajo con gráficas y segmentos de programación ejecutables. Este entorno se conoce como Jupyter Notebook. Project Jupyter es una organización sin fines de lucro creada para "desarrollar software de código abierto, estándares abiertos y servicios para computación interactiva en docenas de lenguajes de programación". Este entorno es muy similar a lo que se conoce como Matlab.

Instalar las librerías



```

Urban Sound Classification, Part 2
Data Preprocessing and CNN configuration
Introduction
In this chapter we preprocess dataset from feature extraction, defining training and test dataset. Then we create our CNN to classify urban sound into 10 classes, defined at tutorial part 1.

[2]: from __future__ import absolute_import, division, print_function, unicode_literals

import tensorflow as tf
print("Num CPUs Available: ", len(tf.config.experimental.list_physical_devices('CPU')))
print("Num GPUs Available: ", len(tf.config.experimental.list_physical_devices('GPU')))

from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
%matplotlib inline

import glob
import os
import pandas as pd
import librosa
from datetime import datetime

from sklearn import model_selection
from sklearn.model_selection import train_test_split

import numpy as np

print(tf.__version__)

Num CPUs Available: 1
Num GPUs Available: 1
2.2.0

```

Figura 3.1: Entorno de desarrollo Jupyter Notebook

```

# Instalar Librosa
(tensorflow) C:\Users\sglvjadi> pip install librosa

# Instalar Numpy para creación de arrays
(tensorflow) C:\Users\sglvjadi> pip install numpy

# Instalar Matplotlib para visualizar gráficas
(tensorflow) C:\Users\sglvjadi> pip install matplotlib

# Instalar Pandas
(tensorflow) C:\Users\sglvjadi> pip install pandas

# Instalar glob
(tensorflow) C:\Users\sglvjadi> pip install glob

# Instalar glob
(tensorflow) C:\Users\sglvjadi> pip install os

# Instalar scikit-learn
(tensorflow) C:\Users\sglvjadi> pip install scikit-learn

```

Librosa es un paquete de Python para análisis de audio y música. Proporciona los componentes básicos necesarios para crear sistemas de recuperación de información musical. En nuestro caso se ha empleado en la extracción de características usando la función de *melspectrogram*, entre otras.

Numpy es una biblioteca para el lenguaje de programación Python, que agrega soporte para matrices y matrices grandes y multidimensionales, junto con una gran colección de funciones matemáticas de alto nivel para operar en estas matrices.

Matplotlib es una biblioteca de trazado para el lenguaje de programación Python y su extensión matemática numérica NumPy.

Pandas es una biblioteca de software escrita para el lenguaje de programación Python para la manipulación y análisis de datos. En particular, ofrece estructuras de datos y operaciones para manipular tablas numéricas y series de tiempo.

glob El módulo `glob` encuentra todos los nombres de ruta que coinciden con un patrón especificado de acuerdo con las reglas utilizadas por el shell de Unix

scikit-learn es una biblioteca de aprendizaje automático de software gratuito para Python. Cuenta con varios algoritmos de clasificación, regresión y agrupamiento, entre otros.

Una vez se han instalado todas estas librerías, se emplea el entorno de python para ejecutar los scripts de cada función `.py`. Desde el terminal realizamos una ejecución del estilo `python prueba.py` y obtenemos por el terminal el resultado del mismo.

Además de la instalación del framework y librerías que se han empleado para el desarrollo de este proyecto, también se ha empleado los servicios de contenerización de Docker para el despliegue de los servicios de red y aplicación. Esta técnica es muy útil para poder desplegar una serie de servicios en cualquier dispositivo con sólo el despliegue de los contenedores sin tener que instalar ninguno de esos servicios. En nuestro caso sólo se ha empleado esta herramienta para el despliegue de la red FIWARE y los servicios de MongoDB, CrateDB y Grafana en el servidor para que de forma automática se despliegue toda la red IoT del proyecto.

Docker es un conjunto de productos de plataforma como servicio que utilizan la virtualización a nivel de sistema operativo para entregar software en paquetes llamados contenedores. Los contenedores están aislados entre sí y agrupan su propio software, bibliotecas y archivos de configuración; pueden comunicarse entre sí a través de canales bien definidos. En este proyecto se ha empleado en concreto una herramienta de docker que se conoce **Docker-Compose**. **Compose** es una herramienta para definir y ejecutar aplicaciones Docker de varios contenedores. Con Compose, usa un archivo YAML para configurar los servicios de su aplicación. Luego, con un solo comando, crea e inicia todos los servicios desde su configuración. Esta herramienta nos ha permitido desplegar todos los servicios que requeríamos para el funcionamiento de la red y aplicación con sólo una imagen y por ello un simple comando de despliegue de esa imagen.

Los pasos que se han seguido para poder emplear esta herramienta, ha sido simplemente instalar el cliente de docker en el dispositivo en el que se va a desplegar los servicios de red, en nuestro caso el PC de la red local (LAN) que va a hacer de servidor. Una vez instalado, sólo habrá que desplegar la imagen `.yaml` en la que se especifica correctamente cada uno de los servicios que se van a ejecutar de forma simultánea y en diferentes contenedores. Si se requiere para el proceso, sólo habrá que para el despliegue o simplemente borrar cada uno de los contenedores que se han creado con la ejecución de la imagen.

En cuanto a la arquitectura de red, se ha empleado FIWARE. La Comunidad FIWARE es una Comunidad abierta e independiente cuyos miembros están comprometidos a materializar la misión FIWARE, es decir: “construir un ecosistema sostenible abierto alrededor de estándares de plataforma de software públicos, libres de regalías e impulsados por la implementación que facilitarán el desarrollo de nuevas aplicaciones inteligentes en múltiples sectores”. La Comunidad FIWARE no solo está formada por contribuyentes a la tecnología (la plataforma FIWARE), sino también por aquellos que contribuyen a construir el ecosistema FIWARE y hacerlo sostenible en el tiempo. Como tal, las personas y organizaciones que comprometen recursos relevantes en las actividades de FIWARE Lab o las actividades de los programas FIWARE Accelerator, FIWARE Mundus o FIWARE iHubs también se consideran miembros de la comunidad FIWARE. Sin entrar en mucho detalle, en este proyecto se han empleado los componentes de FIWARE Orion Context Broker y QuantumLeap para poder crear un modelo de suscripción que además pueda almacenar un histórico de datos que se envían a través del broker. En la sección 4.4, se especifica en mayor detalle

los elementos de la red FIWARE y su función en la red teniendo en cuenta la Arquitectura.

El último componente de software a tener en cuenta es el sistema operativo empleado en la Raspberry Pi. En concreto se ha instalado Raspbian distribución 2020.

3.2. Hardware

Para la realización de este proyecto se han empleado hasta 3 dispositivos distintos a través de los cuales se ha podido implementar todo el sistema. Por una parte describiremos las características del ordenador empleado para el entrenamiento y desarrollo de la red neuronal encargada de la clasificación de los fragmentos acústicos, y por otra parte se encuentra las características de la Raspberry Pi que se han empleado como nodo sensor y que empleará una interfaz acústica (micrófono) para poder captar los eventos acústicos que se quieren clasificar.

La raspberry que se ha empleado se trata de una Raspberry Pi Model 3B. La Raspberry Pi 3 Model B es el primer modelo de la Raspberry Pi de tercera generación. Reemplazó a la Raspberry Pi 2 Modelo B en febrero de 2016. Consulte también la Raspberry Pi 3 Modelo B +, el último producto de la gama Raspberry Pi 3.

- CPU de cuatro núcleos a 1,2 GHz Broadcom BCM2837 de 64 bits
- 1 GB de RAM
- BCM43438 LAN inalámbrica y Bluetooth de baja energía (BLE) a bordo
- 100 Base Ethernet
- GPIO extendido de 40 pines
- 4 puertos USB 2
- Salida estéreo de 4 polos y puerto de video compuesto
- HDMI
- Puerto de cámara CSI para conectar una cámara Raspberry Pi
- Puerto de pantalla DSI para conectar una pantalla táctil Raspberry Pi
- Puerto micro SD para cargar su sistema operativo y almacenar datos
- Fuente de alimentación micro USB conmutada mejorada de hasta 2,5 A
- Precio: 40 Euros

El UMIK-1 es un micrófono de medición USB omnidireccional que proporciona medición acústica Plug Play. Desde la medición de la acústica del altavoz y de la sala hasta la grabación, este micrófono proporciona un bajo nivel de ruido y resultados precisos en los que puede confiar. Olvídense de la instalación de controladores, la compatibilidad del sistema operativo y los micrófonos no calibrados. El Umik1 es un dispositivo USB Audio clase 1 reconocido automáticamente por

todos los sistemas operativos (Windows / Mac) e Ipad con kit de conexión de cámara. Se proporciona con un archivo de calibración único basado en el número de serie para asegurar una medición precisa. Combinado con nuestro software de medición acústica recomendado, es la combinación perfecta para un sistema de medición plug play.

- Compatible con Windows / Mac / Linux
- Compatible con todo el software de medición
- Funciones especiales como la medición de SPL calibrada habilitada con el software gratuito Room EQ Wizard (REW)
- Respuesta de frecuencia: 20 Hz - 20 kHz +/- 1 dB con calibración cargada
- Audio USB: Interfaz sin controlador USB Audio clase 1.0 para Windows, Mac y Linux
- Precio: 70 Euros

En cuanto al ordenador, se ha utilizado un XPS 15 9550, del año 2016, con las siguientes prestaciones:

- Screen: 15.6-inch 3840 x 2160 (UHD) touch screen, IGZO IPS
- Processor: Intel Core i5-6300HQ 2.3-3.2GHz
- Graphics: Nvidia GeForce GTX 960M (2GB GDDR5)
- Memory: 16GB DDR4@ 2133MHz
- Storage: Samsung PM951 NVMe m.2 512 GB SSD
- Connectivity: Wi-Fi 802.11ac (DW1830, Broadcom 3x3, max speed 1.3Gbps), Bluetooth 4.1
- Ports: 1x Thunderbolt 3/USB-C 3.1, 2x USB 3.0, HDMI, 3.5 mm audio, SD card reader
- Battery: 6-cell (85Wh)
- Operating system: Windows 10
- Size: (H) 357 x (W) 235 x (D) 17-11mm
- Weight: as configured 2 kg
- Prize: 1579 Euros

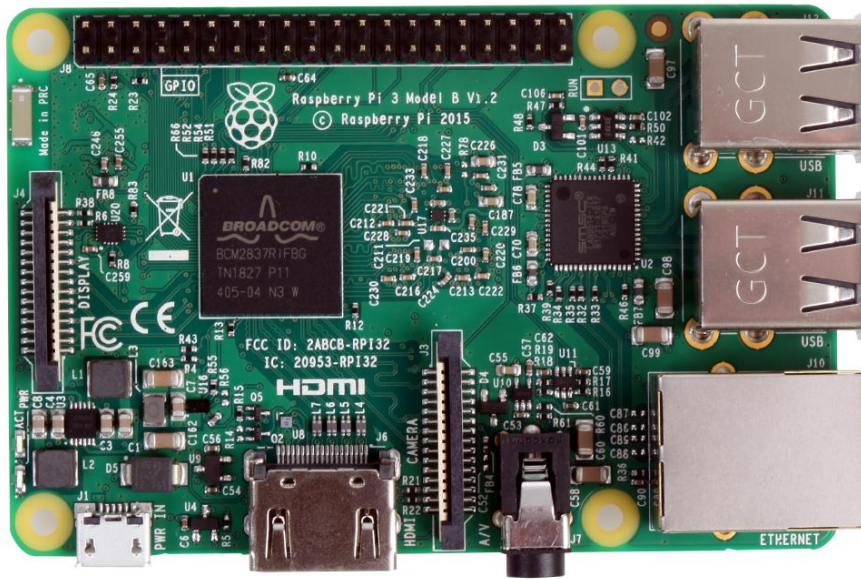


Figura 3.2: Imagen de una Raspberry Pi 3 Model B



Figura 3.3: Imagen de un Micrófono mini DSP

3.3. Dataset

En esta experiencia nos enfocamos en desarrollar un modelo de red neuronal convolucional capaz de clasificar automáticamente los diferentes sonidos urbanos. Para entrenar y trabajar este proyecto, se ha utilizado el conjunto de datos de sonido urbano UbanSound8K publicado por los investigadores del proyecto SONYC [3]. Contiene 8,732 clips de sonido etiquetados, cuya duración no supera los cuatro segundos, en diez clases diferentes según su publicación Urban Sound

Taxonomy:

- Aire acondicionado, *classID* = 0
- Claxon de coche, *classID* = 1
- Niños jugando, *classID* = 2
- Ladrillo de perro, *classID* = 3
- Perforador, *classID* = 4
- Motor en marcha, *classID* = 5
- Disparo, *classID* = 6
- Martillo neumático, *classID* = 7
- Sirena, *classID* = 8
- Música en la calle, *classID* = 9

El indicador de clase, es el que se emplea en toda la programación para el entrenamiento de la red neuronal, esta etiqueta es la que se indica con *classID*. Este conjunto de datos está disponible de forma gratuita en este enlace, **UrbanSound8K**.

Cuando descargue el conjunto de datos, obtendrá un archivo comprimido '.tar.gz' (distribución de compresión UNIX), desde Windows puede usar programas como 7-zip para descomprimir el archivo.

Ese archivo contiene dos directorios diferentes, en uno de ellos puede encontrar información sobre la clasificación de fragmentos de audio en un archivo de metadatos 'UrbanSound8K.csv'. El otro directorio contiene los segmentos de audio divididos en 10 bloques diferentes no clasificados por clases. Finalmente, los datos de audio se distribuyen como:

- **slice-file-name:** El nombre del archivo de audio. El nombre toma el siguiente formato: fsID-classID-occurrenceID-sliceID.wav
- **fsID:** el ID de Freesound de la grabación de la que se tomó este extracto (porción)
- **start:** La hora de inicio del segmento en la grabación original de Freesound
- **end:** La hora de finalización del corte en la grabación original de Freesound
- **salience:** Clasificación subjetiva, por escucha, indicando en que plano se encuentra la clase identificada dentro del fragmento. 1 = primer plano, 2 = fondo.
- **fold:** El numero de fold al que pertenece este fragmento.
- **classID:** Un identificador numérico del 1 – 10 que identifica la clase a la que pertenece.
- **class:** El nombre de la clase.

source: J. Salamon, C. Jacoby and J. P. Bello, "A Dataset and Taxonomy for Urban Sound Research", 22nd ACM International Conference on Multimedia, Orlando USA, Nov. 2014.

Capítulo 4

Demostrador

4.1. Arquitectura

En este proyecto se ha realizado un clasificador de sonidos urbanos implementando un modelo CNN en una Raspberry Pi, además de estar conectado a una red de sensores en FIWARE. Tal y como se puede observar en el esquema por un lado se encuentra el nodo sensor que consiste en la Raspberry Pi en el que se captura un evento acústico, se clasifica y se envía dicha información a la red FIWARE. Esta información se almacena en la entidad correspondiente a la que pertenece ese sensor dentro del Broker, y al mismo tiempo se almacena esa información de forma persistente en la base de datos de QuantumLeap (CrateDB) para poder realizar un histórico de los cambios. Esta información persistente se visualiza a través de la herramienta Grafana, en la que se ha creado un Dashboard que utiliza todos los datos que se han obtenido de la entidad almacenados en la red QuantumLeap.

El desarrollo de este proyecto se ha realizado en una red local (LAN, local area network), por lo que todos los nodos de la red se encuentran alojados en la misma red y conectado directamente a esta. Es decir, en este proyecto sólo se ha evaluado el rendimiento y funcionamiento de la red y de los nodos actuando como clasificadores en el nodo sin necesidad de enviar los fragmentos de audio por la red para clasificarlos en el servidor. Por este motivo, no se han implementado soluciones de cloud ni servidores privados para alojar el servidor de la red y que los sensores necesiten un protocolo de seguridad para la transmisión de los datos de clasificación. En las Fig. 4.1, Fig. 4.2 y Fig. 4.3 se puede observar el esquema del proyecto así como el prototipo desarrollado para el mismo. En la primera imagen se aprecia el esquema conceptual de todo el flujo de tareas del proyecto, y en las dos ilustraciones siguiente se observa el resultado del prototipo creado en el proyecto. Se observa la raspberry pi que se usa como nodo conectado a un micrófono a través del cual se captura los eventos acústicos.

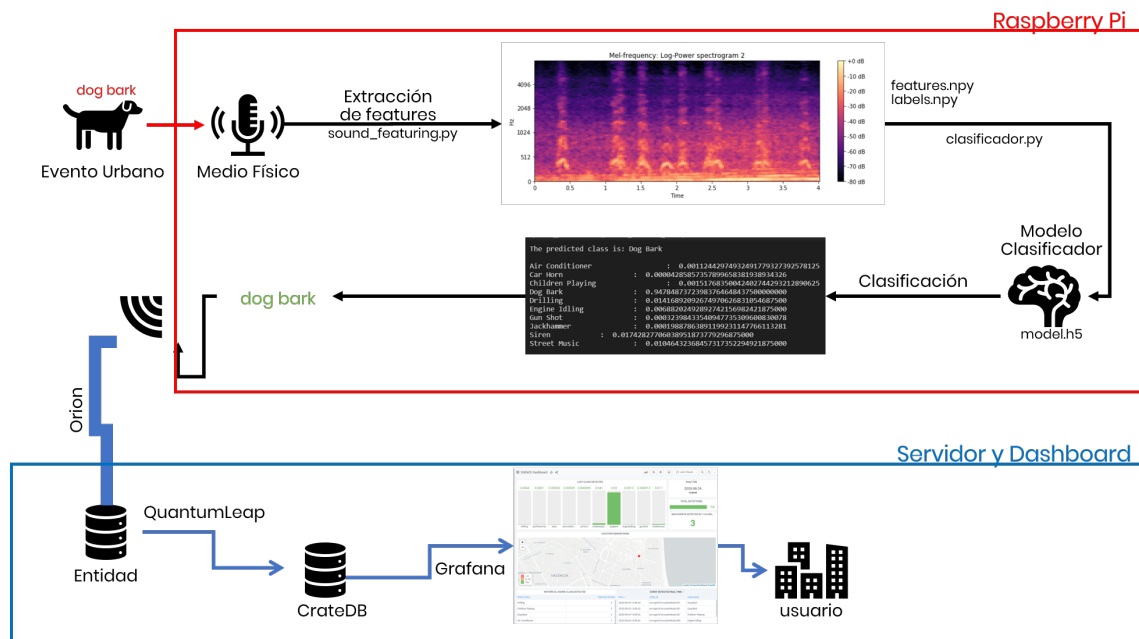


Figura 4.1: Esquema global del proyecto

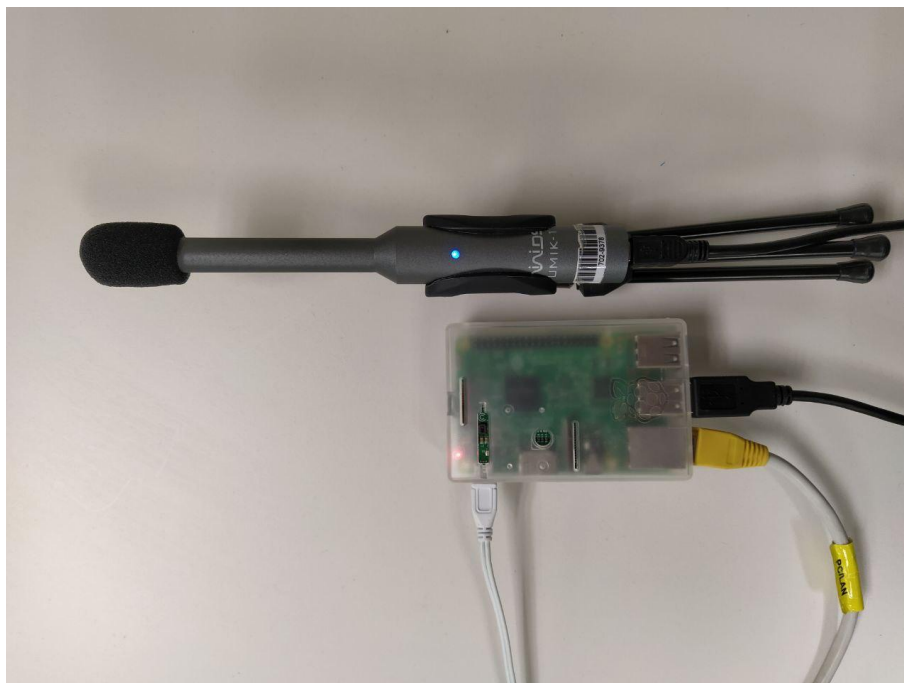


Figura 4.2: Imagen descriptiva del nodo sensor de este proyecto I



Figura 4.3: Imagen descriptiva del nodo sensor de este proyecto II

4.2. Extracción de Características del Sonido Urbano

En los últimos años, en el campo de la inteligencia artificial se han logrado avances significativos a través del modelado de sistemas humanos. Tal y como hemos mencionado anteriormente, las redes neuronales artificiales son modelos matemáticos que solo se combinan libremente replicando las estructuras y modelos de las neuronas humanas reales, pero su capacidad para resolver problemas complejos y ambiguos del mundo real ha ido en aumento a niveles que no se hubiesen alcanzado de otra manera. Además, el estudio de estos modelos neuronales ha llevado a que nuestra sociedad y la humanidad adquiera amplias nociones sobre nuestra capacidad cognitiva y poder interpretar datos de una manera más significativa.

Un ejemplo de esto lo encontramos en el reconocimiento y procesamiento del sonido. La búsqueda y análisis de nuestra capacidad cognitiva y nuestra forma de procesar los datos, nos ha llevado a crear modelos matemáticos que modifican y procesan los datos de manera análoga al ser humano. Lo que parece un análisis poco preciso o más abstracto es lo que ha permitido generar modelos de aprendizaje para las neuronas artificiales análogos a los del ser humano, replicando de esta forma nuestra capacidad cognitiva y perceptiva, obteniendo resultados muy próximos a los nuestros.

Mientras la capacidad perceptiva humana excede la de las máquinas, podemos ganar entendiendo los principios de los sistemas humanos. Los humanos somos muy hábiles cuando se trata de tareas de percepción y el contraste entre la comprensión humana y el status quo de la IA se hace particularmente evidente en el área de la audición mecánica.

