
MÁSTER EN INGENIERÍA INDUSTRIAL

TRABAJO FIN DE GRADO

# Mejora de sistemas de alarma y video vigilancia mediante la incorporación de análisis de audio

Autor: Alberto Menéndez Ruiz de Azúa

Directores: Álvaro López López y Lucía Güitta López

Madrid

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

"Mejora de sistemas de alarma y video vigilancia mediante la incorporación de análisis de audio"

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el

curso académico 2020/21 es de mi autoría, original e inédito y

no ha sido presentado con anterioridad a otros efectos.

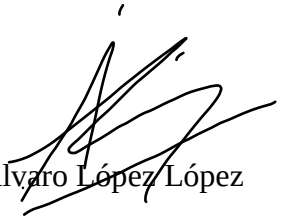
El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido

tomada de otros documentos está debidamente referenciada.

Fdo.: Alberto Menéndez Ruiz de Azúa Fecha: 21 / 07 / 2021

Autorizada la entrega del proyecto

LOS DIRECTORES DEL PROYECTO



Fdo.: Álvaro López López Fecha: 20 / 07 / 2021

Fdo.: Lucía Güitta López Fecha: 21 / 07 / 2021


MÁSTER EN INGENIERÍA INDUSTRIAL

TRABAJO FIN DE GRADO

# Mejora de sistemas de alarma y video vigilancia mediante la incorporación de análisis de audio

Autor: Alberto Menéndez Ruiz de Azúa

Directores: Álvaro López López y Lucía Güitta López

Madrid

# Agradecimientos

Me gustaría dar mis agradecimientos en primer lugar a Álvaro López y Lucia Güitta por darme la oportunidad de llevar a cabo este proyecto y ser mis mentores durante la consecución del mismo; invitándome cada día a ser mejor ingeniero, ayudándome a desarrollar al máximo mis capacidades y poniendo todas las facilidades para que pudiera dar el máximo de mí mismo y llevar el proyecto a buen puerto.

En segundo lugar, me gustaría agradecer a mis padres, Álvaro y Teresa, y a mis hermanos, Pablo e Isabel, que son los que me han tenido que aguantar los días que estuve de peor humor por no conseguir avanzar el proyecto de la manera que me gustaría y también por darme su apoyo incondicional y siempre creer en mí.

En tercer lugar, al resto de mi familia, tíos, primos y abuelos, que a pesar de ser un año complicado por la pandemia del Covid-19 y no habernos podido ver tan de costumbre como otros años los llevo siempre en mi pensamiento y me he acordado mucho de todos en el día a día.

Por último, y no menos importantes, a todas aquellas personas que hacen que mis momentos de desconexión del trabajo y de este proyecto en particular sean eso, de desconexión, de pasárselo bien y de coger fuerzas para que todo lo demás vaya mejor. Mi novia, Raquel, por estar ahí siempre y aguantarme a pesar de mis mil defectos. Mis “chavales” del Colegio San José del Parque, por saber hacerme desconectar como nadie con nuestras historias y anécdotas. Los compañeros de Rue Adolphe 39 en Luxemburgo, por hacer la vuelta a casa mucho más amena y por conseguir que todos los días haya risas y buen humor garantizados.

A todos ellos y a aquellos que en mayor o menor medida hayan contribuido a la consecución del proyecto o de mi mejor productividad para llevarlo a cabo:

GRACIAS

# MEJORA DE SISTEMAS DE ALARMA Y VIDEO VIGILANCIA MEDIANTE LA INCORPORACIÓN DE ANÁLISIS DE AUDIO

**Autor: Menéndez Ruiz de Azúa, Alberto.**
Director: López López, Álvaro; Güitta López, Lucia
Entidad Colaboradora: ICAI – Universidad Pontificia Comillas

## RESUMEN DEL PROYECTO

El proyecto ha logrado crear, entrenar e implementar diversos modelos de inteligencia artificial para la detección y clasificación de audio en el contexto de la seguridad y la video vigilancia. Se han logrado resultados al nivel del estado del arte y se ha hecho un estudio comparando diversas arquitecturas y tipos de modelos para seleccionar aquellas que se ajustaran mejor a la problemática concreta del proyecto.

**Palabras clave**: Espectrograma, Inteligencia artificial, Redes Neuronales, Clasificación de audio, Seguridad

### 1. Introducción

Este proyecto surge como respuesta a una limitación en la industria de la seguridad, más concretamente al sector de las alarmas y la video vigilancia. La exigencia de una solución viene fundamentada por la necesidad de ofrecer un mejor servicio a los clientes de este sector y aumentar la fiabilidad y eficacia de los sistemas que existen actualmente.