Quizás el punto más abstracto del sonido es cómo nosotros, como humanos, lo percibimos. Si bien una solución para un problema de procesamiento de señales se debe analizar y operar con parámetros de intensidad, propiedades espectrales y temporales, el objetivo final es cognitivo: por lo que la caracterización de la señal debe englobar toda esa información de la misma forma que nosotros la percibimos. Es decir, los parámetros de bajo nivel ofrecen información de la señal obtenida, pero no es hasta que se interpretan y relacionan estos datos que se obtiene el verdadero “significado” detrás de ese evento acústico.

En el contexto de los sonidos urbanos, se ha popularizado el uso de los coeficientes Mel para su clasificación debido a su caracterización del sonido en tiempo y frecuencia. Los MFCCs es una representación en el espectro, de la señal acústica, que deriva de la FFT. A diferencia de las imágenes, la representación espectral del sonido no refleja en sus ejes el sonido en el espacio, lo cual es necesario para un sistema de interpretación del sonido. Por este motivo, es necesario transformar la señal del dominio temporal a su representación espectral en tiempo y frecuencia. De esta forma, se puede crear una imagen que representa las características fundamentales de cada evento acústico, simulando la interpretación del sonido más semejante a la del ser humano.

Para la obtención del MFCCs, en primer lugar se realiza la Transformada Discreta de Fourier (DFT) de la señal de entrada. El resultado obtenido por la DFT se introduce como entrada a un banco de filtros en escala Mel. Es aquí donde esta técnica, simula nuestra percepción acústica, ya que el oído humano tiene mayor capacidad para distinguir grupos de frecuencias cercanas en bajas frecuencias. La escala Mel relaciona así la frecuencia percibida de un tono con su medida real en frecuencia Eq. 4.1. Con esta técnica se ha obtenido una representación espacial de cada una de las 10 clases acústicas empleadas en este proyecto. Para obtener estos coeficientes se ha empleado la librería de python librosa, cuya función se conoce como melspectrogram, con la que se obtiene los coeficientes de mel para un fragmento de audio dado.

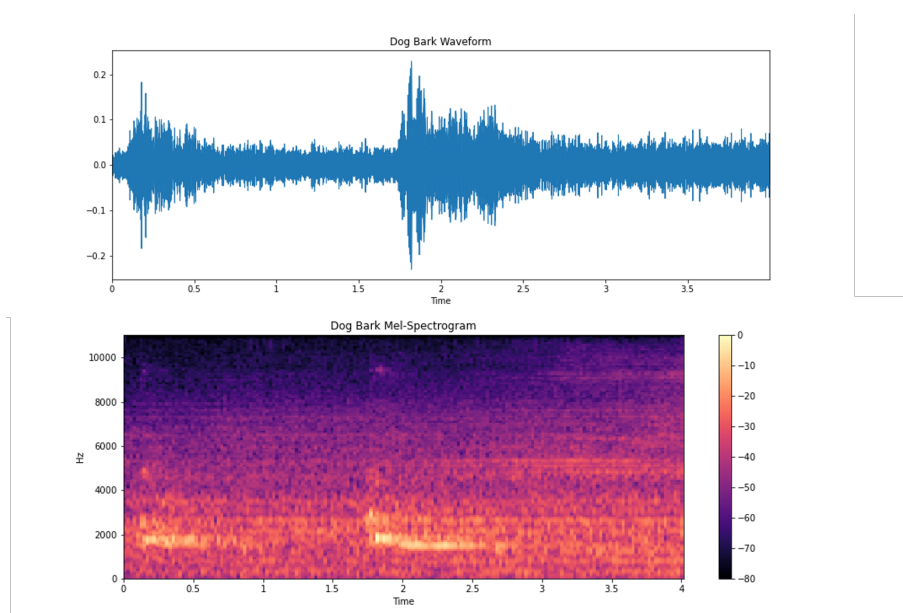


Figura 4.4: Caracterización acústica de perros ladrando

$$M(f) = 1125 \ln\left(1 + \frac{f}{700}\right) \quad (4.1)$$

Previamente a la obtención de características Mel, se ha tenido en cuenta que los eventos acústicos almacenados en la base de datos de SONYC, empleada en este proyecto, tienen una duración entre 1 y 4 segundos. Teniendo en cuenta que los datos que entren a la red neuronal deben seguir una estructura similar y coherente para poder detectar aquellas diferencias generadas en las características intrínsecas del evento detectado y no por su duración, previamente a la extracción de características se realiza un “framing” de 3 segundos. Es decir, se recorta aquellos fragmentos cuya duración es superior a 3 segundos y se realiza un “zero-padding” hasta completar los 3 segundos de duración para aquellos fragmentos de duración inferior. De esta forma, todos los eventos acústicos que se introduzcan a la red, se recortan o rellenan en 3 segundos. [ref función recortado a 3 segundos].

Una vez ajustado los datos de entrada para que cumplan con los requisitos de entrenamiento establecidos para la red, se pasa a la extracción de características de cada fragmento de audio. El procedimiento empleado para la extracción de características es el siguiente:

- Primero: se obtienen un total de 65.536 muestras a una frecuencia de muestreo de $22,050Hz$ del fragmento de audio de $3s$.
- Segundo: para obtener el espectrograma de mel de ese fragmento se emplea una ventana Hanning de $23ms$, es decir, 512 muestras (teniendo en cuenta el “sample rate” de $22,050Hz$) con un salto del mismo tamaño a la ventana, por lo que no se emplea solape (en ocasiones se recomienda el uso de un solape del 50 % compensando la reducción de energía en el espectro ocasionado por el enventanado).
- Tercero: se ha calculado el MFCCs para 128 bandas en el rango de $0-22,050Hz$. Obteniendo como resultado un vector de 128 coeficientes por banda (128, 128).

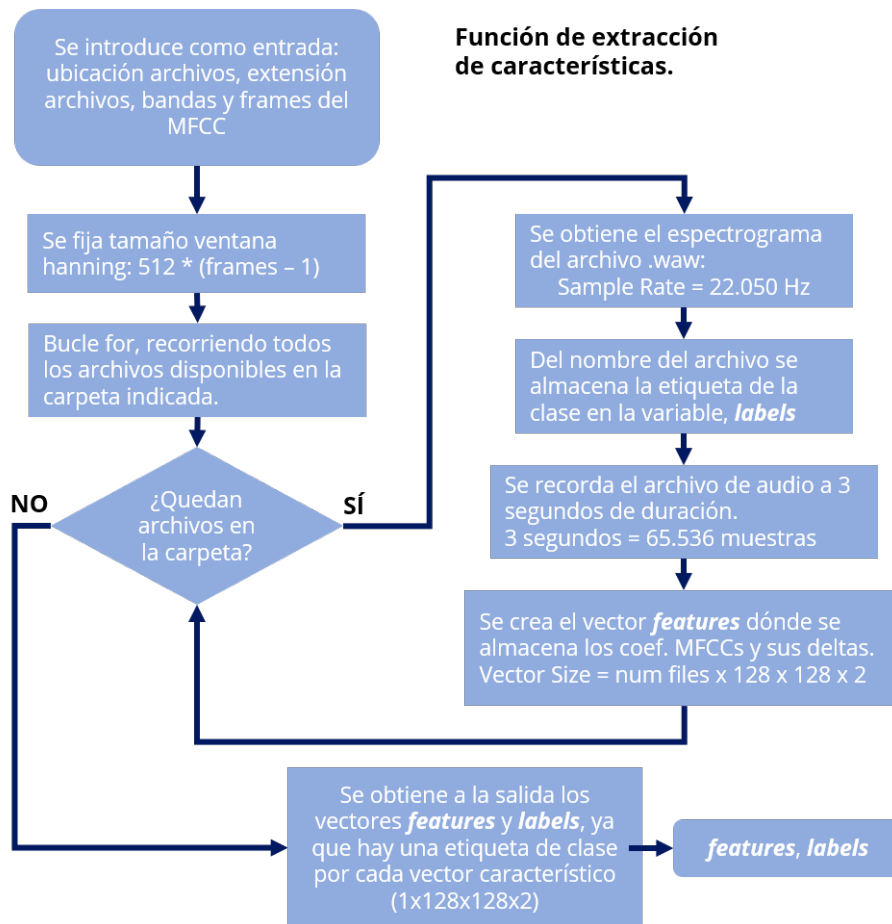


Figura 4.5: Esquema Función de Extracción de Características de un fragmento de Audio

- Cuarto: a continuación se obtiene los MFCCs en potencia (dB)
- Quinto: finalmente se calcula las deltas de los MFCCs, obteniendo un vector final de salida de (128, 128, 2), con dos dimensiones, que formará el vector de características del fragmento de audio analizado dentro de la red neuronal.

A modo de resumen, los MFCCs representan la envolvente espectral de la señal acústica analizada obteniendo importantes características que identifican cada una de las 10 clases acústicas utilizadas en este proyecto. Así como los coeficientes de Mel nos dan información sobre la energía de la señal entre las altas y bajas frecuencias, para obtener más información de la señal se calcula los MFCC-Delta-Delta. Estos coeficientes representan la evolución temporal de la señal acústica permitiendo observar la variabilidad entre diferentes eventos acústicos.

4.3. Clasificador CNN

Tras analizar cómo los coeficientes de Mel nos aportan información sobre la energía de la señal entre las altas y bajas frecuencias, para cada una de las 10 clases acústicas expuestas en este estudio. El objetivo siguiente, fue determinar el mejor modelo que aproxime la correcta clasificación de un evento acústico dado en una de las 10 clases empleadas, haciendo uso de su representación de los coeficientes de Mel.

Partiendo del uso de una representación gráfica como entrada a la red, que representa de forma precisa la energía en altas y bajas frecuencias de una señal acústica detectada, permitiendo un fácil reconocimiento de las clases. Se optó, en primera instancia, del uso de una Red Neuronal Convolutiva como mejor modelo a usar para clasificar los eventos acústicos detectados, haciendo uso de los MFCC-Delta-Delta como entrada a la red.

Experimentalmente las CNN cuentan con diversidad de arquitecturas que optimizan el resultado para casos distintos. En esta experimentación se optó por una arquitectura convencional de 5 capas, 3 capas convolucionales y 2 de densidad. A partir de esta arquitectura se fue testeando la efectividad de diferentes tamaños de kernel para nuestra entrada de datos, y el resultado que obtenemos con esto, hasta que finalmente optamos por la configuración que más se aproximó al 71 % en precisión de clasificación.

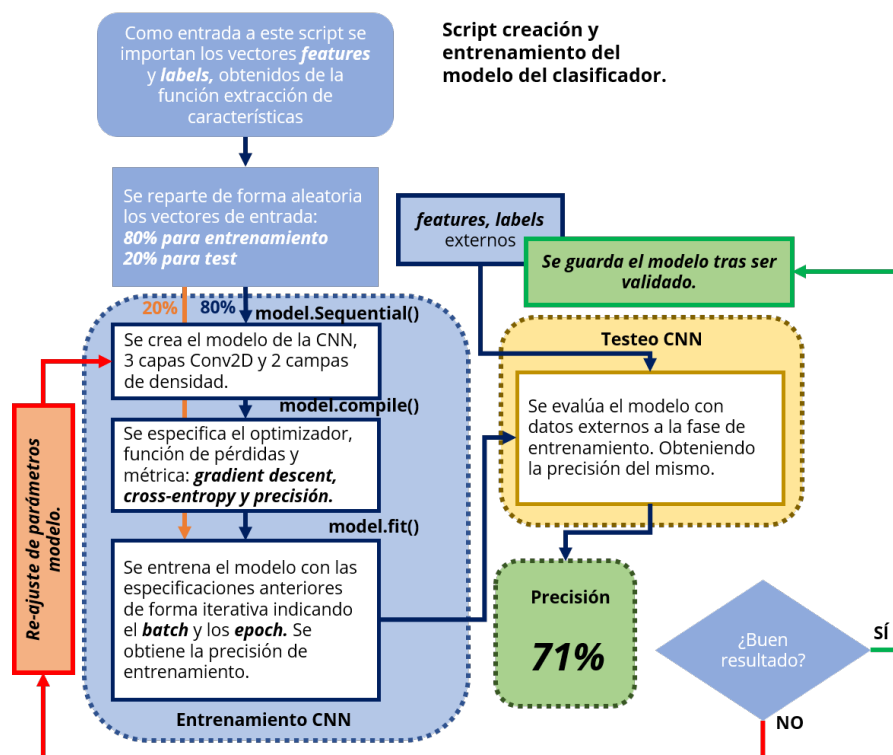


Figura 4.6: Esquema Implementación Clasificador de sonidos urbanos con CNN

Tal y como se puede observar en el esquema Fig. 4.6, se ha realizado 3 tareas principales para la confección del clasificador implementado por una CNN.

4.3.1. Extracción de Características

En primer lugar se debe preparar el conjunto de características que representan los datos analizados para entrenar la red neuronal. En nuestro caso, tras la extracción y cálculo de los coeficientes MFCC-Delta-Delta, se obtienen dos vectores uno en el que se encuentran todas las características extraídas para cada muestra de audio de la base de datos de entrenamiento así como un vector de una sola columna con las etiquetas que identifican la clase a la que pertenece cada muestra en el vector de características.

La preparación de estos datos consiste principalmente en crear dos conjuntos de pares de datos features-labels, en las que se divide el total de muestra analizadas en un 80 % de la misma para entrenar la red y un 20 % para la tarea de testeo de la red. Esta división consiste en ordenar de forma aleatoria las muestras combatiendo posibles situaciones de overfitting, además de dividir posteriormente el total entre el 80 % y 20 %. De esta forma se preparan los datos para asegurar un correcto entrenamiento y testeo de la red.

4.3.2. Estructura de la CNN

Tras la preparación de los datos de entrada a la red, se dispone a la configuración de la arquitectura de red escogida para este clasificador. Tal y como se ha mencionado desde el inicio se optó por la configuración actual de la red que sigue una arquitectura convencional de CNN de 5 capas. En la optimización de la misma se fueron modificando las funciones de activación y los kernels y filtros de cada capa que optimizaran el resultados para ese tipo de datos de entrada. A continuación se puede apreciar la arquitectura empleada en el clasificador Fig. 4.7.

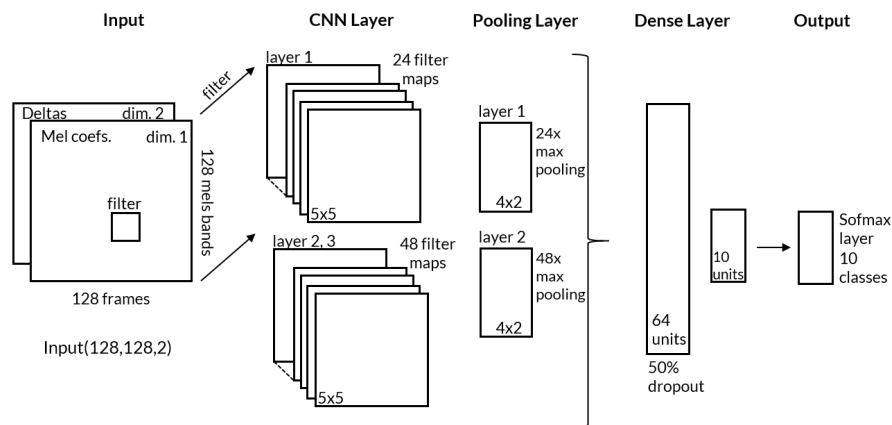


Figura 4.7: Esquema Arquitectura CNN

Se ha modificado el modelo para que sea una Red neuronal convolucional (CNN) nuevamente usando Keras y un backend de Tensorflow.

Se ha utilizado un modelo secuencial, comenzando con una arquitectura de modelo simple, que consta de tres capas de convolución Conv2D, siendo nuestra capa de salida final dos capas densas.

Las capas de convolución están diseñadas para la detección de características. Funciona deslizando una ventana de filtro “kernel” sobre la entrada y realizando una multiplicación de matriz y almacenando el resultado en un mapa de características. Esta operación se conoce como convolu-

ción.

El parámetro de filtro especifica el número de nodos en cada capa. Cada capa aumentará de tamaño de 24 a 48, mientras que el parámetro *kernel-size* especifica el tamaño de la ventana del kernel, que en este caso es 5, lo que da como resultado una matriz de filtro 5×5 .

La **primera capa** recibirá la forma de entrada de $(128, 128, 2)$ donde 128 es el número de MFCC o bandas Mel (representa la señal en frecuencia) y el otro 128 es el número de frames de la señal de entrada teniendo en cuenta el relleno, es decir 128 segmentos de $23ms$ de un total de $3s$ de duración de la señal entrante (representa la señal en tiempo). Y el 2 significa que es un vector de 2 canales, donde se agrega el cálculo delta de MFCC's para ambas columnas Fig. 4.8.

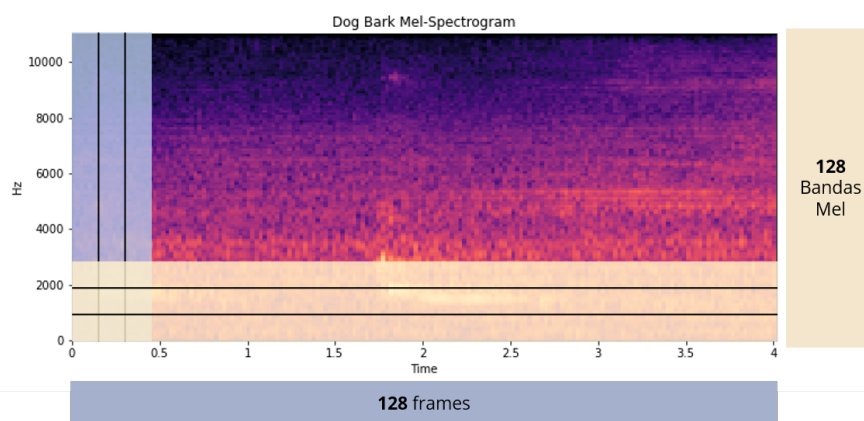


Figura 4.8: Extracción Vector Características (features)

La **función de activación** que utilizaremos para nuestras capas convolucionales es una operación no lineal Rectified Linear Unit (ReLU), por medio de la cual aquellas zonas que tras la convolución se obtiene un resultado negativos se sustituyen por cero, obteniendo un resultado no lineal a partir de una operación lineal. Utilizaremos un valor de dropout menor del 20 % en nuestras capas convolucionales.

Cada capa convolucional tiene una capa de agrupación asociada de tipo MaxPooling2D. La capa de agrupación *pooling* reduce la dimensionalidad del modelo (al reducir los parámetros y los requisitos de cálculo subsiguientes) que sirve para acortar el tiempo de entrenamiento y reducir el sobreajuste o *overfitting*. El tipo de agrupación máxima toma el tamaño máximo para cada ventana, reduciendo la dimensión de los mapas de características manteniendo aquellos más importantes. Por otra parte, la operación de *pooling* presenta invariabilidad ante pequeñas transformaciones en la señal de entrada, ya que al agrupar los valores de cada zona de la señal, un cambio en la señal no produciría cambio alguno.

Para evitar el sobreajuste del modelo, se aplica el método de *Dropout* a cada neurona con una probabilidad de 0,5. De esta forma se anula la activación de la neurona de forma aleatoria dada una probabilidad de 0,5 en este caso.

Nuestra capa de salida tendrá 10 nodos (num-labels) que coinciden con el número de clasificaciones posibles. La activación para la capa de salida es softmax. Softmax hace que la salida sume 1, por lo que la salida puede interpretarse como probabilidades. El modelo hará su predicción según la opción que tenga la mayor probabilidad.

4.3.3. Entrenamiento del Clasificador

Finalmente para el entrenamiento de la red se emplea el método de optimización **adam** que se trata de un algoritmo de descenso de gradiente estocástico, que se emplea de forma iterativa para optimizar la función objetivo. Como hiperparámetro de la red está la tasa de aprendizaje o *learning rate*, mediante la cual el **gradient descent** determina el siguiente punto. Si la tasa de aprendizaje es demasiado pequeña el gradiente converge muy lentamente y si es demasiado grande, este puede no converger. En este proyecto se ha empleado una tasa de entrenamiento de 0,001

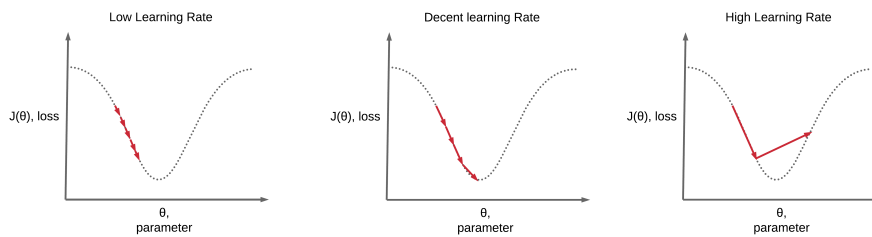


Figura 4.9: Esquema Learning Rate

Debido a que se trata de una red Feed Forward, tal y como se ha explicado anteriormente, la información pasa a la red y la clasificación resultante se compara con la muestra de entrenamiento conocida. Si la clasificación de la red es incorrecta, los pesos se ajustan hacia atrás a través de la red en la dirección que le daría la clasificación correcta, por **BackPropagation**.

Teniendo en cuenta que el gradient descent se trata de un proceso iterativo en el que se calcula los pesos de la red en cada iteración, es necesario procesar el dataset más de una vez. Cada vez que se procesa el dataset completo se conoce como época (epoch). Para realizar el entrenamiento completo se realizan varios epochs al final de la cual se realiza una etapa de validación en la que se emplean datos que no había visto la red anteriormente y de la cual no se obtendrá un aprendizaje. Se calcula así la función de pérdidas y la precisión de ensayo. Esta métrica permite llevar un control del entrenamiento, de forma que si no se obtiene una mejora en el transcurso de varios epoch consecutivos el entrenamiento llega a su fin.

Para el entrenamiento de la red empleada, se ha empleado un total de 25 epochs, los cuales han sido suficientes para obtener el mismo resultado de precisión que con mayor número de épocas.