La idea central del proyecto, y a la que se ha dado respuesta, surge de un análisis exhaustivo de los expertos del sector de la seguridad sobre los casos de fallo de sus sistemas y las limitaciones con las que se suelen encontrar. Tras un estudio de los casos en los que estos sistemas erran o no se alcanza la eficiencia adecuada, se determinó que una fracción de los robos o incidencias que fueron omitidos por la vídeo vigilancia se podrían haber detectado y, en consecuencia, haber sido evitados, si simplemente se hubiera llevado a cabo un análisis del audio en el momento concreto del incidente. Este análisis sería complementario al vídeo de las cámaras de seguridad y en ningún caso sustitutivo del mismo.

Por tanto, la finalidad principal de este proyecto es lograr satisfacer esta necesidad y elaborar un método de clasificación de audio, que funcione de forma complementaria a las actuales cámaras de seguridad, dotándolas de una capacidad de la que hasta ahora carecían y que les permita mejorar su eficacia y contribuir a la seguridad de los usuarios finales del producto.

Finalmente, se ha optado por no limitar el proyecto al ámbito exclusivo de la seguridad y aprovechar las enormes posibilidades que la clasificación de audio aporta en temas de confort para el usuario final del sistema de seguridad. Estas posibilidades vienen limitadas únicamente por las clases en las que se elija clasificar el audio, ya que hay clases como el lloro de un niño o el chisporroteo de fuego que pueden alertar al usuario de incidencias que no implican necesariamente un robo, pero hacen el sistema más completo y atractivo para un consumidor final.

En el proyecto se busca dotar de valor añadido a la tecnología actual en dos aspectos:

- Mediante la mejora de la eficiencia en seguridad de los sistemas.
- Otorgándole capacidades de confort para el usuario final.

El objetivo global del proyecto se determinó como: el desarrollo e implementación de técnicas de vanguardia en materia de clasificación de audio, orientadas a la detección de incidencias de seguridad y comodidad de usuarios.

## 2. Definición del Proyecto

El primer paso en la conclusión de los objetivos del proyecto fue la investigación del estado del arte en esta materia, tanto de técnicas pioneras en materia de seguridad como en tecnologías de tratamiento y clasificación de audio.

En este análisis se detectaron varias ideas interesantes. En primer lugar, que las técnicas más pioneras del estado del arte en materia de clasificación de audio pasaban en su mayoría por tecnologías de *machine learning* y más concretamente por el campo del *deep learning* y las redes neuronales. [1] y [2] representan el estado el arte en materia de clasificación de audio, logrando los mejores KPIs en los *datasets* de GITZAN, ESC50 y UrbanSound8K. Una vez dentro de este campo, el segundo aspecto importante es la escasez de datos en el contexto de la seguridad, ya que es un campo sensible y los datos al respecto están muy restringidos. Esto dificulta en gran medida el entrenamiento e implementación de técnicas de *machine learning.* Por último, el tercer punto importante que se detectó en el estudio del estado del arte fue el hecho de que trabajar con señales de audio es complicado y es por eso que la mayor parte de las soluciones del estado del arte se basan en la transformación de las ondas de audio en imágenes y en la posterior aplicación de técnicas del estado del arte de clasificación de imágenes, como los modelos *inception, vgg19...* Esta idea representa el

punto central de los dos métodos del estado del arte mencionados anteriormente y es la que se empleó a lo largo de este proyecto

Para poder emplear estas técnicas el primer paso es el de convertir el audio en imágenes que puedan ser fácilmente clasificadas por los modelos. Esto se logra en este proyecto mediante una conversión de ondas sonoras conocida como *Espectrograma de Mel* [3] y que emplea en la conversión del audio ideas como el uso de escalas logarítmicas, que simulan la forma en la que el oído humano se enfrenta al tratamiento de esta información. En la Figura 1 se pueden observar ejemplos de los espectrogramas de Mel generados de 4 audios utilizados a lo largo del proyecto para el entrenamiento y validación de los modelos.