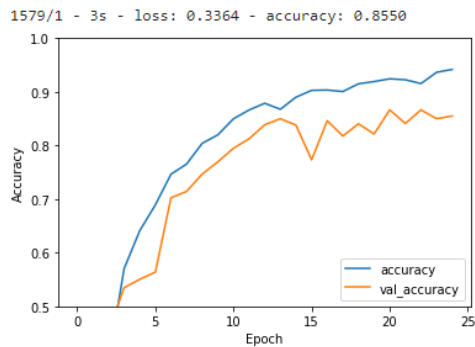
Tal y como se puede apreciar en la gráfica Fig. 4.10, como resultado del entrenamiento se ha obtenido una precisión del 96 % en entrenamiento y 85 % para el test. Una vez entrenado se ha realizado una validación externa del modelo, con objeto de obtener el % de precisión del modelo con una base de datos ajena a la empleada en el entrenamiento. Tras la realización de esta validación se ha obtenido un 71 % de precisión del modelo, que se ajusta a los resultados y publicaciones realizadas en este estudio [3]. Otros modelos e implementaciones han alcanzado 85 % hasta 92 % empleando una mejora por hiperparámetros de la red, y haciendo uso de una arquitectura CNN-VGG cuyo rendimiento mejora ampliamente. [4, 21]

En las figuras Fig. 4.11, Fig. 4.12, Fig. 4.13 y Fig. 4.14 se ha realizado un análisis de validación del modelo con dos segmentos del dataset que son ajenos a la fase de entrenamiento del mismo. Las dos primeras corresponden al resultado de clasificar el dataset que forma parte del 20 % extraído del mismo dataset del entrenamiento, mientras que los otros resultados corresponde a la clasificación de un dataset que es totalmente ajeno a la fase de entrenamiento, como prevención a posibles casos

Evaluate the model

```
[22]: plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1])
plt.legend(loc='lower right')

# test_loss, test_acc = model.evaluate(X_train, Y_train, verbose=2)
test_loss, test_acc = model.evaluate(X_test, Y_test, verbose=2)
```



```
[23]: train_score = model.evaluate(X_train, Y_train, verbose=0)
test_score = model.evaluate(X_test, Y_test, verbose=0)

print("Training Accuracy: ", train_score[1])
print("Testing Accuracy: ", test_score[1])
```

Training Accuracy: 0.96975935
Testing Accuracy: 0.8549715

Figura 4.10: Resultado Fase Entrenamiento Clasificador

```
[[184  0  1  0  0  5  0  2  0  1]
 [  0 70  1  0  2  0  2  1  0  3]
 [  5  0 143  5  4  3  1  0  1 14]
 [  6  2  14 127 10  2  5  0  5 11]
 [  3  0  2  1 140  5  1  6  1  4]
 [  5  3  5  1  2 183  1  0  0  3]
 [  2  0  0  0  0  0  51  1  0  0]
 [  0  0  2  1  5  0  2 174  0  1]
 [  3  0  2  4  0  2  0  4 143  4]
 [  5  1 23  3  9  3  0  1  2 135]]
```

Figura 4.11: Matriz de Confusión test-dataset 1

de overfitting. En este sistema de validación se ha tenido en cuenta la matriz de confusión y el porcentaje de clasificación por clase que se ha obtenido estudiando el etiquetado original respecto el predicho por el clasificador y obteniendo la precisión para cada caso.

De esta forma, se puede apreciar que las clases *children playing* y *street music* son las clases que más se han confundido en la clasificación en ambos dataset-test. También han sido el caso de *siren*, *engine idling* y *air conditioner* que se han clasificado de forma errónea con otras clases.

Analizando los datos, se puede identificar que hay ciertas clases que son más propensas a confusión dentro de la clasificación. Esta información es de vital importancia a la hora de analizar y caracterizar acústicamente los eventos acústicos y poder mejorar el reconocimiento de sonidos de forma autónoma. Tal y como se puede ver en la matriz de confusión la clase *air conditioner* se confunde destacablemente con la clase *engine idling*, *siren* y *children playing* respectivamente de

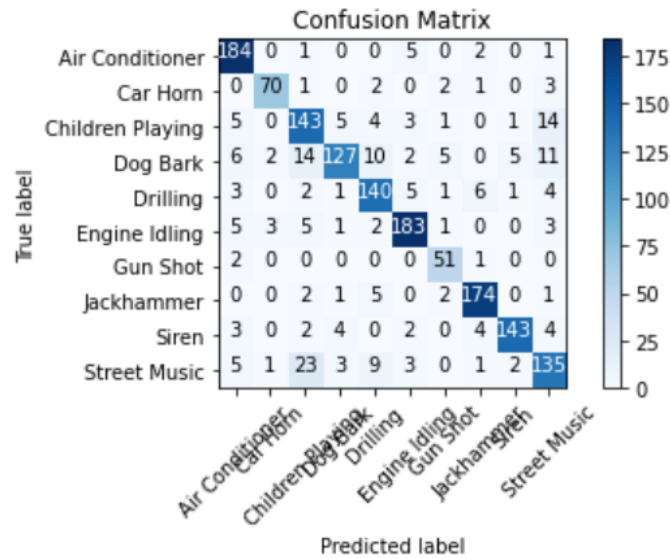


Figura 4.12: Matriz de Confusión test-dataset 1

```
[[50  0 10  0  3 15  0  7 13  2]
 [ 0 28  1  0  1  1  0  1  0  1]
 [ 3  1 66  1  3  5  1  1  6 13]
 [ 1  2 13 62  9  1  4  2  1  5]
 [ 0  0  4  5 72  4  3  1  3  8]
 [10  0  2  0  6 65  0  9  0  1]
 [ 0  0  0  0  0  0 32  0  0  0]
 [ 0  0  0  0  7  1  0 88  0  0]
 [ 1  1 12  3  8  0  0  1 53  4]
 [ 5  1 15  0  2  0  0  0  2 75]]
```

Figura 4.13: Matriz de Confusión test-dataset 2

mayor a menor medida. Finalmente, tal y como se ha comentado anteriormente, la clase *children playing* se confunde destacablemente con las clases *air conditioner*, *dog bark*, *siren* y *street music*.

Extrayendo la información anterior y obteniendo los resultados numéricos de la precisión por clase, se ha representado los resultados de ambos dataset en tablas y gráficas comparando el número de eventos disponibles y aquellos que se han clasificado en cada clase además de su precisión de acierto en cada caso Tabla. 4.1, Tabla. 4.2. Las tablas se han compuesto de la siguiente información, la primera columna corresponde al numero total de muestras disponibles por clase en cada dataset, es decir, para la primera clase hay un total de X muestras etiquetadas con esa clase. En la segunda columna se ha numerado el total de muestras que la red neuronal ha clasificado como perteneciente a esa clase, por lo que para la primera clase indicará que se han clasificado como tal X muestras, las cuales podran ser tantas como haya acertado o errado la red neuronal. Finalmente, la última columna se muestra el porcentaje de precisión de la red neuronal por cada clase, teniendo en cuenta el total de muestras acertadas menos aquellas que sean erróneas para determinar la precisión que presenta el clasificador por cada clase. Otro aspecto a tener en cuenta, es el dataset empleado, el número de eventos con los que se ha realizado el estudio corresponde al siguiente:

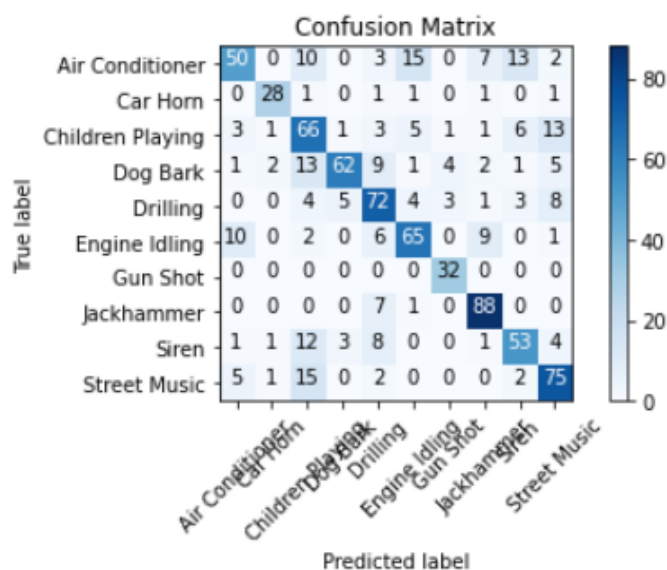


Figura 4.14: Matriz de Confusión test-dataset 2

- Primer dataset (test-dataset 1): se trata del 20 % del total del dataset empleado para entrenar la red, en el que el 80 % restante del mismo dataset se ha usado en la fase de entrenamiento. Este dataset representa un total de 1579 eventos acústicos a clasificar.
- Segundo dataset (test-dataset 2): este conjunto de datos pertenece a un subconjunto del dataset que no ha formado parte de la fase de entrenamiento siendo ajeno al mismo, por lo que no sigue un patrón de reconocimiento identificable por el modelo. En este dataset se ha contado con un total de 837 eventos a clasificar.

De los datos siguientes, se puede observar que las clases *car horn*, *engine idling*, *siren* y *street music*, obtienen un porcentaje de clasificación por encima del 90 % en ambos casos. En gran medida, esta precisión se debe a la caracterización de esos eventos acústicos y su representación Mel, que los hacen más fácilmente reconocibles que otros. Por otra parte, las clases como *air conditioner*, *children playing*, *dog bark*, *drilling*, *gun shot*, *jackhammer* en muchas ocasiones suelen confundirse entre las otras clases empleadas.

Finalmente, el resultado que se obtiene de este análisis y de mayor interés en nuestro sistema está compuesto por el porcentaje de clasificación por cada clase de cada muestra de audio clasificada y la clase predecida por el clasificador para esa misma muestra. De esta forma, se enviará a la red esta información ofreciendo al sistema conocimiento de qué clase de eventos acústicos se están manifestando por todo el sistema. Un ejemplo de este resultado, se puede apreciar en la Fig. 4.15

Ya son muchas las publicaciones relacionadas con la clasificación de eventos urbanos. [ref publicaciones]. Desde 2014 se registra la confección de 3 bases de datos de 50 y 10 clases a clasificar de eventos acústicos que han sido un referente en este área de estudio. En esta tabla se puede observar las diferentes arquitecturas que se han testeado en este área de estudio. En el proyecto que se presenta en esta tesis se ha optado por realizar la configuración Logmel + CNN, a modo de experimentación, con una de las primeras configuraciones que se propusieron dentro de este área de investigación.

Tabla 4.1: Resultado clasificación test-dataset 1

Resultados	Muestras Originales	Muestras Clasificadas	Precisión
Air Conditioner	193	213	89.64 %
Car Horn	79	76	96.20 %
Children Playing	176	193	90.34 %
Dog Bark	182	142	78.02 %
Drilling	163	172	94.4 %
Engine Idling	203	203	100 %
Gun Shot	54	63	83.33 %
Jackhammer	185	189	97.84 %
Siren	162	152	93.83 %
Street Music	182	176	96.70 %

Tabla 4.2: Resultado clasificación test-dataset 2

Resultados	Muestras Originales	Muestras Clasificadas	Precisión
Air Conditioner	100	70	70 %
Car Horn	33	33	100 %
Children Playing	100	123	77 %
Dog Bark	100	71	71 %
Drilling	100	111	89 %
Engine Idling	93	92	98.92 %
Gun Shot	32	40	75 %
Jackhammer	96	110	85.42 %
Siren	83	78	93.98 %
Street Music	100	109	91 %

```

Original class: Siren
The predicted class is: Siren

Air Conditioner      : 0.00000000464027039015491027384996
Car Horn             : 0.00000001728573906234487367328256
Children Playing    : 0.00000004405102771443125675432384
Dog Bark            : 0.00005393630272010341286659240723
Drilling            : 0.00000001956304096495387057075277
Engine Idling       : 0.00000004350610538494947832077742
Gun Shot            : 0.00000000000000000000000037347441946
Jackhammer          : 0.0000000000000000000012945114215104
Siren               : 0.99984598159790039062500000000000
Street Music        : 0.00009986794611904770135879516602

```

Figura 4.15: Datos de salida del Clasificador propuesto.

El objetivo era realizar una primera aproximación a un modelo funcional, capaz de clasificar eventos acústicos en escenarios urbanos. El resultado ha sido exitoso tal y como se puede apreciar en la tabla, nuestra precisión del 71 % se acerca considerablemente a la referencia obtenida

Tabla 4.3: Comparación de precisión de otros modelos publicados sobre el dataset Urbansound8k

Model	Feature	Urbansound8k
SB-CNN	Log-mel	79 %
Logmel + CNN	Log-mel	73.7 %
Logmel + CNN + BN	Log-mel	74.7 %
MelNet	Log-mel	90.2 %
M18	Raw waveform	71.68 %
RawNet	Raw waveform	87.7 %
DS-CNN	Combine	92.2 %
Ave-CNN	Combine	91.6 %
Pro-CNN	Combine	91.9 %

por el proyecto SONY cuya misma configuración obtuvo un 73 % de precisión, en ambos casos empleando el dataset *Urbansound8k*.

El estudio que se ha realizado en esta tesis es una primera aproximación a un modelo de clasificación funcional de eventos acústicos urbanos. Con objeto de crear todo un prototipo funcional de una red de sensores capaces de clasificar y detectar eventos acústicos en tiempo real. Partiendo de esta base, el modelo propuesto no es el más óptimo descubierto hasta la fecha. Existen ya muchas publicaciones que proponen una gran variedad de modelos que aproximan un mejor resultado de clasificación, tal y como se puede observar en la tabla anterior Tabla. 4.3.

A modo de introducción a la mejora del modelo, cabe saber que en esos estudios se ha detectado que el uso de los MFCCs junto con la información de la señal acústica detectada ofrecen la posibilidad de reconocer diferentes patrones dado un sonido. De esta manera, los modelos de reconocimiento basados en esta dos características combinadas pueden detectar patrones de reconocimiento más precisos que los modelos en los que sólo se emplean las características por separado MelNet o RawNet. Si a esta mejora, le añades un algoritmo de validación, ofreces la posibilidad de crear un modelo más robusto que los modelos anteriormente presentados. En esta línea, sigue el desarrollo de clasificadores de sonidos urbanos, en los que se alcanza hasta un 92,2 % de precisión de clasificación implementando esta técnicas, como es el caso del modelo DS-CNN Fig. 4.16.

Además del número de iteraciones y el optimizador empleado para el entrenamiento, otro aspecto a considerar en la fase de entrenamiento y validación es el coste temporal que ha significado entrenar toda la red. Empleando un ordenador con una CPU Intel Core i7 6700HQ @ 2.6GHz con 16GB de Ram y sistema operativo Windows 10, se obtenía un tiempo medio de 30 minutos para la realización del entrenamiento completo de la red, empleando el 90 % del dataset de Urban-Sound8K.

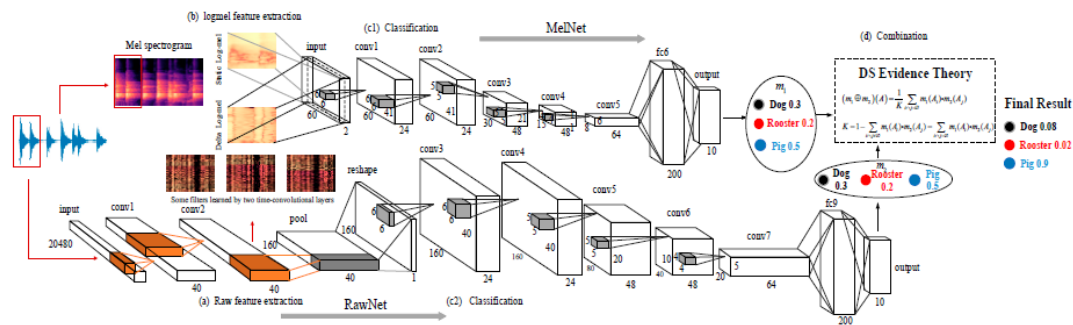


Figura 4.16: Arquitectura del modelo DS-CNN

4.4. Red IoT FIWARE

Una vez la red clasifica un evento acústico que ha sido detectado, esta clasificación se envía al servidor y a otros nodos subyacentes a través de la red FIWARE. De esta forma los nodos clasificadores que se despliegan con las Raspberry Pi se comunican en tiempo real con el servidor y así poder obtener datos de toda la red y validarlos en todo momento.

FIWARE es un framework que ofrece tanto a los desarrolladores como a los usuarios una plataforma de código abierto que se puede utilizar de forma independiente y que se puede integrar fácilmente con otros marcos. FIWARE fue desarrollado para crear una solución inteligente y abierta para redes de sensores. [18]

Hemos implementado los componentes requeridos para crear un canal de suscripción entre usuarios y sensores, así la información que genera cada sensor acústico de la red se comparte en tiempo real. El flujo de datos dentro de una arquitectura FIWARE pasa por tres componentes esenciales: productor de contexto, intermediario de contexto y consumidor de contexto, en el caso de la red propuesta sólo se ha empleado el broker o intermediario de contexto de la red FIWARE. El resto de elementos se han implementado con otras tecnologías o frameworks de acuerdo a los requerimientos de la aplicación final. En la Fig. 4.17 muestra la arquitectura de red IoT propuesta que incluye los siguientes bloques: el nodo del sensor acústico, los componentes de FIWARE Orion context broker y QuantumLeap, el acceso a la base de datos MongoDB y CrateDB, y finalmente la herramienta de visualización Grafana.

4.4.1. Arquitectura

Cada componente cumple un papel determinado en el proceso de visualización y almacenamiento de los datos generados por los sensores de clasificación. Este papel viene determinado por sus características de funcionalidad. Para implementarlo se ha empleado Docker de forma que se ha instanciado cada una de las imágenes correspondientes a Orion, QuantumLeap, CrateDB, MongoDB y Grafana. En esta sección se describe cada uno de los componentes así como la arquitectura de la red y el empleo de Docker para desplegarla.

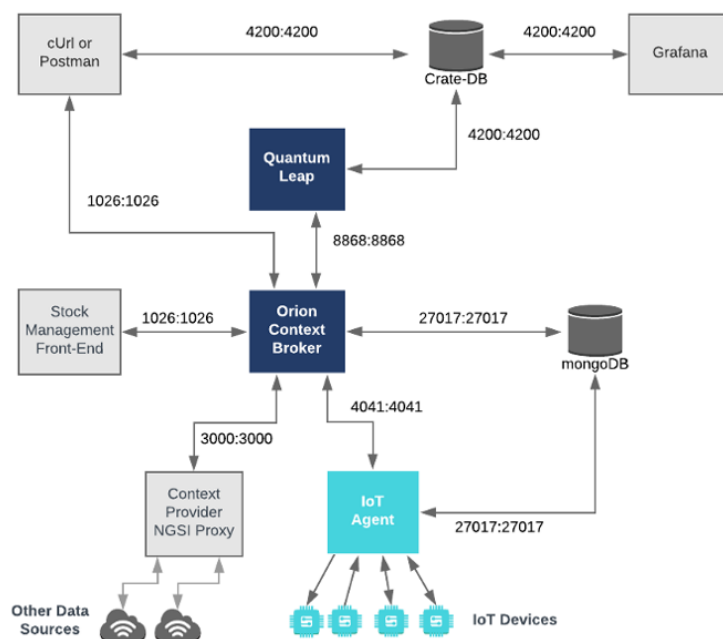


Figura 4.17: Arquitectura de Red IoT del proyecto

4.4.1.1. Modelo Suscripción

En este proyecto se ha creado una red IoT y servicio de visualización Web con objeto de poder obtener en tiempo real y de forma remota los datos generados por los nodos clasificadores, además de poder gestionar los datos y modificarlos para su correcta visualización. Para hacer esto posible se ha seguido una arquitectura de red por suscripción, en la que los nodos y usuarios se conectan al broker sobre el que se vierten los datos generados por los sensores y los clientes pueden acceder en todo momento a ellos.

Este modelo es el más óptimo para redes IoT en el que los nodos vierten información de actualización de forma asíncrona sin necesidad de que un usuario realice una petición como es el caso del protocolo HTTP, permitiendo así a los usuarios poder monitorizar eventos en tiempo real y sin necesidad de supervisión de la red.

Tal y como se puede observar en la imagen este protocolo consiste en la publicación de un tópico o canal, en el que el generador de contenido va a verter un conjunto de datos determinado, aquellos usuarios que estén interesado en el contenido de ese tópico o canal, simplemente se suscriben al mismo y obtendrán toda la información que se vierta en el mismo. Este protocolo de comunicación ofrece la ventaja de poder obtener información de valor en tiempo real y de forma asíncrona al observador, además de genera mucho menos tráfico debido a que los observadores no deben de realizar peticiones cada vez que quieran adquirir el nuevo valor del *topic* como ocurre en el protocolo HTTP.

La red empleada Fig. 4.19 depende de tres agentes principales. Por un lado está el generador de contexto que será el que vierta información de monitorización sobre la red distinguiendo su contenido entre diferentes “topics” (sensor), el intermediario de contexto es el canal en el cual se vierte la información para poder distribuirla por toda la red (broker) y el consumo de contexto es

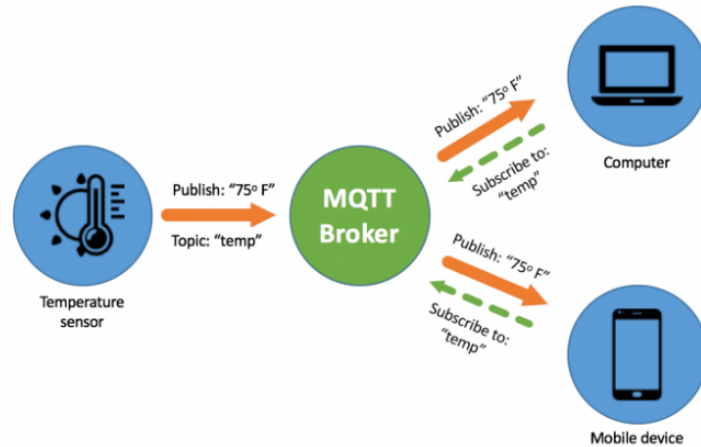


Figura 4.18: Ejemplo Protocolo de Publicación/Suscripción (MQTT)

aquello que se suscribe a un “topic” para poder visualizar y gestionar los datos de monitorización asociados a ese tópico (aplicación o cliente).

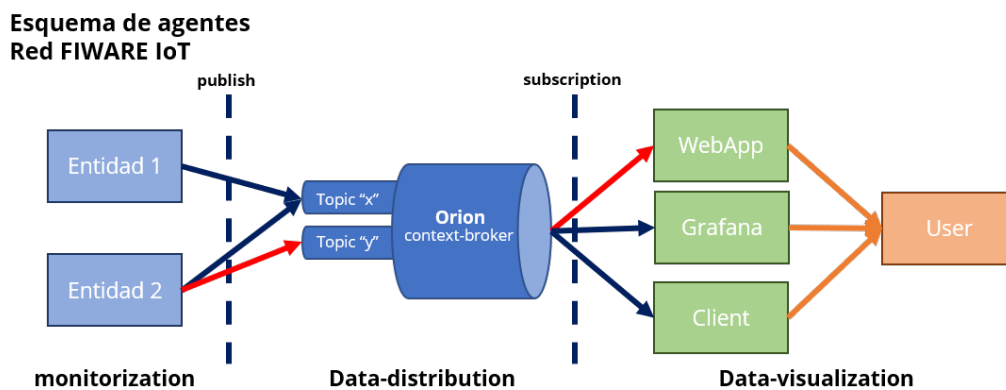


Figura 4.19: Esquema agentes red IoT FIWARE

Para llevar a cabo la arquitectura propuesta, se han empleado un conjunto de componentes que se explicarán a continuación. Los elementos que se han empleado de la arquitectura FIWARE son el context-broker y quantumLeap, los cuales instancian servicios terceros para un correcto funcionamiento.

4.4.1.2. Orion Context-Broker

Orion Context Broker nos permite administrar todo el ciclo de vida de la información del contexto, incluida las actualizaciones, consultas, registros y suscripciones. Es una implementación de servidor NGSIv2 para administrar información de contexto y su disponibilidad. Se pueden crear elementos de contexto y administrarlos mediante actualizaciones y consultas, además de poder suscribirse a la información de contexto para que cuando se produzca alguna condición reciba una notificación.

En este se gestionan los topic sobre los cuales las entidades vierten sus datos de monitorización.

A través de estos topics varias entidades pueden verter sus datos sobre un mismo topic pudiendo agrupar diferentes generadores de contexto sobre una misma fuente de datos de cara a los clientes.

Orion es la herramienta encargada de gestionar las entidades y sus suscripciones, pero no se encarga de mantener los datos persistentes en el tiempo. Esto significa que en Orion sólo se puede acceder a los estados actuales de cada entidad además de gestionar las suscripciones asociadas a cada entidad.

Para poder mantener de forma persistente la información de la entidades, esta emplea la herramienta de MongoDB para almacenar la información de cada entidad y poder modificarla en cualquier instante.

4.4.1.3. Entidad

Se conoce como entidad aquel objeto de la vida real que se va a monitorizar a ojos de la red FIWARE. Una entidad puede ser desde una nevera a una tienda completa, o una persona física. Cualquier cosa que pueda ser monitorizada en el mundo real, y de la cual se va a llevar un seguimiento en la red.

En el caso de nuestro proyecto, se ha creado como entidad cada Raspberry Pi que va a actuar de sensor acústico en la ciudad. De esta forma se almacenará la información de cada sensor en Orion y a través de QuantumLeap se crearán las suscripciones pertinentes para poder almacenar información persistente en el tiempo de cambios obtenidos en la entidad.

La creación de la entidad ha de seguir un modelo de datos que se ajuste a las características de la entidad que va a ser monitorizada en el mundo real. En este proyecto, la entidad sigue el modelo de datos **Device**. En el modelo de datos se definen los atributos esenciales que debe llevar un componente con las características de **Device**. Por definición este modelo corresponde a una entidad que monitoriza un evento específico en forma de sensor. Es por ese motivo que los atributos principales del modelo son los de localización y descripción del dispositivo. Además, de aquellos atributos en los cuales se almacenará la información de monitorización como puede ser la clase acústica detectada o el porcentaje de clasificación por cada clase como ocurre en nuestro caso.

Tras determinar qué datos necesitamos monitorizar en nuestro clasificador de eventos urbanos, el modelo de datos implementado en las entidades utilizadas en este proyecto para lo que sería una red de sensores de clasificación de eventos acústicos urbanos viene representado en las Fig. 4.20. En este caso se ha considerado que tanto el nombre de la clase con la que se ha clasificado el evento acústico detectado como el porcentaje de clasificación de cada clase son de especial interés para nuestra red de sensores clasificadores de eventos acústicos. De esta forma los únicos atributos que se han añadido de forma extraordinaria a las entidades de esta red, son un atributo por cada clase en el que se obtendrá el porcentaje de clasificación por clase, así como un único atributo en el que se indica el nombre de la clase clasificada para ese evento acústico detectado.

Para monitorear los resultados, se ha creado un total de 14 atributo nuevos Fig. 4.21:

- **Geohash:** este representa la localización exacta en la que se encuentra el sensor/entidad. Este atributo será modificado cada vez que se modifique la localización del sensor sin necesidad de modificar la dirección exacta del sensor que se había establecido inicialmente. Además este atributo permite la visualización de la localización en servicios de visualización.

```

{
  "id": "urn:ngsi-ld:AcousticNode:001",
  "type": "Device",
  "address": {
    "type": "PostalAddress",
    "value": {
      "streetAddress": "Camí de Vera, s/n Edificio 8G",
      "addressRegion": "Valencia",
      "addressLocality": "Valencia",
      "postalCode": "46022"
    }
  },
  "metadata": {
    "verified": {
      "type": "Boolean",
      "value": "true"
    }
  }
},
"category": {
  "type": "Text",
  "value": "sensor",
  "metadata": {}
},
"controlledProperty": {
  "type": "Text",
  "value": "noiseClass",
  "metadata": {}
},
"dataProvider": {
  "type": "URL",
  "value": "https://gtac.webs.upv.es/",
  "metadata": {}
},
"function": {
  "type": "Text",
  "value": "sensing",
  "metadata": {}
},
"location": {
  "type": "geo:json",
  "value": {
    "type": "Point",
    "coordinates": [
      39.477313,
      -0.335811
    ]
  }
},
"metadata": {},
"name": {
  "type": "Text",
  "value": "Raspberry David 000",
  "metadata": {}
},
"source": {
  "type": "URL",
  "value": "https://gtac.webs.upv.es/",
  "metadata": {}
}
}
]

```

Figura 4.20: Estructura modelo de datos Device, empleado en el proyecto

- 2 **atributos** añaden de forma manual la fecha de creación y de modificación de la entidad. Estos se han creado para usarlos como *trigger* en las suscripciones, de forma que cada vez que se modifique la fecha de modificación se realice una actualización de la entidad. Este atributo se modifica cada vez que se realiza una clasificación enviando así la fecha y hora actual de la clasificación.
- 10 **atributos** uno por cada clase de clasificación de los eventos acústicos. En estos atributos se almacenará por cada clase el porcentaje de clasificación que se ha obtenido por cada clase para un evento acústico dado. De esta forma se podrá obtener información de contexto que pueda mejorar la calidad de información de los eventos acústicos clasificados.
- Finalmente **noiseClass** es el atributo en el que se envía el nombre de la clase que ha clasificado dado un evento acústico.

Todos estos son los atributos que forman parte de cada sensor/entidad que se han creado para nuestra red. De esta forma se ha creado la estructura de datos compatible con los requisitos de red de sensores que queríamos diseñar.

```

{
  "id": "urn:ngsi-ld:AcousticNode:001",
  "type": "Device",
  "Geohash": {
    "type": "Text",
    "value": "ezpb86ekp",
    "metadata": {}
  },
  "creDate": {
    "type": "Text",
    "value": "2020-08-17 13:43:42.840175",
    "metadata": {}
  },
  "modDate": {
    "type": "Text",
    "value": "2020-08-17 13:43:42.840175",
    "metadata": {}
  },
  "noiseClass": {
    "type": "Text",
    "value": "Unknown",
    "metadata": {}
  },
  "Drilling": {
    "type": "Number",
    "value": "Unknown",
    "metadata": {}
  },
  "dogBark": {
    "type": "Number",
    "value": "Unknown",
    "metadata": {}
  },
  "engineIdling": {
    "type": "Number",
    "value": "Unknown",
    "metadata": {}
  },
  "gunShot": {
    "type": "Number",
    "value": "Unknown",
    "metadata": {}
  },
  "Jackhammer": {
    "type": "Number",
    "value": "Unknown",
    "metadata": {}
  },
  "Siren": {
    "type": "Number",
    "value": "Unknown",
    "metadata": {}
  },
  "airConditioner": {
    "type": "Number",
    "value": "Unknown",
    "metadata": {}
  },
  "carHorn": {
    "type": "Number",
    "value": "Unknown",
    "metadata": {}
  },
  "childrenPlaying": {
    "type": "Number",
    "value": "Unknown",
    "metadata": {}
  },
  "streetMusic": {
    "type": "Number",
    "value": "Unknown",
    "metadata": {}
  }
}

```

Figura 4.21: Atributos extraordinarios modelos de datos Device

4.4.1.4. MongoDB

MongoDB es un programa de base de datos orientado a documentos multiplataforma. Clasificado como un programa de base de datos NoSQL, MongoDB utiliza documentos similares a JSON. Orion emplea esta aplicación para poder almacenar la información actual de cada entidad creada. Es en ella en la que se almacena toda la información de la entidad y sus atributos, y a la cual se accede cada vez que se quiera verificar el estado actual de la entidad.

4.4.1.5. QuantumLeap

QuantumLeap es un servicio REST para almacenar, consultar y recuperar datos espacio-temporales NGSI v2. QuantumLeap convierte los datos semiestructurados de NGSI en formato tabular y los almacena en una base de datos time-series, asociando cada registro de la base de datos con un índice de tiempo y, si está presente en los datos de NGSI, una localización. Los clientes REST pueden recuperar entidades NGSI filtrando conjuntos de entidades a través de rangos de tiempo y operadores espaciales. Desde el punto de vista del cliente, estas consultas se definen en entidades NGSI en lugar de tablas de base de datos. Sin embargo, la funcionalidad de consulta disponible a través de la interfaz REST es bastante básica y las consultas más complejas generalmente requieren que los clientes usen la base de datos directamente.

La especificación de la API REST, denominada NGSI-TSDB, que QuantumLeap implementa se ha definido con el objetivo de proporcionar una interfaz REST independiente de la base de datos para el almacenamiento, consulta y recuperación de series de tiempo de entidades NGSI que podrían estar lo más cerca posible de la especificación NGSI sí mismo. Por lo tanto, NGSI-TSDB proporciona un mecanismo uniforme y familiar (para los desarrolladores de FIWARE) para acceder a datos de series de tiempo que permite implementar servicios como QuantumLeap para admitir de forma transparente múltiples backends de bases de datos. De hecho, en la actualidad QuantumLeap admite tanto CrateDB como Timescale como bases de datos de back-end. A través de los cuales se puede manipular y gestionar los datos almacenados en esas bases de datos.

4.4.1.6. CrateDB

CrateDB es un sistema distribuido de administración de bases de datos SQL que integra un almacén de datos orientado a documentos con capacidad de búsqueda. Es de código abierto, escrito en Java, basado en una arquitectura de nada compartido, y está diseñado para una alta escalabilidad e incluye componentes de Presto, Lucene, Elasticsearch y Netty.

Es el que se ha implementado en este proyecto para almacenar los datos temporales de la monitorización de cada entidad. Para realizar esto, se ha conectado este servicio a QuantumLeap a través del cual, cada vez que se configura una suscripción a una entidad en Orion por parte de Quantum esta almacena en CrateDB cada vez que se realiza una modificación en los datos observados por el “topic” al que se ha suscrito el cliente. Es de esta forma que por la que se emplea esta aplicación para poder gestionar, administrar y visualizar los datos en el tiempo que se han monitorizado en cada entidad. Es a esta base de datos a la que se conectarán los servicios de terceros para el tratamiento de los datos en formato histórico y principalmente para su visualización.

4.4.1.7. Grafana

Grafana es una aplicación web de visualización interactiva y analítica de código abierto multiplataforma. Proporciona cuadros, gráficos y alertas para la web cuando se conecta a fuentes de datos compatibles. Es ampliable mediante un sistema plug-in. Los usuarios finales pueden crear paneles de control complejos utilizando constructores de consultas interactivos.

En este proyecto se ha empleado Grafana como Agente para visualizar los datos históricos almacenados en la aplicación de CrateDB para cada entidad. De esta forma se puede acceder de forma sencilla a los datos de monitorización de la red de sensores creada.

4.4.2. Despliegue de RED

Para el despliegue de la red se ha empleado la herramienta de Docker para poder virtualizar el despliegue de forma simultánea de todas las aplicaciones necesarias para el correcto funcionamiento de la red. Este despliegue se ha realizado desde el PC del laboratorio del GTAC en el que se realizará la instanciación como servidor, al cual se conectarán los nodos/Raspberries Pi para enviar la información de clasificación.

Para un correcto funcionamiento de la demo, se ha empleado una VPN como método de autenticidad y servicio de red seguro. De esta forma el servidor se ha alojado en la red de la UPV, a la cual es necesario conectarse de forma remota a través de una VPN. De esta forma se puede modificar las aplicaciones del servidor y visualizar los datos de la base de datos de forma remota, además de conectar los sensores al broker a través de la VPN.

4.4.2.1. Paso 1: Configuración imagen Docker-Compose servidor FIWARE

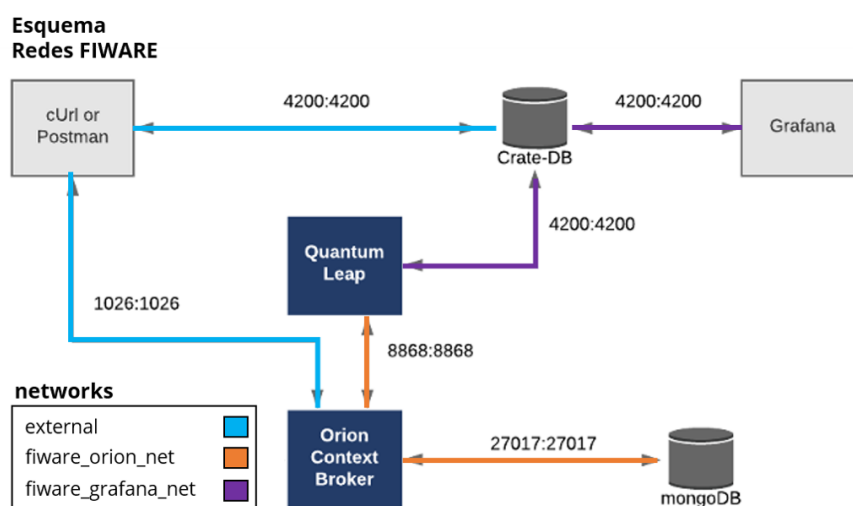


Figura 4.22: Esquema de redes del servicio FIWARE

En primer lugar se ha creado una imagen de docker-compose ?? en la que se ha realizado una instanciación a cada una de las aplicaciones descritas en la sección anterior, que forman parte de

la red. Siguiendo el esquema de la Fig. 4.17, se tiene en cuenta la asignación de puertos correspondiente a cada servicio de red. Esta utilidad permite desplegar de forma inmediata y sencilla todo el conjunto de servicios y sus relaciones que se requiere para un correcto funcionamiento de la red. De esta forma se puede implementar y desplegar en cualquier lugar toda la configuración de red empleada para este proyecto.

En la imagen docker, es necesario especificar la asignación de puertos de cada servicio y la distribución de red de cada servicio. De forma predeterminada, Compose configura una única red para su aplicación. Cada contenedor de un servicio se une a la red predeterminada y es accesible para otros contenedores en esa red y detectable por ellos en un nombre de host idéntico al nombre del contenedor.

En lugar de simplemente usar la red de aplicaciones predeterminada, se puede también especificar sus propias redes con la clave de redes de nivel superior. Esto permite crear topologías más complejas y especificar opciones y controladores de red personalizados. También puede usarlo para conectar servicios a redes creadas externamente que no son administradas por Compose.

Cada servicio puede especificar a qué redes conectarse con la clave de redes de nivel de servicio, que es una lista de nombres que hacen referencia a entradas bajo la clave de redes de nivel superior. Con esta utilidad, se ha creado dos redes distintas en las que se relacionan cada servicio según la arquitectura expuesta en la Fig. 4.22, Fig. 4.23.

```
# Redes
networks:
  fiware_orion_net:
  fiware_grafana_net:

# Volumenes para persistir la data.
volumes:
  orion_mongo_db:
  orion_mongo_configdb:
  crate_db_storage:
  grafana_storage:
```

Figura 4.23: Configuración de redes y volúmenes en la imagen de docker-compose.

Tal y como se puede observar cada servicio se encuentra conectado a las redes necesarias para poder alcanzar el resto de servicios a los que está conectado, además de la asignación de puertos correspondientes en cada caso.

Una vez se ha tenido en cuenta la configuración de red de cada servicio, otro aspecto a tener en cuenta dentro de la configuración es la versión de cada servicio. En nuestro proyecto se han asignado las versiones de cada servicio que se han testeado compatibles unas respecto de las otras. En numerosas ocasiones esta tarea suele ser tediosa hasta dar con las versiones de cada servicio para un correcto funcionamiento de la red.

En las siguientes figuras se puede observar la configuración realizada por cada servicio dentro de la imagen de docker-compose, que se ha utilizado en este proyecto como servidor. En ella se pueden observar los 5 servicios comentados en la sección anterior: Orion Context-Broker Fig. 4.24, QuantumLeap Fig. 4.25, CrateDB Fig. 4.26, MongoDB Fig. 4.27 y Grafana Fig. 4.28.

```

services:
  # Orion Context Broker
  orion:
    image: fiware/orion:2.4.0
    hostname: orion
    container_name: orion
    depends_on:
      - orion_mongo
    networks:
      - fiware_orion_net
    ports:
      - "1026:1026"
    command: -dbhost orion_mongo -noCache
    healthcheck:
      test: curl --fail -s http://127.0.0.1:1026/version || exit 1

```

Figura 4.24: Imagen docker-compose Orion

```

# Quantum Leap is persisting Short Term History to Crate-DB
quantumleap:
  image: smartsdk/quantumleap
  hostname: quantumleap
  container_name: quantumleap
  ports:
    - "8668:8668"
  networks:
    - fiware_grafana_net
    - fiware_orion_net
  depends_on:
    - cratedb
    - orion
    - orion_mongo
  environment:
    - "CRATE_HOST=http://cratedb"
  healthcheck:
    test: curl --fail -s http://127.0.0.1:8668/version || exit 1

```

Figura 4.25: Imagen docker-compose QuantumLeap

```

# Crate para utilizar Grafana
cratedb:
  image: crate:4.1
  hostname: cratedb
  container_name: cratedb
  ports:
    # Admin UI
    - "4200:4200"
    # Transport protocol
    - "4300:4300"
    - "5432:5432"
  # command: crate -Clicense.enterprise=false -Cauth.host_based.enabled=false -Ccluster.name=d
  networks:
    - fiware_grafana_net
  command: -Cdiscovery.type=single-node -Chttp.cors.enabled=true -Chttp.cors.allow-origin="*"
  volumes:
    - type: volume
      source: crate_db_storage
      target: /data

```

Figura 4.26: Imagen docker-compose CrateDB

4.4.2.2. Paso 2: Despliegue del servicio

Una vez se ha configurado la imagen de docker-compose para poder ejecutar todos los servicios necesarios para crear la red, simplemente se requiere su ejecución. Teniendo en cuenta que el

```
# MONGO DB solo para servicio de Orion
orion_mongo:
  image: mongo:3.6
  container_name: orion_mongo
  hostname: orion_mongo
  expose:
    - "27017"
  volumes:
    - type: volume
      source: orion_mongo_db
      target: /data/db
      read_only: false
    - type: volume
      source: orion_mongo_configdb
      target: /data/configdb
      read_only: false
  networks:
    - fiware_orion_net
```

Figura 4.27: Imagen docker-compose MongoDB

```
# Monitorizacion utilizando Grafana
grafana:
  image: grafana/grafana:7.0.4
  container_name: grafana
  depends_on:
    - cratedb
  ports:
    - "3003:3000"
  environment:
    - GF_INSTALL_PLUGINS=https://github.com/orchestracities/grafana-map-plugin/
  networks:
    - fiware_grafana_net
  volumes:
    - type: volume
      source: grafana_storage
      target: /var/lib/grafana
```

Figura 4.28: Imagen docker-compose Grafana

nombre de la imagen es el *default* definido por la arquitectura de docker-compose no hará falta indicar que imagen debe ejecutar, ya que ejecutará aquella imagen que se encuentre en ese directorio cuyo nombre sea **docker-compose.yml**. En caso contrario se debe indicar el nombre de la imagen haciendo uso del comando *-filename*.

Partiendo de que el nombre del docker-compose es el predeterminado, para realizar la ejecución de los contenedores de todos los servicios especificados en el docker-compose, simplemente habrá que ejecutar el comando de la Fig. ??, tras haber inicializado los servicios de docker *docker.exe* en el servidor o PC en el que se vaya a lanzar los servicios.

```
D:\UPMWorld\ml-exercises\ml-soundFeat\soundFeat\docker>docker-compose -p FIWARE up -d
Creating network "fiware_fiware_orion_net" with the default driver
Creating network "fiware_fiware_grafana_net" with the default driver
Creating cratedb ... done
Creating orion_mongo ... done
Creating grafana ... done
Creating orion ... done
Creating quantumleap ... done
```

Figura 4.29: Comando para lanzar los servicios de la imagen docker-compose

Tal y como se puede observar en la imagen, en caso de que los puertos que se han asignado a los servicios no estén ya asignados previamente en ese dispositivo por otro servicio, o que las

```

B:\UPVWorld\ml-exercises\ml-soundFeat\soundFeat\docker>docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS                               NAMES
1f9025725d3        sonarsdk/quantumleap  /bin/sh -c "python ..." 23 minutes ago     Up 23 minutes (healthy) 0.0.0.0:8668->8668/tcp             quantumleap
748c728d9831        fiware/orion:2.4.8   /usr/bin/contexthub...    23 minutes ago     Up 23 minutes (healthy) 0.0.0.0:1826->1826/tcp             orion
4d4e872162d9        grafana/grafana:7.4.4 /run.sh                  23 minutes ago     Up 23 minutes         0.0.0.0:3003->3003/tcp             grafana
122bc31577a        cratedb:4.1         /docker-entrypoint...     23 minutes ago     Up 23 minutes         0.0.0.0:4200->4200/tcp, 0.0.0.0:4300->4300/tcp, 0.0.0.0:5432->5432/tcp  cratedb
ef731fe11b5        mongo:3.6           /docker-entrypoint...     23 minutes ago     Up 23 minutes         27027/tcp                          orion_mongo

```

Figura 4.30: Comando para comprobar el estado de los contenedores instanciados en Docker

```

D:\UPVWorld\ml-exercises\ml-soundFeat\soundFeat\docker>docker-compose -p FIWARE down
Stopping quantumleap ... done
Stopping orion ... done
Stopping grafana ... done
Stopping cratedb ... done
Stopping orion_mongo ... done
Removing quantumleap ... done
Removing orion ... done
Removing grafana ... done
Removing cratedb ... done
Removing orion_mongo ... done
Removing network fiware_fiware_orion_net
Removing network fiware_fiware_grafana_net

```

Figura 4.31: Comando para finalizar la ejecución de los servicios especificados en la imagen docker-compose

versiones de las aplicaciones no estén disponibles, no debe de dar ningún problema. Siguiendo la configuración de la imagen anterior, y tras ejecutar el comando de Fig. ??, debe de desplegarse automáticamente todos los servicios instanciados para esta red. En docker-compose para lanzar los servicios se emplea el comando *up*, el cual construye, (re)crea, inicia y conecta contenedores para un servicio. A menos que ya se estén ejecutando, este comando también inicia los servicios vinculados.

El comando **docker-compose up** agrega la salida de cada contenedor (esencialmente ejecutando `docker-compose logs -f`). Cuando sale el comando, todos los contenedores se detienen. La ejecución de **docker-compose up -d** inicia los contenedores en segundo plano y los deja en ejecución.

Estos servicios son accesibles a través de la red local **localhost** o en caso de que nos encontremos dentro de la misma red o servidor cloud, con la dirección de red correspondiente, e indicando los puertos de cada servicio para conectarse a cada uno de ellos. Una forma de comprobar el estado de los servicios desplegados por la imagen docker-compose, es observar el estado de los contenedores asignados a cada servicio, tal y como se puede observar en la Fig. ?. En esa figura se puede apreciar como todos los servicios que se han instanciados están ejecutándose correctamente y cada uno en su espacio de red determinado. **STATUS** indica el estado del contenedor y el tiempo que lleva inicializado. En **PORTS** se indica la dirección de red en la que está alojado ese contenedor y finalmente **NAMES** indica el nombre que se ha asignado a cada contenedor y que es visible dentro de la red interna de los contenedores. Esto quiere decir, que entre contenedores se pueden acceder unos respecto a los otros a través de la red interna empleando el nombre del contenedor y puerto de acceso, mientras que un servicio externo a la red debe emplear la dirección de red del dispositivo en el que se ha alojado los servicios e indicar el puerto en el que está ubicado el servicio requerido para poder alcanzarlos.

En el caso de nuestro proyecto, para poder realizar una demo funcional del mismo, se ha empleado un ordenador del laboratorio del GTAC como servidor en el que se desplegará todos estos servicios contenerizados por docker. Por lo que la red estará alojada dentro de la red de la UPV y como sistema de seguridad se emplea una VPN capaz de acceder a la red de la UPV y se indica la dirección IP del ordenador del laboratorio que actúa como servidor para poder acceder a los

servicios de la red.

De esta forma los sensores deberán enviar la información a la entidad de docker creada para cada sensor e instanciada en la dirección IP del lab y puerto 1026, en el que se encuentra el servicio del broker Orion, en el que se aloja la información de cada Entidad.

4.4.2.3. Paso 3: Comprobación de servicios con Postman

Para comprobar el correcto funcionamiento de la red, se ha empleado la herramienta de Postman con el fin de poder verificar que los servicios de Orion Context-Broker y QuantumLeap funcionan correctamente. A través de Postman realizamos peticiones Http GET y POST con objeto de consultar el estado de los servicios y crear instancias dentro de los mismos.

En primera instancia se realiza una comprobación del correcto funcionamiento de los servicios principales de Orion Context-Broker y QuantumLeap. Para ello se realiza las siguientes consultas:

- Comprobar versión Orion: **http://localhost:1026/version/**
- Comprobar versión QuantumLeap: **http://localhost:8668/version**
- Comprobar las entidades creadas en Orion: **http://localhost:1026/v2/entities**

Otro aspecto a tener en cuenta, para poder distinguir el flujo de datos y comunicación proveniente de diferentes servicios dentro de la red FIWARE, se puede hacer uso de cabeceras de forma que identifica canales de comunicación diferentes dentro de la misma red. En nuestro caso las peticiones GET tanto para Orion como para QuantumLeap deben contar con este campo de cabecera, en el que se indica el servicio de fiware y el directorio Fig. 4.35

4.4.2.4. Paso 4: Crear una Entidad en Orion

Una vez comprobado que los servicios están activos y en funcionamiento, el siguiente paso es crear una primera entidad para comprobar que funciona correctamente el servicio de Fiware. Tal y como se indicó en la sección anterior se ha elegido el modelo de datos **Device** como estructura de la entidad que vamos a asignar a cada Raspberry Pi que actuará como clasificador de sonidos urbanos.

Para crear la entidad, se puede hacer a través de la herramienta de Postman, realizando una petición POST a orion, indicando en el cuerpo de la petición la estructura de datos que se ha mencionado en la Fig. ??, con los mismos atributos y valores.

En caso de no usar la herramienta de Postman para crear la entidad, se puede realizar un script en cualquier lenguaje que permita realizar peticiones HTTP. En este caso se ha realizado una petición POST con python, empleando la librería **requests** y la librería **json** para introducir los datos que formarán parte del *body* de la petición tal y como se puede apreciar en Fig. 4.36 y Fig. 4.37.

Dos aspectos fundamentales se han de tener en cuenta para crear una correcta Entidad dentro de una red FIWARE.

- En primer lugar, asegurarnos de que la dirección de la petición POST corresponde a la de Orion Context-Broker **http://IP-server:1026/v2/entities/**.

► Obtaining Version Information


GET ▼ http://localhost:1026/version/

Params Authorization Headers (6) Body Pre-request Script

Query Params

KEY
Key

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON ▼ 

```

1  {
2    "orion": {
3      "version": "2.4.0-next",
4      "uptime": "0 d, 0 h, 0 m, 8 s",
5      "git_hash": "4f26834ca928e468b091729d93dabd22108a2690",
6      "compile_time": "Tue Mar 31 15:41:02 UTC 2020",
7      "compiled_by": "root",
8      "compiled_in": "d99d1dbb4d9e",
9      "release_date": "Tue Mar 31 15:41:02 UTC 2020",
10     "doc": "https://fiware-orion.rtdf.io/"
11   }
12 }

```

Figura 4.32: Comprobación versión Orion Context-Broker

- En segundo lugar, asegurarnos de que la estructura de datos sigue los modelos de datos estandarizados según el tipo de objeto que se va a monitorizar.

► Check QuantumLeap


GET ▼ http://localhost:8668/version

Params Authorization Headers (8) Body Pre-request Scr

Query Params

KEY
Key

Body Cookies Headers (4) Test Results

Pretty Raw Preview Visualize JSON ▼ 

```
1 {  
2   "version": "0.8.0"  
3 }
```

Figura 4.33: Comprobación versión QuantumLeap

► Retrieving Context Information


GET http://localhost:1026/v2/entities

Params Authorization Headers (8) Body Pre-request Script Tests Settings

Query Params

KEY
Key

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON ▼ 

```

1  [
2  {
3    "id": "urn:ngsi-Id:AcousticNode:001",
4    "type": "Device",
5    "Drilling": {
6      "type": "Number",
7      "value": "7.704217e-09",
8      "metadata": {}
9    },
10   "Jackhammer": {
11     "type": "Number",
12     "value": "6.016408e-15",
13     "metadata": {}
14   },
15   "Siren": {
16     "type": "Number",
17     "value": "1.8000566e-05",
18     "metadata": {}
19   },
20   "address": {
21     "type": "PostalAddress",
22     "value": {
23       "streetAddress": "Camí de Vera, s/n Edificio 8G",
24       "addressRegion": "Valencia",
25       "addressLocality": "Valencia",
26       "postalCode": "46022"
27     },
28     "metadata": {
29       "verified": {
30         "type": "Boolean",
31         "value": "true"
32       }
33     }
34   }
35 }
36 ]
    
```

Figura 4.34: Comprobación de Entidades disponibles en Orion

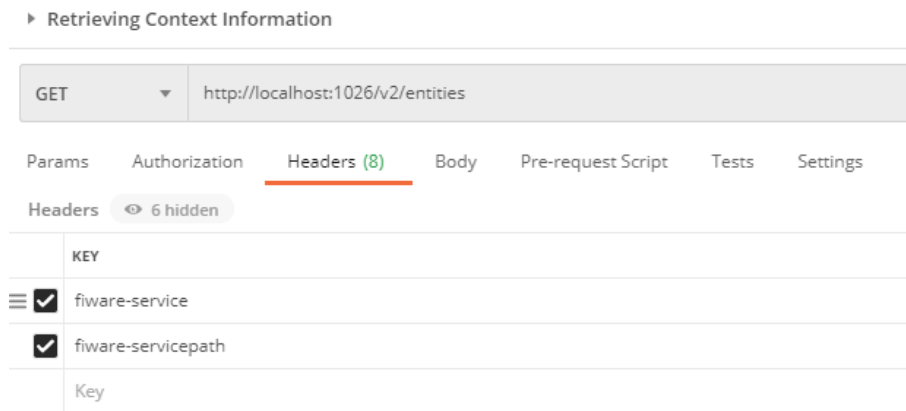


Figura 4.35: Cabecera empleada en el contexto FIWARE de este proyecto

```
# UNIVERSITAT POLITÈCNICA DE VALÈNCIA
# Author: David Salvo Gutiérrez

import json
import datetime

currentDT = datetime.datetime.now()
print (str(currentDT))

# passing data classification to json format
data = {
    "type": "Device",
    "id": "urn:ngsi-ld:AcousticNode:002",
    "source": {
        "type": "URL",
        "value": "https://gtac.webs.upv.es/"
    },
    "dataProvider": {
        "type": "URL",
        "value": "https://gtac.webs.upv.es/"
    },
    "category": {
        "type": "Text",
        "value": "sensor"
    },
}
```

Figura 4.36: Creación entidad en Python I

```
## Making a POST request to Orion Broker
import requests

# defining the api-endpoint
API_ENDPOINT = "http://localhost:1026/v2/entities/"

# headers
headers_string={
    'Content-Type':'application/json',
    'fiware-service':'openiot',
    'fiware-servicepath':'/'
}

# data to be sent to api
payload = json.dumps(data)

# request post (using json payload)
x = requests.post(url= API_ENDPOINT, data= payload, headers= headers_string,)
```

Figura 4.37: Creación entidad en Python II

4.4.2.5. Paso 5: Crear suscripción de QuantumLeap a Orion para esa Entidad o conjunto de Entidades

Una vez se ha creado una entidad que representa el objeto del mundo físico que se va a monitorizar, se requiere crear una suscripción a través de la cual se podrá obtener el histórico de datos que se actualizan de la entidad en el transcurso del tiempo. Recordemos que Orion no almacena el histórico de valores de los atributos, por lo que mantiene únicamente la información más reciente de cada atributo de la entidad.

Por este motivo, se implementa QuantumLeap, una herramienta que es time-persistence por lo que almacena cada cambio que se realiza sobre los atributos de la entidad y de esta forma poder observar el histórico de datos generados por esa entidad. Para hacer esto posible, se debe crear una suscripción a la entidad o conjunto de entidades que se quieren monitorizar y especificar qué atributos se van a monitorizar y cada cuanto se va a realizar una captura de datos (trigger). Toda esta información es la que recoge el conjunto de datos requeridos para una monitorización específica de los sensores y nodos de la red FIWARE. En nuestro caso, el objetivo es monitorizar de forma conjunta todos los sensores de la red, y obtener los valores de clase acústica detectada y el porcentaje de clasificación por cada clase, cada vez que se realice una nueva clasificación. Es decir, cada vez que se realice una nueva clasificación se modifica la fecha de actualización de la entidad en el broker así como los valores de la clase detectada y los porcentajes de clasificación por clase. De esta forma, se emplea como activador de la suscripción el atributo **modDate** con el cual se realiza una captura de la información de la entidad cada vez que ese atributo es modificado.

Una vez definido el activador de la suscripción, sólo hay que establecer los atributos que se van a monitorizar y capturar los datos cada vez que se cambie la fecha de clasificación de la entidad en el broker, tal y como se ha especificado en la suscripción que vamos a utilizar en nuestro proyecto. En la Fig. 4.38 se puede observar el formato de suscripción que se ha empleado para las entidades creadas en Orion. Para entender mejor su funcionamiento analizamos en detalle los atributos y valores que se han empleado en la estructura de nuestra suscripción, que servirá para monitorizar los sensores acústicos de nuestra red.

- **entities:idPattern** representa el nombre de las entidades que se van a registrar con esta suscripción. Es decir, las entidades que se indique en el idPattern, serán aquellas que se monitorizará los cambios por medio de la suscripción, aquellas que no se encuentren representadas por el idPattern no aparecerán monitorizadas. En el ejemplo, se puede observar como se asigna la suscripción a toda aquellas entidades cuyo id sea el siguiente: **urn:ngsi-ld:AcousticNode.***, cogiendo así todas aquellas entidades que cumplan con el nombre anterior sin tener en cuenta el valor contiguo a AcousticNode.*. En nuestro caso se han creado dos entidades cuyos id son: urn:ngsi-ld:AcousticNode.000 y urn:ngsi-ld:AcousticNode.001, por lo que ambos serán monitorizados por la suscripción del ejemplo.
- **conditions:attrs** es el campo en el que se indica el nombre del atributo de la entidad indicada por el idPattern que se empleará como *trigger* para la monitorización. Es decir, cada vez que se produzca un cambio en la entidad en el Broker del atributo indicado en este campo, en nuestro caso **modDate**, se obtendrá el valor actual de todas aquellos atributos indicados en la suscripción.
- **notification:attrs** indica todos aquellos atributos que van a ser capturados en la entidad cada vez que se obtenga una actualización del atributo indicado en *conditions:attrs*.

► Create a Subscription QuantumLeap (Product)

POST http://localhost:1026/v2/subscriptions/

Params Authorization Headers (11) **Body** Pre-request Script Tests Settings

● none ● form-data ● x-www-form-urlencoded ● **raw** ● binary ● GraphQL **JSON** ▼

```

1  {
2    "description": "Notify QuantumLeap of 001 Urban Acoustic Classification",
3    "subject": {
4      "entities": [
5        {
6          "idPattern": "urn:ngsi-ld:AcousticNode.*",
7          "type": "Device"
8        }
9      ],
10     "condition": {
11       "attrs": [
12         "modDate"
13       ]
14     }
15   },
16   "notification": {
17     "attrs": [
18       "noiseClass",
19       "Drilling",
20       "Jackhammer",
21       "Siren",
22       "airConditioner",
23       "carHorn",
24       "childrenPlaying",
25       "dogBark",
26       "engineIdling",
27       "gunShot",
28       "streetMusic",
29       "location",
30       "Geohash"
31     ],
32     "http": {
33       "url": "http://quantumleap:8668/v2/notify"
34     },
35     "metadata": [
36       "dateCreated",
37       "dateModified"
38     ]
39   }
40 }

```

Figura 4.38: Estructura suscripción de QuantumLeap a Orion

- **http:url** indica la dirección en la que debe almacenar la información capturada de la entidad, siguiendo la estructura de la suscripción.
- **metadata** se indica aquella información extra que se quiere añadir a la captura de datos de la entidad y que forma parte del sistema. No son atributos de la entidad, son indicadores predefinidos por el sistema de FIWARE. En el caso de **dateCrated** y **dateModified** indica las fechas de creación y modificación de esa entidad.

De una forma más clara, esta suscripción se interpreta de la siguiente manera: se va a crear una suscripción a todas aquellas Entidades cuyo id cumpla con el patron **urn:ngsi-ld:AcousticNode.*** y cuyo tipo de entidad sea **Device**. De esta forma se especifica aquellos sensores que se quieren monitorizar. Se quiere monitorizar todos los atributos que representan cada una de las clases del clasificador acústico *Drilling, Jackhammer, Siren, airConditioner, carHorn, childrenPlaying, dogBark, engineIdling, gunShot, streetMusic*, el nombre de la clase clasificada por el clasificador-sensor *noiseClass* y finalmente los atributos que representan la localización del sensor en caso de que el sensor sea un sensor móvil es de gran utilidad este atributo. Todos estos datos serán capturados de la entidad o entidades cada vez que se detecte una modificación del atributo *modDate* en la entidad del Broker. El resultado de esto, se puede observar en la Fig. 4.40.

Automáticamente, el servicio de QuantumLeap de la red FIWARE genera una tabla en la aplicación CrateDB en la que almacena todos los datos obtenidos en la suscripción. La tabla que crea en CrateDB tiene tantas columnas como atributos se haya especificado en la suscripción. Siguiendo el esquema de la suscripción mencionada antes, la tabla queda de la siguiente manera.

Para poder visualizar los datos en la aplicación de **CrateDB**, recordemos que en la imagen de Docker de este servicio se especificó como puerto asignado a este servicio el 4200. De esta forma, si introducimos en el navegador, la dirección IP seguida del puerto 4200 podremos acceder al servicio de bases de datos de QuantumLeap y de esta forma gestionar y manipular los datos que se obtienen a través de la suscripciones creadas en QuantumLeap (*http://IP-server:4200/*). En esta aplicación se puede modificar la estructura de las tablas, así como crear nuevas tablas asociadas a los datos que se monitorizan de los sensores.

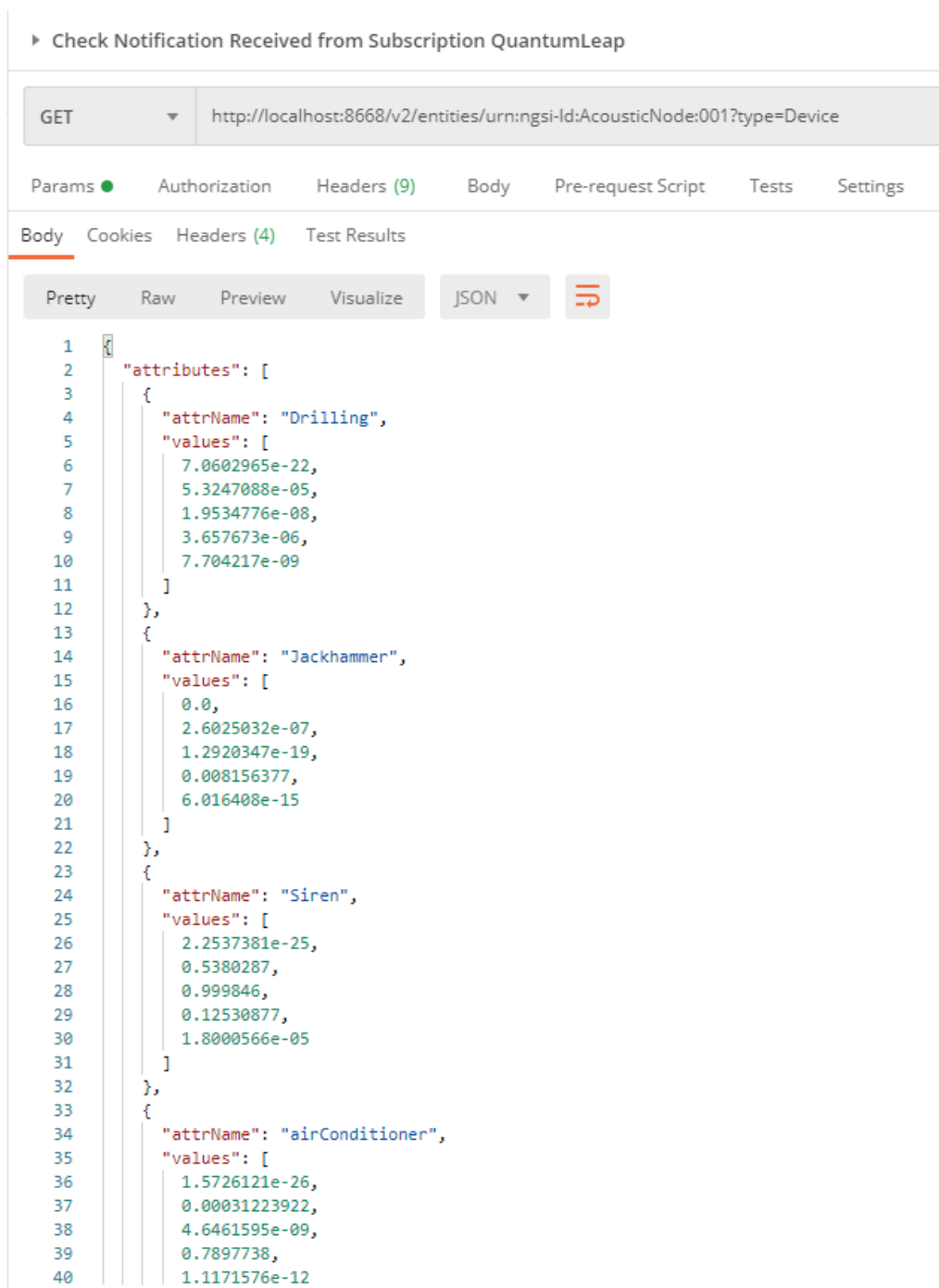
Para esta Demo no es necesario hacer ninguna modificación sobre las tablas, simplemente accederemos al servicio de CrateDB para verificar si el contenido de la tabla está acorde al contenido de la suscripción. Acto seguido, se realizará el dashboard para visualizar los datos reflejados en la tabla.

4.4.2.6. Paso 6: Crear Dashboard en Grafana

Una vez se ha comprobado el correcto funcionamiento de la suscripción y que se almacena los datos correctamente en la base de datos del servidor, empleamos el servicio de Grafana para poder visualizar los datos y crear un dashboard en el que reflejar correctamente la monitorización de los sensores.

Recordemos que la imagen de Grafana, que se ha instanciado en docker, cuenta con un puerto de gestión asignado al 3003. De esta forma, al igual que hacíamos para acceder al servicio de CrateDB, accedemos a Grafana a través de la dirección IP del servidor y el puerto asignado al servicio de Grafana en el 3003 (*http://IP-server:3003/*). Cuando se accede por primera vez al servicio, se solicita un nombre de usuario y contraseña. Se introduce aquellas que más se prefiera y accedemos a la página inicial de la aplicación Grafana Fig. 4.41.

Para representar gráficamente los datos de la base de datos CrateDB, se necesita en primera instancia crear una conexión en Grafana hacia la base de datos que tenemos en CrateDB. Para esto, cuando se creó la imagen del servicio de CrateDB se especificó un puerto de operación asignado en el 5432. Por ello, para conectarnos al servicio de CrateDB desde Grafana, habrá que crear una fuente de datos asociada a nuestro servicio de CrateDB. La forma de alcanzar un servicio que forma parte del mismo despliegue en el servidor, es a través del nombre de su contenedor al igual que se hizo



▶ Check Notification Received from Subscription QuantumLeap

GET http://localhost:8668/v2/entities/urn:ngsi-Id:AcousticNode:001?type=Device

Params ● Authorization Headers (9) Body Pre-request Script Tests Settings

Body Cookies Headers (4) Test Results

Pretty Raw Preview Visualize JSON ↕

```

1  {
2    "attributes": [
3      {
4        "attrName": "Drilling",
5        "values": [
6          7.0602965e-22,
7          5.3247088e-05,
8          1.9534776e-08,
9          3.657673e-06,
10         7.704217e-09
11        ]
12      },
13      {
14        "attrName": "Jackhammer",
15        "values": [
16          0.0,
17          2.6025032e-07,
18          1.2920347e-19,
19          0.008156377,
20          6.016408e-15
21        ]
22      },
23      {
24        "attrName": "Siren",
25        "values": [
26          2.2537381e-25,
27          0.5380287,
28          0.999846,
29          0.12530877,
30          1.8000566e-05
31        ]
32      },
33      {
34        "attrName": "airConditioner",
35        "values": [
36          1.5726121e-26,
37          0.00031223922,
38          4.6461595e-09,
39          0.7897738,
40          1.1171576e-12

```

Figura 4.39: Información almacenada por QuantumLeap de la suscripción anterior

en la suscripción. Es decir, se indica cratedb para acceder a al contenedor del servicio CrateDB desde dentro de la red desplegada, al igual que cuando se crea una suscripción en Orion se indica que notifique en quantumleap en vez de en localhost, para alcanzar el servicio de QuantumLeap desde la red interna. Teniendo en cuenta esto, la configuración de la Fuente de Datos que vamos a emplear en nuestro Dashboard de Grafana se puede observar en la Fig. 4.42

Esquema	
Nombre	Tipo
entity_id	TEXT
entity_type	TEXT
time_index	TIMESTAMP WITH TIME ZONE
fiware_servicepath	TEXT
noiseclass	TEXT
drilling	REAL
jackhammer	REAL
siren	REAL
airconditioner	REAL
carhorn	REAL
childrenplaying	REAL
dogbark	REAL
engineidling	REAL
gunshot	REAL
streetmusic	REAL
location	GEO_SHAPE
geohash	TEXT
location_centroid	GEO_POINT

Figura 4.40: Tabla asociada a la suscripción Fig. 4.38, aplicación CrateDB

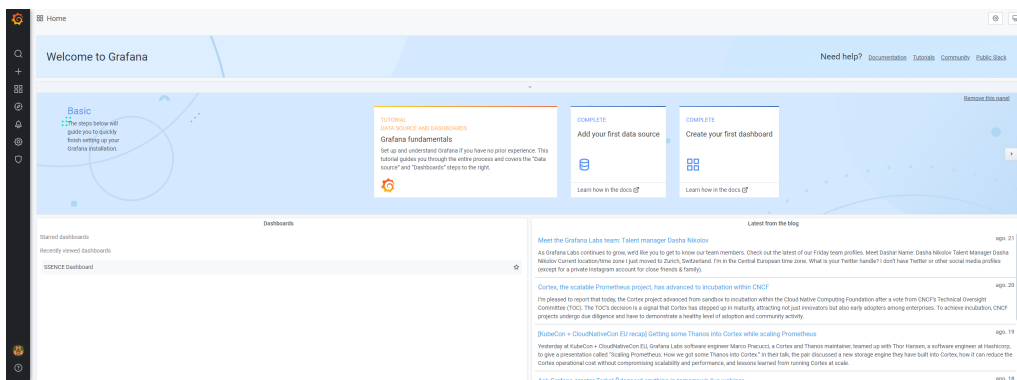


Figura 4.41: Aplicación de Grafana

Tras haber instanciado correctamente la conexión a nuestra base de datos, se crea un nuevo dashboard en el que se irán añadiendo los diferentes paneles que ofrece la aplicación de Grafana, que a través de consultas SQL se podrá acceder a la información de la base de datos a la que nos hemos conectado (CrateDB). En la Fig. 4.43 se puede apreciar el resultado obtenido tras crear diferentes paneles asociados a la información que se monitoriza de los sensores. El dashboard cuenta

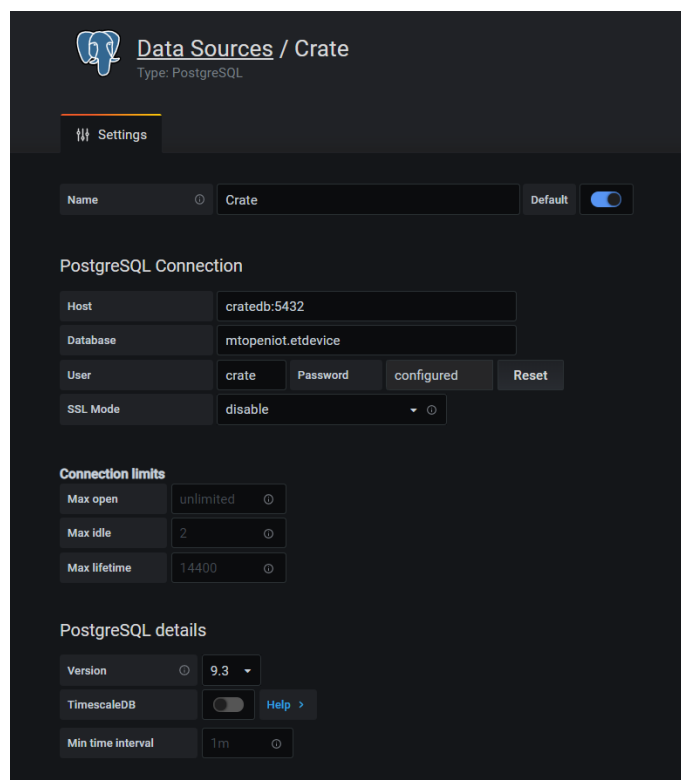


Figura 4.42: Asignación de nuestro Data Source para crear el Dashboard

con un panel en el que se indica el porcentaje de clasificación por clase de la última clasificación detectada en cualquier de los nodos monitorizados, se puede apreciar los resultados de clasificación en tiempo real de los sensores monitorizados, se puede observar la localización de los sensores que se están monitorizando en panel del mapa a través del atributo geohash, además de poder observar la etiqueta de tiempo de cada clasificación y el sensor al que pertenecen para poder obtener una información global de lo que la red de sensores está monitorizando.

Para crear cada panel, se han empleado utilidades diferentes y consultas a la base de datos distintas. En las Fig. 4.44, Fig. 4.45, Fig. 4.46, Fig. 4.47, Fig. 4.48 se puede observar la configuración empleada en cada panel para poder obtener el dashboard final de nuestra red de sensores. Con la configuración final del dashboard ya habremos completado el diseño y ejecución de una red de sensores capaces de clasificar 10 clases de eventos acústicos en entornos urbanos, poder almacenarlos y visualizarlos en tiempo real.

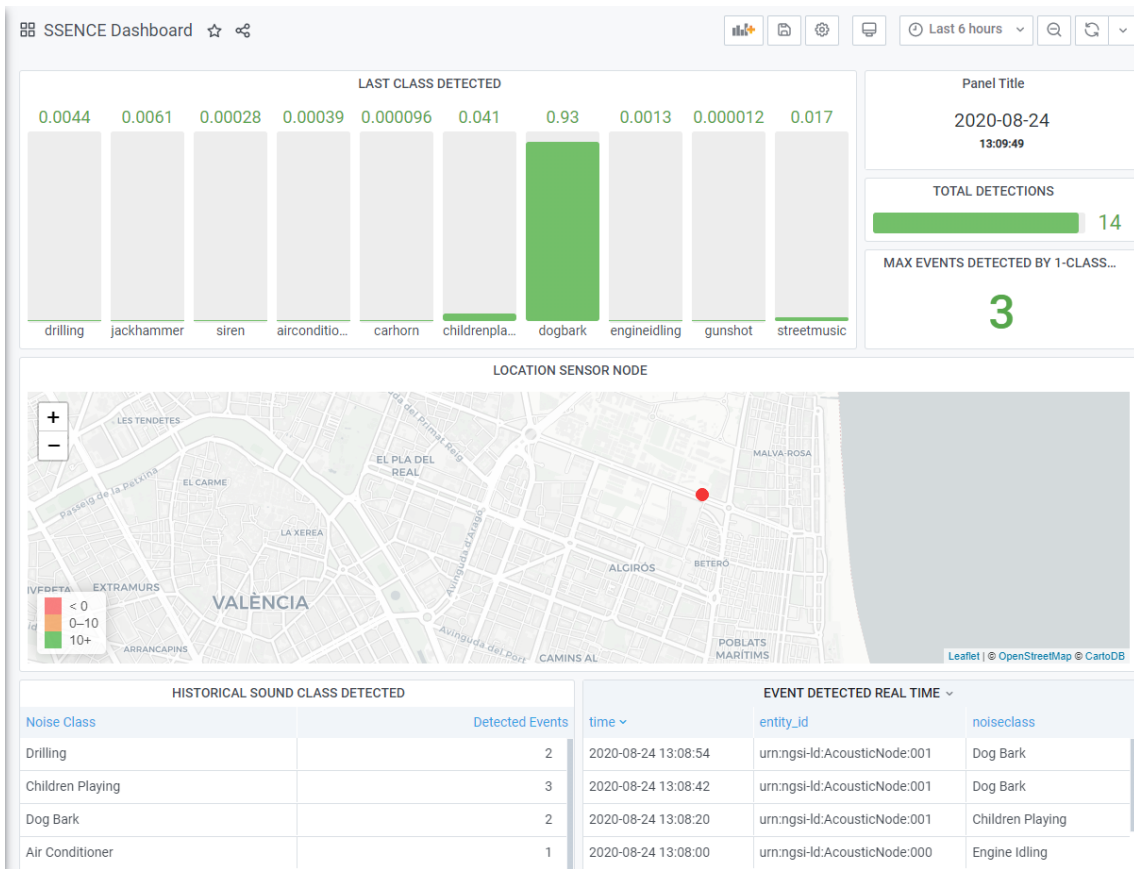


Figura 4.43: Dashboard de la red creado en Grafana

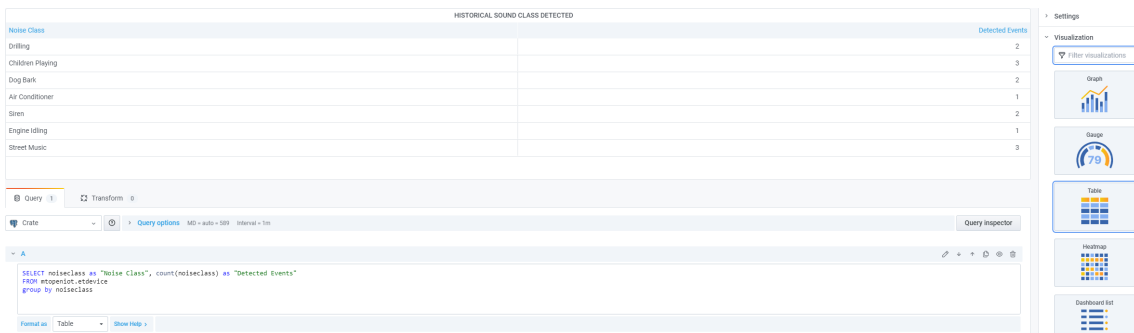


Figura 4.44: Panel que indica la cantidad de eventos detectados por cada clase clasificada.

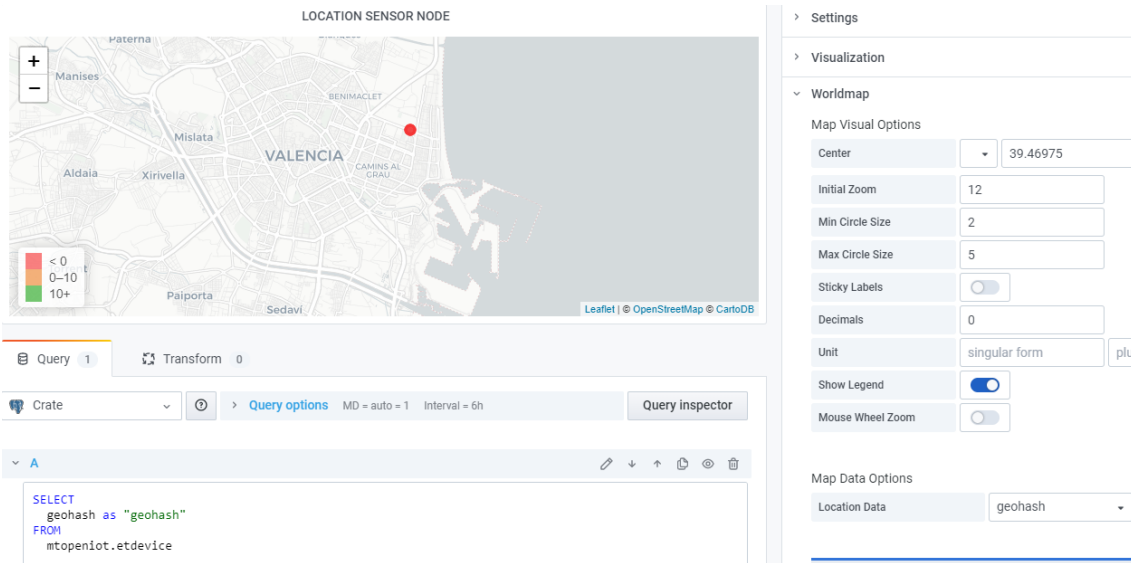


Figura 4.45: Panel para visualizar la localización de los sensores en el mapa.

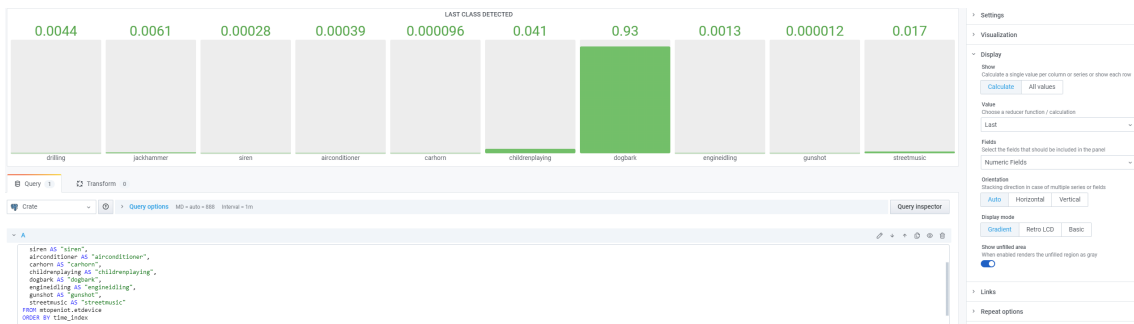


Figura 4.46: Panel para visualizar el porcentaje de clasificación por cada clase.



Figura 4.47: Panel que indica el número total de clases detectadas por la red.

EVENT DETECTED REAL TIME		
time	entity_id	noiseclass
2020-08-24 12:51:47	urn:ngsi-Id:AcousticNode:001	Children Playing
2020-08-24 12:52:16	urn:ngsi-Id:AcousticNode:001	Children Playing
2020-08-24 12:52:56	urn:ngsi-Id:AcousticNode:001	Air Conditioner
2020-08-24 12:57:01	urn:ngsi-Id:AcousticNode:001	Drilling

The screenshot shows a query interface with a table of event data and the SQL query used to retrieve it. The table has three columns: time, entity_id, and noiseclass. The query is a SELECT statement that filters events based on a time filter and orders them by time.

```
SELECT
  time_index AS "time",
  entity_id,
  noiseclass
FROM mtopeniot.etdevice
WHERE
  $__timeFilter(time_index)
ORDER BY 1
```

Figura 4.48: Panel en el que se visualiza en tiempo real cada evento clasificado y el sensor responsable de esa clasificación.

4.4.2.7. Esquema Funcional de la red

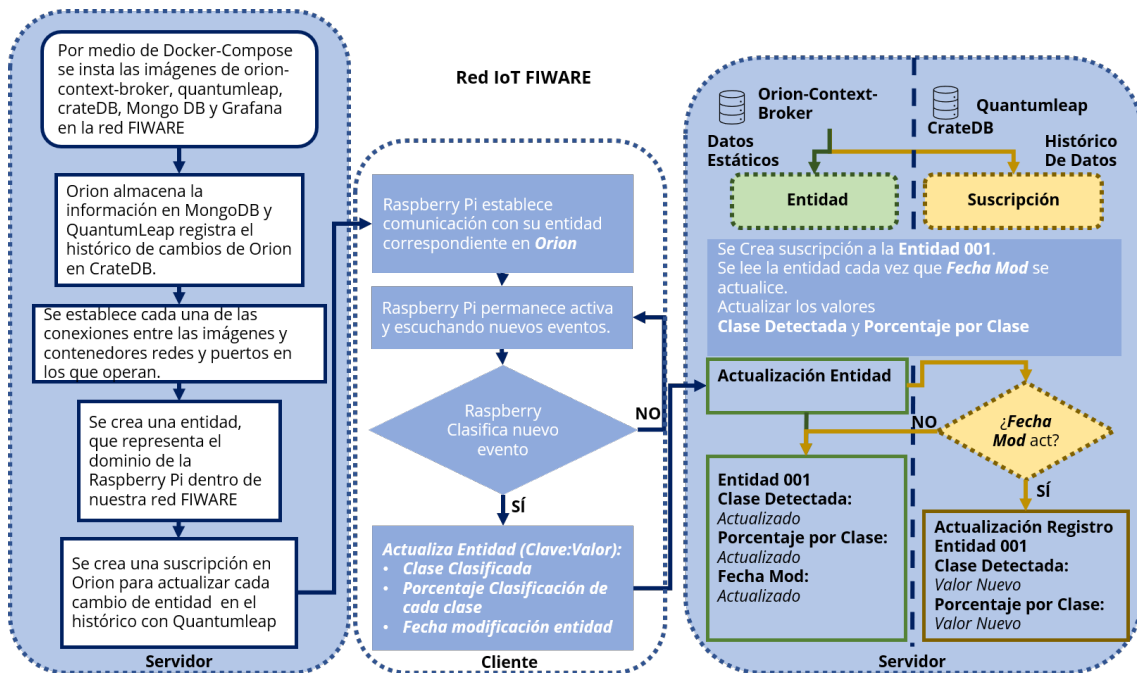


Figura 4.49: Esquema Funcional de la red al completo.

A modo resumen, en la siguiente figura Fig. 4.49 se refleja todo el flujo de tareas que se realizan para desplegar la red empleada en este proyecto. Desde que se despliega la imagen Docker en el servidor hasta que la Raspberry Pi envía un nuevo evento clasificado al Borker y este lo almacena a través de QuantumLeap en la base de datos CrateDB y poder después visualizarlo en el dashboard creado en Grafana.

Como se puede observar en el esquema, de izquierda a derecha se sigue todo el flujo de tareas que se ha explicado detalladamente en los apartados anteriores con los pasos del [1 – 6]. De esta manera, empleando este esquema como resumen y estructura global de la red empleada en este proyecto.

4.5. Raspberry Pi

Una vez se ha comprobado el correcto funcionamiento del proyecto en el PC. Se dispone a implementar el clasificador de sonidos urbanos en la Raspberry Pi. Para poder llevarlo a cabo, hay que instalar en la raspberry pi todas las dependencias y librerías necesarias para ejecutar el clasificador en la propia Raspberry Pi.

4.5.1. Configuración Raspberry Pi

En primer lugar para poder usar de una forma más sencilla la Raspberry Pi, sin necesidad de conectarla al monitor ni a periféricos como mando y ratón emplearemos el servidor VNC Server. A través de este servidor nos conectaremos desde nuestro ordenador principal a la Raspberry Pi.

- Paso 1: Activar la opción VNC en la Raspberry Pi, **Preferences > Raspberry Pi Configuration > VNC (Enable)**.
- Paso 2: Una vez activado obtenemos la dirección IP de nuestra interfaz eth0 de la Raspberry Pi, que es la que estará conectada directamente a la misma red del PC de control.
- Paso 3: En el PC instalamos VNC Viewer.
- Paso 4: Abrimos en el PC el VNC Viewer e introducimos la IP de la Raspberry Pi a la que nos queremos conectar.
- Paso 5: Si no se han cambiado los credenciales de la Raspberry Pi, por defecto, el usuario es **pi** y la contraseña **raspberrypi**. Una vez nos identificamos tras haber solicitado la conexión desde VNC Viewer en el PC a la IP de la Raspberry Pi, ya podremos controlar el dispositivo de forma remota. Sólo deberá estar encendido y conectado por cable LAN a la misma red del PC de control.

Siguiendo los pasos anteriores se obtiene un manejo más cómodo de los dispositivos desde un mismo dispositivos y usando sólo unos periféricos. El siguiente paso es instalar todo el entorno necesario para poder ejecutar los scripts que se han preparado para clasificar el sonido urbano. El entorno de desarrollo está basado en python versión 3.6 e instalaremos las dependencias de python necesarias para el correcto funcionamiento del proyecto.

La tarea de la Raspberry Pi se centra primordialmente en capturar 4 segundos de grabación de sonido urbano a través del micrófono y posteriormente clasificar ese fragmento de audio en una de las 10 clases que conoce nuestro modelo entrenado y enviarlo al broker de FIWARE (Orion) para poder almacenar y visualizar esa información en nuestro dashboard.

La configuración del entorno se ha centrado en la instalación de las siguientes dependencias:

- Tensorflow version 1.14.0
- Keras
- llvmlite

- librosa
- numba
- pandas
- numpy
- scipy

Existen diversos métodos para instalar el entorno completo que se requiere para el correcto funcionamiento del proyecto. En este caso se va a exponer el método empleado para instalar el entorno que ha sido testeado en Raspberry Pi 3 Model B y Raspberry Pi 4 Model B (4GB RAM).

El entorno de Raspberry Pi es más complejo que el del PC convencional, teniendo en cuenta que la arquitectura de la Raspberry Pi es **Armv1**. Por lo que cuando se intenta implementar entornos de desarrollo de alto nivel, se depende en gran medida del desarrollador de que existan algunas compatibilidades con esa arquitectura para poder hacer que funcione como en cualquier entorno normal. En nuestro caso, Python 3,6, Tensorflow y Keras son compatible con la arquitectura de nuestra Raspberry Pi 3 Modelo B. El problema es que la librería Librosa que se emplea en nuestro proyecto para el tratamiento de la señal de audio que forma parte de los *features* de nuestra CNN, tiene una instalación tediosa y compleja y se requiere de otras herramientas para su instalación. Por este motivo presentamos **Berryconda**, el entorno conda compatible con nuestra raspberry a través del cual vamos a instalar Librosa y poder emplearlo correctamente junto con el resto de la librerías necesarias.

El punto de partida, tras haber iniciado por primera vez nuestra Raspberry Pi con SO Raspbian, pasamos a instalar Berryconda para poder instalar Librosa como primera librería. Será necesario descargar Berryconda siguiendo este enlace (<https://github.com/jjhelmus/berryconda>). Una vez descargado el archivo *.sh* compatible con nuestra arquitectura, haremos que sea ejecutable y lo instalaremos en nuestro usuario tal y como indica el tutorial.

Comprobación del entorno

```
chmod +x Berryconda3-2.0.0-Linux-armv7l.sh #Convierte en ejecutable
./Berryconda3-2.0.0-Linux-armv7l.sh #Instalar en la ruta /home/pi/berryconda3
reboot #reiniciamos
conda install -c numba numba #instalamos librerías necesarias para Librosa
```

Una vez instalado sin derechos de administrados, no emplear la instancia de administrador **sudo**. Se instala las librerías necesarias para instalar correctamente Librosa. Acto seguido, será necesario instalar todas las dependencias necesarias para el correcto funcionamiento del sistema: Tensorflow, Keras, Numpy y Librosa. En esta caso sí que instalamos las dependencias como administrador.

Comprobación del entorno

```
sudo apt-get update
```

```

sudo apt-get install -y build-essential tk-dev libncurses5-dev libncursesw5-dev
libreadline6-dev libdb5.3-dev libgdbm-dev libsqlite3-dev libssl-dev libbz2
-dev libexpat1-dev liblzma-dev zlib1g-dev libffi-dev

sudo apt-get install libatlas-base-dev python3-numpy libblas-dev liblapack-dev
python3-dev gfortran python3-setuptools python3-scipy

conda install scikit-learn

reboot

```

Instalamos el paquete de Scikit-learn necesario en el entorno y reiniciamos el dispositivos para seguir con la configuración del sistema.

Comprobación del entorno

```

pip install librosa # Instalamos Librosa

pip list # Observamos que tenemos librosa correctamente instalado

sudo apt update

pip install --upgrade pip

pip list

sudo su - # Nos logueamos como administradores del sistema

pip3 install --user --upgrade tensorflow # Instalamos Tensorflow en el sistema
python3 -c 'import tensorflow as tf; print(tf.__version__)' # Comprobamos que
se ha instalado correctamente

pip3 install pandas # Instalamos Pandas

# En caso de que no funcione el entorno, habrá que instalar Librosa como
administrador
# Para ello se prueba instanciar el entorno Berryconda con el nuevo directorio /
root/berryconda3 en vez de /home/pi/berryconda3

pip install librosa # desde el directorio root (sudo su -)

```

La librería de librosa suele tardar aproximadamente una hora en el proceso de instalación, por lo que no hay que preocuparse de la demora. Esta librería es fundamental, ya que se usa en todo el estudio de extracción de características acústicas, que se realiza previamente a la clasificación.

Entorno Virtual

En caso de que se quiera crear un entorno virtual en vez de instalar el entorno en el sistema raíz, se haría de la siguiente manera.

```

# Para crear un entorno virtual nuevo, elige un intérprete de Python y crea un
directorio ./venv para contenerlo:

sudo pip3 install -U virtualenv
virtualenv --system-site-packages -p python3 env

# Para activar el entorno virtual, usa un comando de shell específico:

```



```
source env/bin/activate

# Cuando virtualenv está activo, la solicitud del shell tiene el prefijo (venv)
.

# Instala paquetes dentro de un entorno virtual sin afectar la configuración
del sistema host. Primero, actualiza pip:

pip install --upgrade pip

pip list

# Para salir de virtualenv más tarde:

deactivate # no salir hasta acabar de instalar las dependencias necesarias
para instalar correctamente el entorno.
```

En nuestro caso se ha realizado una instalación sin entorno virtual desde la cual poder instalar en el nodo todas las librerías necesarias para el correcto funcionamiento del proyecto.

Cabe destacar que todo el contenido del proyecto, desde los scripts hasta el tutorial correspondiente para desplegar y testear el correcto funcionamiento del mismo se encuentra disponible en un repositorio de GitHub que se puede acceder a través del siguiente enlace: **Github Repository**.

Funcionamiento

Una vez todo el entorno está instalado y operativo, comprobamos que el funcionamiento del modelo entrenado en el PC es el correcto e idéntico en la Raspberry Pi. Para ello empleamos el mismo script que se empleó para testear la correcta clasificación de un evento acústico que formase parte del segmento de la base de datos que no se empleo para el entrenamiento. En este script, se importa el audio que se quiere clasificar y que forma parte de una de las 10 clases, y se clasifica con el modelo ya entrenado. Obteniendo resultados como los de las Fig. 4.50, Fig. 4.51 y Fig. 4.52.

Una vez comprobado que el modelo funciona correctamente, y que el entorno de desarrollo para ejecutar el modelo funciona correctamente, se pasa a implementar el script de grabación de audio con el micrófono. Es decir, añadiremos al script del modelo `sound-class-raspberry.py` una función cuyo objetivo consiste en capturar 4 segundos de grabación del micrófono, cada vez que se llame a esa función. Ese fragmento se introduce posteriormente al clasificador para obtener la clase a la que más se asemeja o pertenece el evento acústico capturado.

Esa función se ha reciclado de un proyecto realizado por mi compañero de grupo Daniel Sanz, en el cual a través de la interfaz del micrófono la Raspberry Pi captura 4 segundos y los almacena en un `.wav`, que será importado al clasificador el cual extraerá su espectrograma de Mel y posteriormente será clasificado por el modelo ya entrenado, aproximando a la clase que mayor se asemeje.

En resumen, la Raspberry Pi se ha configurado de forma que pueda capturar 4 segundos de sonido y clasificarlo en una de las 10 clases con las que ha sido entrenado el modelo. Una vez puesta en marcha el clasificador y el proceso de captura del sonido, se ha comprobado que dentro de la misma LAN los sensores envían correctamente los datos de clasificación y la clase clasificada Orion Context-Broker, explicado en la sección anterior, de forma que se puede visualizar los

```

pi@raspberrypi: ~
Archivo Editar Pestañas Ayuda
b = a[a_slice]

Original class: Car Horn
The predicted class is: Car Horn

Air Conditioner      : 0.00000000000111566702057364075529
Car Horn             : 0.99989664554595947265625000000000
Children Playing     : 0.00000088357097638436243869364262
Dog Bark             : 0.00006589754775632172822952270508
Drilling             : 0.0000000770464403387904894771054
Engine Idling        : 0.00000028525897732833982445299625
Gun Shot             : 0.00000000000000028360251260850077
Jackhammer           : 0.000000000000000601266930531528174
Siren                : 0.00001800891186576336622238159180
Street Music         : 0.00001840102413552813231945037842

Classification Duration: 0:00:20.892207
root@raspberrypi:/home/pi/ml-exercise/ml-soundFeatV2/soundFeat#

```

Figura 4.50: Raspberry clasificando el evento acústico Cláxon

```

pi@raspberrypi: ~
Archivo Editar Pestañas Ayuda
b = a[a_slice]

Original class: Siren
The predicted class is: Siren

Air Conditioner      : 0.00000000464027039015491027384996
Car Horn             : 0.00000001728573906234487367328256
Children Playing     : 0.00000004405102771443125675432384
Dog Bark             : 0.00005393630272010341286659240723
Drilling             : 0.00000001956304096495387057075277
Engine Idling        : 0.00000004350610538494947832077742
Gun Shot             : 0.000000000000000000000037347441946
Jackhammer           : 0.00000000000000000012945114215104
Siren                : 0.99984598159790039062500000000000
Street Music         : 0.00009986794611904770135879516602

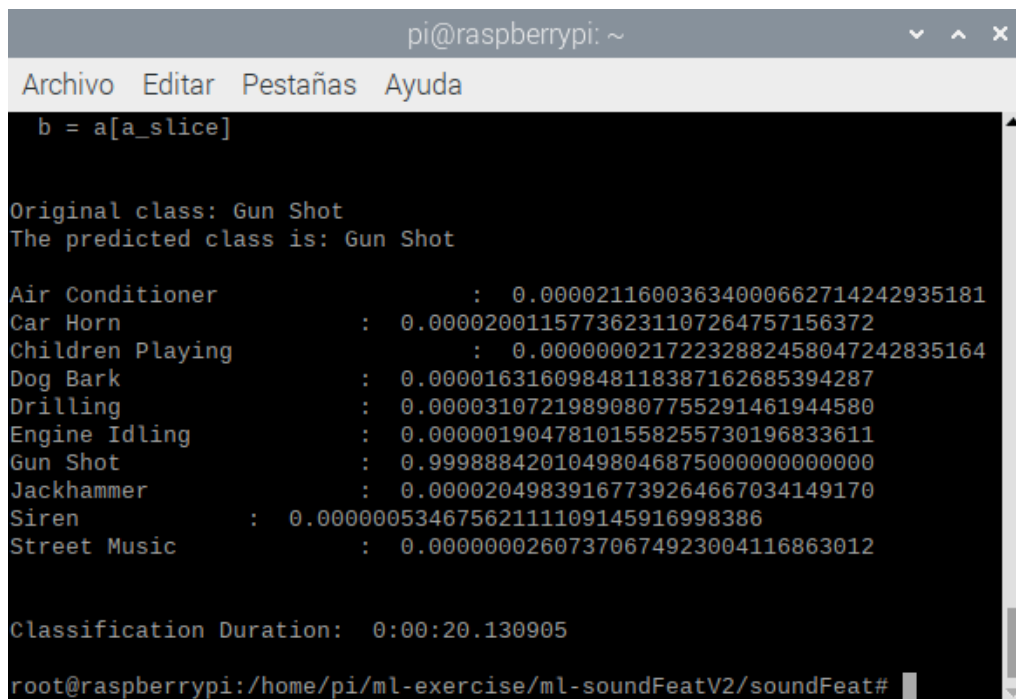
Classification Duration: 0:00:21.279334
root@raspberrypi:/home/pi/ml-exercise/ml-soundFeatV2/soundFeat#

```

Figura 4.51: Raspberry clasificando el evento acústico Sirena

resultados en tiempo real.

Como se puede observar en la Fig. 4.53 el esquema del clasificador queda de la siguiente



```
pi@raspberrypi: ~
Archivo Editar Pestañas Ayuda
b = a[a_slice]

Original class: Gun Shot
The predicted class is: Gun Shot

Air Conditioner           : 0.00002116003634000662714242935181
Car Horn                   : 0.00002001157736231107264757156372
Children Playing          : 0.00000002172232882458047242835164
Dog Bark                   : 0.00001631609848118387162685394287
Drilling                   : 0.00003107219890807755291461944580
Engine Idling             : 0.00000190478101558255730196833611
Gun Shot                   : 0.99988842010498046875000000000000
Jackhammer                 : 0.00002049839167739264667034149170
Siren                      : 0.00000053467562111109145916998386
Street Music               : 0.00000002607370674923004116863012

Classification Duration: 0:00:20.130905
root@raspberrypi:/home/pi/ml-exercise/ml-soundFeatV2/soundFeat#
```

Figura 4.52: Raspberry clasificando el evento acústico Pistola

manera. A través de la interfaz del micrófono y con un script programado en python se captura 4 segundos de un evento acústico, y estos son almacenados como un archivo .wav. Acto seguido este archivo se importa y se procesa por el clasificador para obtener las características y poder clasificarlo en una de las 10 clases posibles. Una vez clasificado se envía a la red la información de clasificación: el porcentaje de clasificación por cada clase y la clase clasificada. De esta forma se ha creado un sensor clasificador de eventos acústicos con una CNN en una Raspberry Pi.

Cabe destacar, que en este proyecto no se ha desplegado el Orion Context-Broker en ninguna red remota como podría ser el cloud o un servidor privado. El proyecto se ha testeado en un red local (LAN) en la que se encontraban conectados directamente los nodos sensores y el PC que actuaba de servidor Fig 4.54. De esta forma no ha hecho falta integrar dispositivos de seguridad o implementar una solución de cloud en la que alojar las funciones del servidor especificado en la sección de FIWARE.

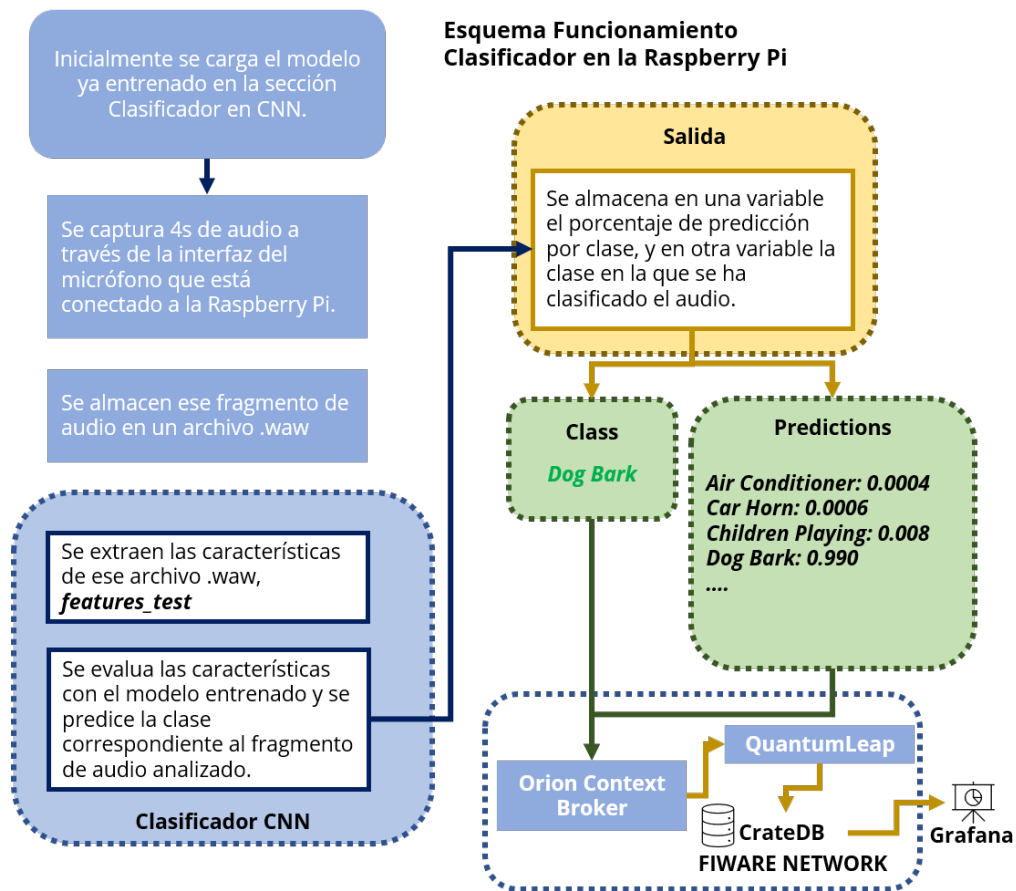


Figura 4.53: Esquema funcionamiento del Clasificador en la Raspberry Pi

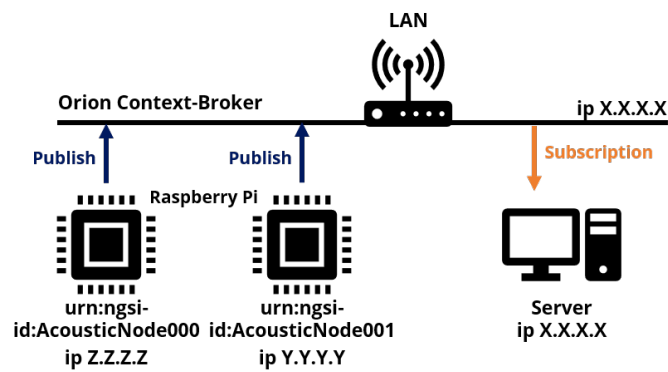


Figura 4.54: Arquitectura de Red implementada en el proyecto

4.5.2. Rendimiento

A modo de comprobación, se ha estudiado el tiempo y coste computacional para la raspberry pi durante el proceso de clasificación de un evento acústico. Es evidente que los dispositivos como la Raspberry Pi presentan limitaciones en términos de computación. Para comprobar las diferencias entre realizar la tarea de clasificación en el PC o en la Raspberry Pi, se han obtenido el coste

Tabla 4.4: Especificaciones de hardware y tiempo total de procesamiento

Device	Processor	RAM	Time
Dell XPS 15 (2015)	I7-6700HQ CPU 2.60 GHz	16GB	0,195 s
Raspberry Pi 3 Model B	Quad Core 1.2 GHz	1GB	2,147 s

Tabla 4.5: Tiempo de procesamiento por tarea específica

Device	wav file Loading	Feature Extraction	CNN Classification
Dell XPS 15 (2015)	0,124 s	0,012 s	0,059 s
Raspberry Pi 3 Model B	2 s	0,052 s	0,095 s

temporal por tarea en todo el proceso de clasificación. Estos resultados nos aportan la idea de poder contar con un sistema en tiempo real o hablar de un sistema que genera información de contexto en forma de históricos.

El tiempo necesario para realizar todo el proceso de grabación y clasificación también se muestra en la Tabla 4.4 para ambos dispositivos. Tenga en cuenta que este tiempo de procesamiento es el momento de grabar el archivo de audio y clasificarlo como una de las 10 clases de urbansound. En el tiempo de procesamiento que se muestra en la Tabla 4.5, no se incluyen los 3,089 necesarios para grabar el sonido y guardarlo en .wav, ya que es el mismo tiempo para ambos dispositivos. Por otro lado, la Tabla 4.5 muestra los tiempos de procesamiento dedicados a cada tarea específica de la etapa de clasificación, incluida la carga del segmento de audio del archivo wav. La tarea de extracción de características tarda cuatro veces más en el Raspberry Pi que en el PC, mientras que la tarea de clasificación tarda casi el doble.

Dados los tiempos de procesamiento de la Tabla 4.5, nuestra red es capaz de proporcionar una clasificación de eventos sólida aproximadamente cada 5 segundos para un tiempo de observación de 3 segundos, lo que puede considerarse procesamiento en tiempo real para numerosas aplicaciones de smart city. Sin embargo, el principal inconveniente de nuestra red acústica de bajo costo es el gran consumo de energía debido a su actividad constante, lo que significa que la red solo podría implementarse en sitios con acceso a red eléctrica, ya que el clasificador consume casi un 90 % de los recursos, y con esta propuesta debería estar activo contantemente para clasificar cualquier evento acústico detectado. Esta situación se puede mejorar integrando un detector previamente a la clasificación de forma que sólo se clasificarían aquellos eventos que superen un umbral establecido por el detector, y de esta forma no tendrá que estar clasificando cada 5.

Ambos estudios se han realizado modificando los scripts de cada función para obtener información de paso. En el caso de el coste temporal, se ha creado una etiqueta temporal empleando la librería de *datetime*. Se ha añadido una etiqueta temporal al principio y final de cada bloque funcional, correspondiendo a las 3 funciones que se describen en la Tabla 4.5. Con estas etiquetas se obtuvo una media de 15 muestras, a través de las cuales se obtuvieron los datos que se muestran

Tabla 4.6: Consumo de Recursos del clasificador en la Raspberry Pi

	<i>Clasificador</i>
CPU	97,4 %
RAM	29,84 %

en la tabla. Con esto se observa que implementando todo el proceso de capturar un fragmento de audio y clasificarlo, es una tarea que se puede realizar en un dispositivo de bajo coste (Raspberry Pi) en tiempo real. Con un coste del sensor total de unos 10 euros el sensor, teniendo en cuenta el coste del micrófono de unos 70 euros y la Raspberry Pi 3 Model B unos 30 euros. Es un resultado muy bueno teniendo en cuenta las soluciones actuales de sensores inteligentes que sobrepasan los 200 euros, como es el caso del proyecto SONYC.

Capítulo 5

Conclusiones

5.1. Conclusiones

Se ha presentado aquí la red cuyos nodos están formados por Raspberry Pi que son capaces de llevar a cabo todo el reconocimiento de eventos sonoros urbanos, desde la extracción de características hasta la clasificación sonora mediante una CNN profunda. También se ha descrito la estructura de la red que se basa en el estándar abierto FIWARE, por lo que todo el sistema se puede replicar sin necesidad de software propietario o hardware específico.

Aunque el rendimiento es bueno en términos de precisión y la red puede obtener una salida cada 5 segundos, nuestro problema es la alta carga de cálculo requerida por el algoritmo de aprendizaje profundo. Un uso constante de la etapa de clasificación significa que la carga de la CPU alcanza un promedio de 97 %, lo que a su vez significa un gran consumo de energía de la batería. Por este motivo, esta propuesta requiere de conectarse a la red eléctrica para un correcto funcionamiento. Otro aspecto a considerar de el uso de la Raspberry Pi como sensor de nuestra red, es que es de compleja implementación a la hora de contar con librerías de alto nivel que operan sobre python como es el caso de Librosa, Tensorflow y Keras. Por lo que a medida que se vayan modificando estas librerías y sus funcionalidades esta solución puede quedar obsoleta. El caso ideal sería contar con un dispositivo hecho a medida o cuya compatibilidad se asegure con el software implementado. La realización de este proyecto ha sido mera comprobación de la capacidad de un dispositivo de bajo coste, y la posible implementación de redes neuronales capaces de clasificar eventos sobre estos mismos dispositivos.

Con la realización de este proyecto, se ha expuesto el comportamiento de una red de sensores de bajo coste, capaz de clasificar sonidos urbanos de forma autónoma por redes CNN. Además se ha demostrado las ventajas de poder implementar este tipo de sistema en una arquitectura de bajo coste y sin necesidad de implementar soluciones de cloud y sensores de gran coste. Esta técnica ofrece la posibilidad de poder monitorizar eventos acústicos urbanos en tiempo real y crear un contexto por medio de datos procesados de forma autónoma por clasificadores basados en redes neuronales convolucionales CNN. De esta manera se plantea una solución viable y de bajo coste de monitorización en la que se procesa información en los nodos sin necesidad de sobrecargar la red con datos sin procesar, sino con información procesada en el nodo.

5.2. Líneas futuras

Teniendo en cuenta los resultados obtenidos y los estudios realizados en esta línea de trabajo, este proyecto se puede mejorar y obtener un mayor rendimiento. En cuanto al clasificador, se puede implementar una arquitectura CNN-VGG con objeto de obtener un mayor porcentaje de precisión. A espensas de verificar que este tipo de redes pueden implementarse en un dispositivo de bajo coste como la Raspberry Pi.

En cuanto a la decisión de usar la Raspberry Pi como sensor clasificador, tiene la ventaja de ser un sensor de muy buen rendimiento y bajo coste. El problema que presenta este tipo de sensores es la escalabilidad y la implementación de entornos de desarrollo de alto nivel. Como es el caso del uso de las librerías de sonido Librosa y las de la CNN que son Tensorflow y Keras. Este tipo de librerías cuentan con compatibilidad en **Armv16**, **Armv17**, la cuestión es que su implementación e instalación no es tan estandar como se requeriría para un despliegue masivo de sensores. En esta línea, la confección de un dispositivo que se adapte a las necesidades pasando los scripts actuales a C y modificando a bajo nivel la confección del proyecto podría ser una solución más potente, versátil y de menor coste que la presentada. Por otra parte, se puede emplear Docker para el despliegue de servicios automáticos como solución a la poca estandarización a la hora de poner en marcha todo el entorno de desarrollo necesario para el funcionamiento del sensor. En esta solución sigue existiendo la limitación de compatibilidad de sistemas **armv1** con ciertas librerías.

Otra mejora a tener en cuenta, es el despliegue de este proyecto sobre una red WASN, y no en una LAN. En el que los servicios de red estén alojados en un servidor externo o en el cloud, y los sensores de la red accedan de forma inalámbrica y remota a estos servicios. Creando así una solución WASN de sensores inteligentes con edge computing. Ya que en este proyecto sólo se ha comprobado el correcto funcionamiento de la red en un entorno local (LAN) para comprobar el correcto funcionamiento de cada componente y su rendimiento durante el proceso del mismo.

Capítulo 6

Anexo I

6.1. Extracción de Características por Clase

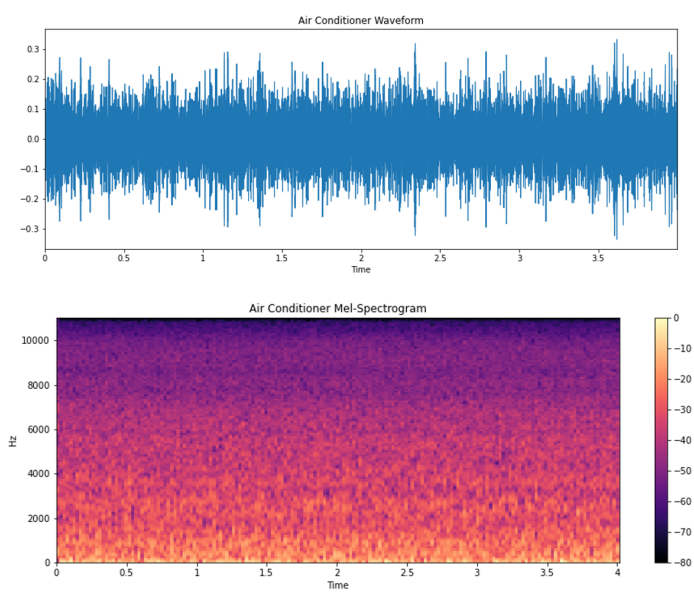


Figura 6.1: Caracterización acústica del aire acondicionado

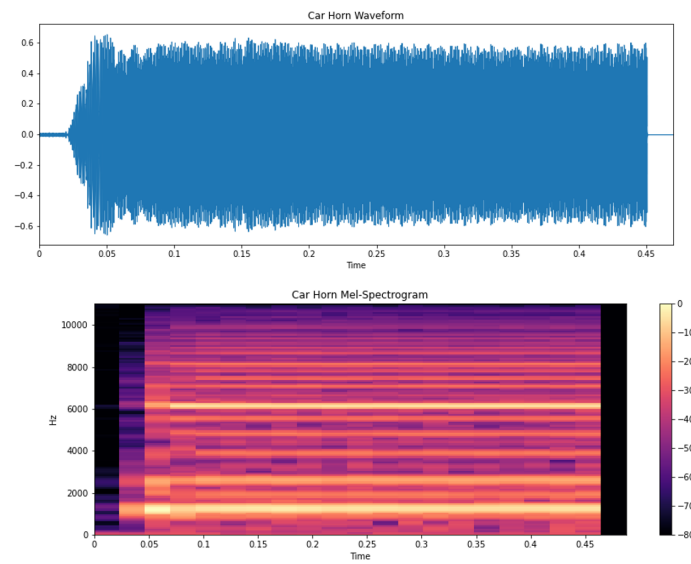


Figura 6.2: Caracterización acústica del cláxon

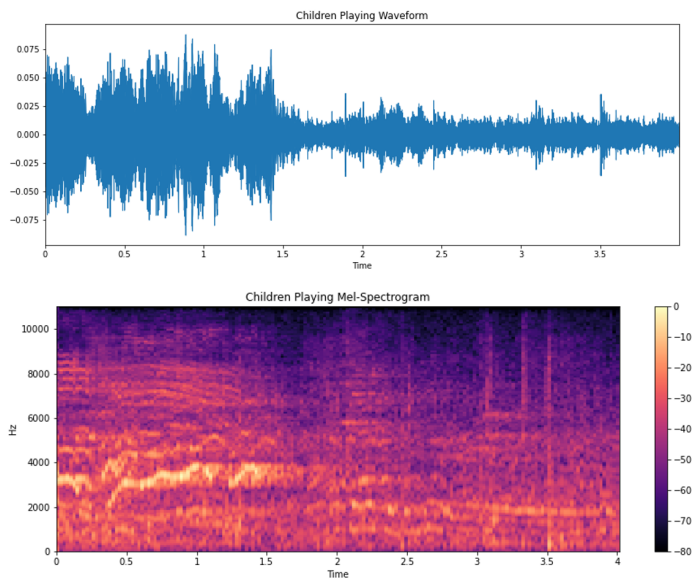


Figura 6.3: Caracterización acústica de niños jugando

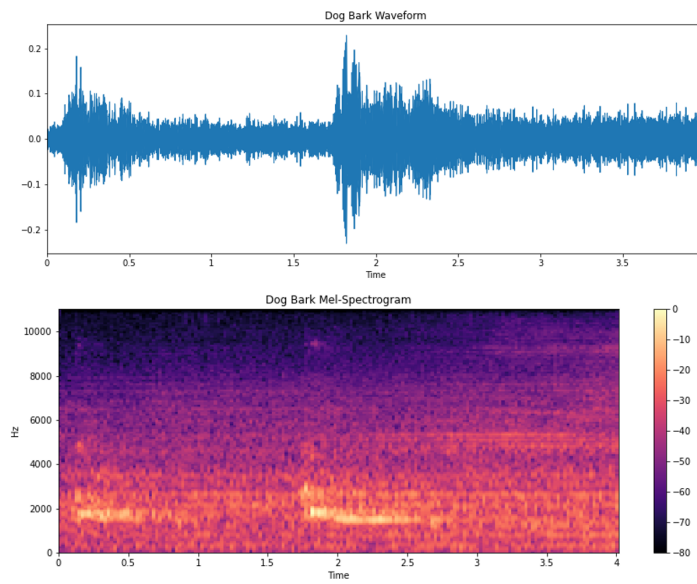


Figura 6.4: Caracterización acústica de perros ladrando

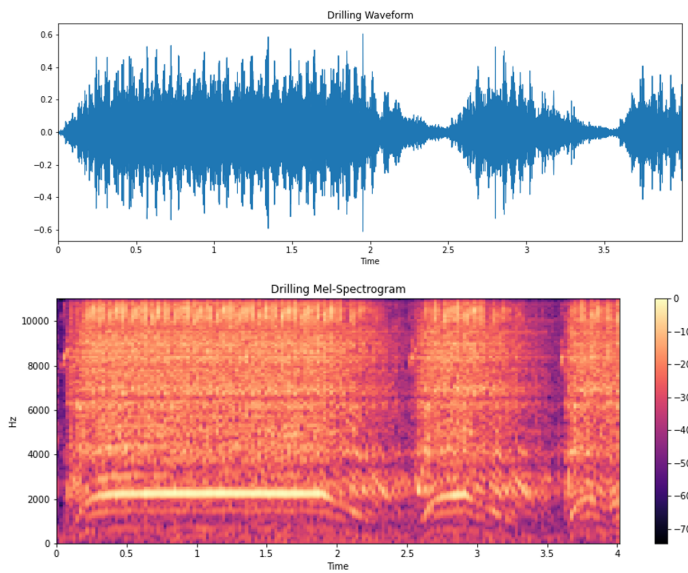


Figura 6.5: Caracterización acústica del ruido por Perforación

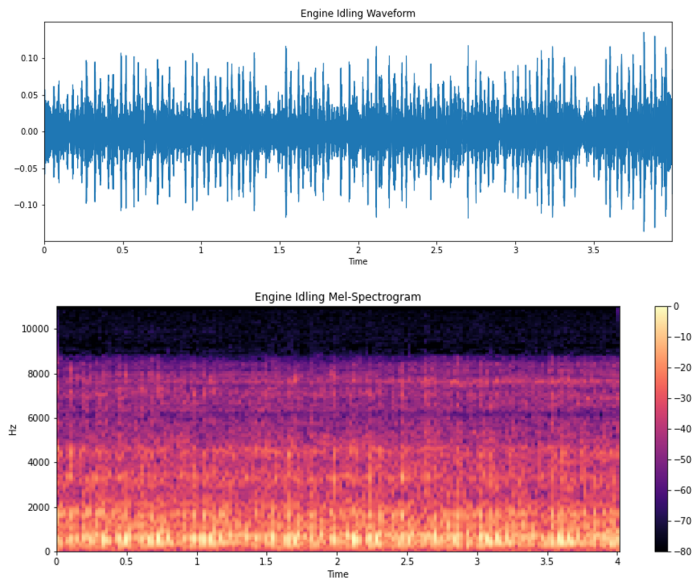


Figura 6.6: Caracterización acústica del ruido de motor

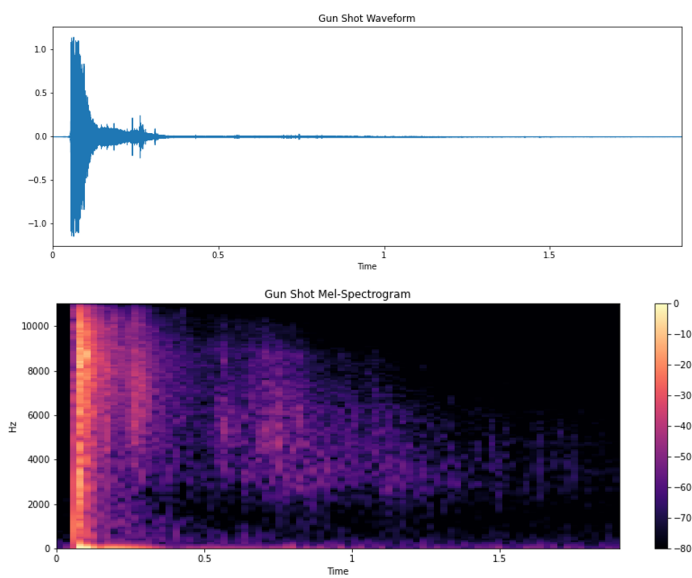


Figura 6.7: Caracterización acústica de los disparos

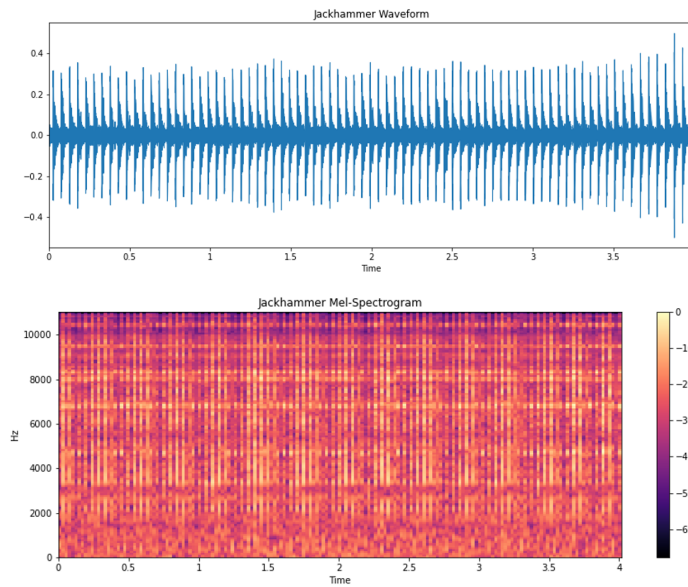


Figura 6.8: Caracterización acústica del martillo neumático

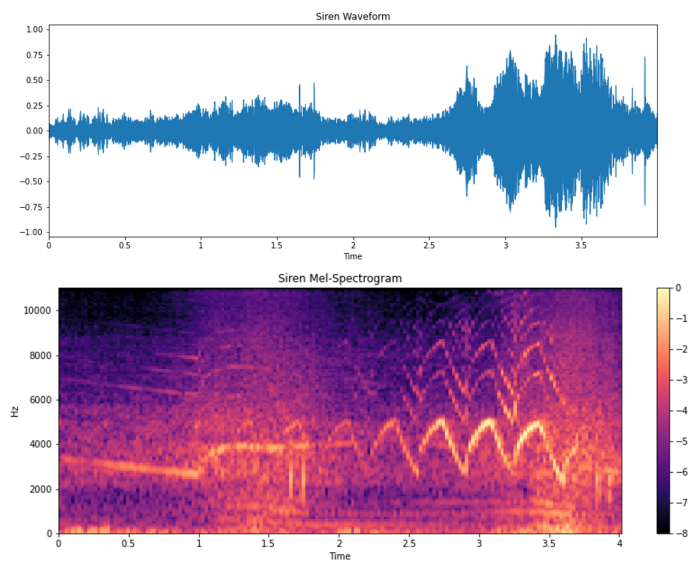


Figura 6.9: Caracterización acústica de la Sirena

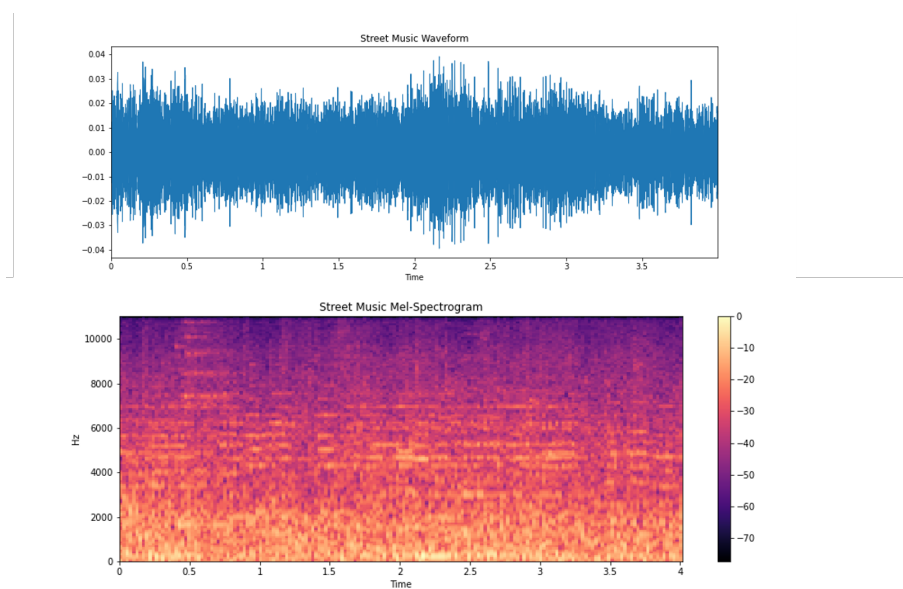


Figura 6.10: Caracterización acústica de la música callejera

Capítulo 7

Anexo II

7.1. Código fuente Github

Todo el código fuente de este proyecto está disponible en un repositorio de GitHub que es público. En él se ha dejado adjunto un Readm.md en el que se expone a modo de tutorial el funcionamiento y proceso para poder implementar todo el proyecto y probar en casa con los mismos componentes que se han utilizado en este proyecto. En el tutorial se explica el contenido de todos los scripts y los pasos que se han realizado tanto en el PC como en la Raspberry Pi para poder hacer uso de ellos. De esta forma cualquier persona puede implementar y testear el funcionamiento de este proyecto, como base y posible referencia a investigaciones futuras.

El repositorio se puede encontrar en el siguiente enlace:

<https://github.com/dSalvoG/ML-Urban-Sound-Classifier>.

sound_class_raspberry.py	full project dir	41 seconds ago
sound_classifier.py	getting ready rasp script	6 months ago
sound_featuring.py	updating Quantum Architecture	2 months ago
usound_classifier_part1.ipynb	updated QuantumLeap	2 months ago
usound_classifier_part2.ipynb	full project dir	41 seconds ago

Readme.md ✎

🔗 Urban Sound Classification with CNN on Raspberry Pi Model 3 and Windows 10

Author: David Salvo Gutiérrez
 Work Group: GTAC-ITEAM (Universitat Politècnica de València)

Environment Project Set Up

Windows 10 installations

We will train and test our model on the PC first. We used windows 10 for Dell XPS 15 Lapto, which we used to train and test de classification model. To set-up the windows 10 lapto we must install Python 3.6 version environment and its libraries, for this project. To make things simple, we used Anaconda as a Python Framework to work with this environment.

1. Install Anaconda 3.8 (<https://www.anaconda.com/products/individual>)
2. Create a virtual environment into Anaconda Prompt for Tensorflow.
3. (open anaconda prompt, user example)
 - i. C:\Users\sqlvladi> conda create -n tensorflow pip python=3.8

Figura 7.1: Muestra del repositorio en el que se ha alojado el proyecto completo.

```

## Making a POST request to Orion Broker
import requests

# defining the api-endpoint
API_ENDPOINT = "http://localhost:1026/v2/entities/"

# headers
headers_string={
    'Content-Type': 'application/json',
    'fiware-service': 'openiot',
    'fiware-servicepath': '/'
}

# data to be sent to api
payload = json.dumps(data)

# request post (using json payload)
x = requests.post(url= API_ENDPOINT, data= payload, headers= headers_string,)
```

Figura 7.2: Fragmento de código subido al repositorio

Bibliografía

- [1] *2019 Revision of World Population Prospects*. URL: <https://population.un.org/wpp/>.
- [2] Salvo Gutiérrez, D. *Smart Cityzen. September 2018*. URL: <http://hdl.handle.net/10251/109782>.
- [3] Justin Salamon, Christopher Jacoby y Juan Pablo Bello. “A Dataset and Taxonomy for Urban Sound Research”. En: *Proc. ACM Internat. Conf. on Multimedia - MM '14*. 2014, págs. 1041-1044.
- [4] Justin Salamon y Juan Pablo Bello. “Deep Convolutional Neural Networks and Data Augmentation for Environmental Sound Classification”. En: *IEEE Signal Processing Letters* 24.3 (2017), págs. 279-283.
- [5] Sachin Chachada y C. C. Jay Kuo. “Environmental sound recognition: A survey”. En: *AP-SIPA Transactions on Signal and Information Processing* 3.2014 (2014).
- [6] Karol J Piczak. “ESC: Dataset for Environmental Sound Classification”. En: *Proceedings of the 23rd ACM international conference on Multimedia - MM '15*. 2015, págs. 1015-1018.
- [7] Dan Stowell y col. “Detection and Classification of Acoustic Scenes and Events”. En: *IEEE Transactions on Multimedia* 17.10 (2015), págs. 1733-1746.
- [8] Cagdas Bilen y col. “A Framework for the Robust Evaluation of Sound Event Detection”. En: *ICASSP 2020*. 2020.
- [9] Francesc Alías y Rosa Ma Alsina-Pagès. “Review of Wireless Acoustic Sensor Networks for Environmental Noise Monitoring in Smart Cities”. En: *Journal of Sensors* 2019 (2019), págs. 1-13.
- [10] M. Shamim Hossain y Ghulam Muhammad. “Environment Classification for Urban Big Data Using Deep Learning”. En: *IEEE Communications Magazine* 56.11 (2018), págs. 44-50.
- [11] Zhengguo Sheng y col. “Wireless acoustic sensor networks and edge computing for rapid acoustic monitoring”. En: *IEEE/CAA Journal of Automatica Sinica* 6.1 (2019), págs. 64-74.
- [12] Liyan Luo y col. “Wireless Sensor Networks for Noise Measurement and Acoustic Event Recognitions in Urban Environments”. En: *Sensors* 20.7 (2020), pág. 2093.
- [13] Charlie Mydlarz y col. “The Life of a New York City Noise Sensor Network”. En: *Sensors* 19.6 (2019), pág. 1415.
- [14] Sebastián García y col. “Heterogeneous LoRa-Based Wireless Multimedia Sensor Network Multiprocessor Platform for Environmental Monitoring”. En: *Sensors* 19.16 (2019), pág. 3446.

- [15] Jaume Segura-Garcia y col. “Low-Cost Alternatives for Urban Noise Nuisance Monitoring Using Wireless Sensor Networks”. En: *IEEE Sensors Journal* 15.2 (2015), págs. 836-844.
- [16] Annamaria Mesaros y col. “Detection and Classification of Acoustic Scenes and Events: Outcome of the DCASE 2016 Challenge”. En: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 26.2 (2018), págs. 379-393.
- [17] Yu Su y col. “Environment Sound Classification Using a Two-Stream CNN Based on Decision-Level Fusion”. En: *Sensors* 19.7 (2019), pág. 1733.
- [18] *fiware.org*, accessed on June 2020. URL: <https://www.fiware.org/developers/>.
- [19] *CS231n: Convolutional Neural Networks for Visual Recognition*. URL: <https://cs231n.github.io/convolutional-networks/>.
- [20] *Applied Deep Learning - Part 4: Convolutional Neural Networks*, Arden Dertat, Nov 2017. URL: <https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2>.
- [21] Karol J. Piczak. “Environmental sound classification with convolutional neural networks”. En: *2015 IEEE 25th International Workshop on Machine Learning for Signal Processing (MLSP)*. 2015, págs. 1-6.