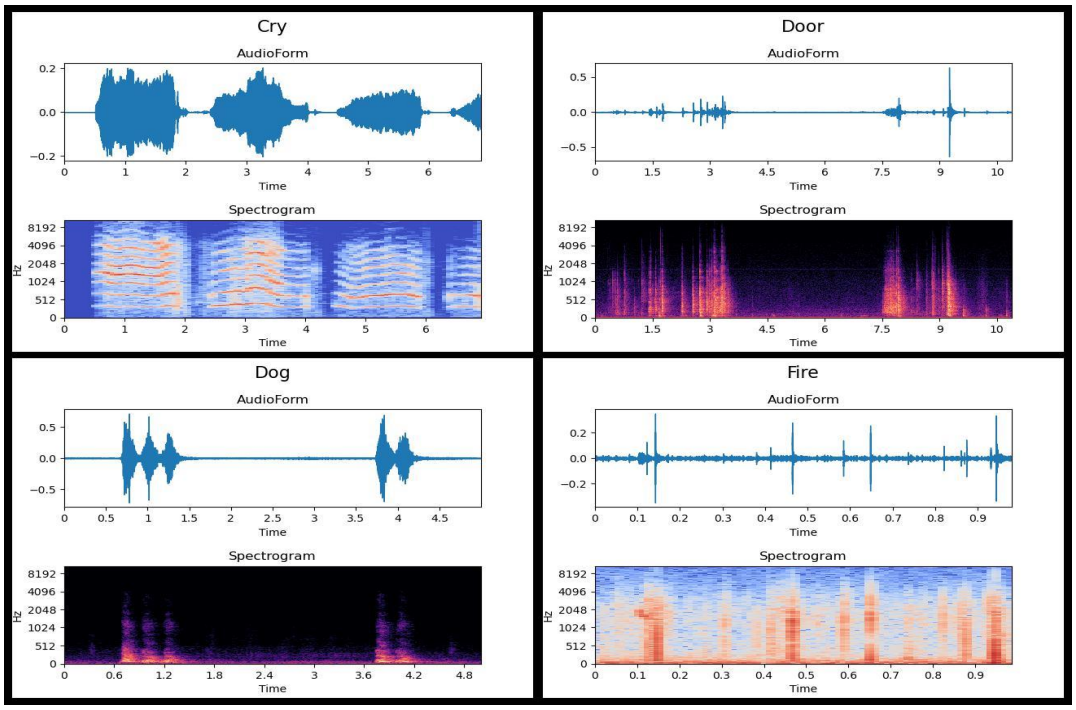



*Figura 1: Ejemplo del Espectrograma de Mel de 4 de los audios empleados en el proyecto (Elaboración propia)*

Una vez analizado el estado del arte se determinó el uso de técnicas de inteligencia artificial y se pasó a la parte central del proyecto, donde se procedió a desarrollar y entrenar diversas arquitecturas de modelos. Este proceso constó de tres etapas:

1. Recopilación de datos de audio para el entrenamiento de los modelos.

2. Análisis, pre procesamiento y limpieza de los datos para la optimización del entrenamiento.
3. Entrenamiento y análisis de los KPIs de las distintas arquitecturas.
4. Comparación de los resultados de los distintos modelos para determinar la mejor solución de cara a nuestro proyecto.

Los pasos 3 y 4 se repitieron en bucles y cambiando arquitecturas y detalles hasta que se obtuvieron los resultados deseados para algunos de los modelos y se pudo realizar una comparación adecuada. El proceso de mejora se basó en el análisis de las matrices de confusión del modelo y las gráficas de entrenamiento, seguido de un estudio de las clases más problemáticas en la matriz para de esta forma poder llevar a cabo las acciones pertinentes orientadas a mejorar los KPIs: limpieza de los datos, modificación de hiper parámetros, técnicas de entrenamiento…

Tras estas etapas y una vez seleccionado el modelo que mejor se ajustaba a los requerimientos del proyecto, se procedió a su implementación y puesta en producción. Las clases a emplear se definieron en la etapa de requerimientos y de forma previa al entrenamiento de las arquitecturas, así como los requerimientos mínimos de los modelos para ser puestos en producción. Se vera más adelante como estas clases cambiaron a lo largo del proyecto para hacer frente a ciertas dificultades y refinar el resultado final.

El despliegue del modelo se hizo en primera instancia mediante una App web que permitía subir audios y obtener un resultado de estos en tiempo real, para que el cliente pudiera hacer pruebas y hacerse una idea de la posterior implementación en su plataforma. En el Código 1 se presenta un ejemplo de la salida de la aplicación con los resultados del modelo y el tiempo de procesamiento para un audio de entrada, de forma que se tuviera no solo una percepción de la precisión, sino también del tiempo de inferencia, que resulta clave en las aplicaciones finales del modelo. La entrega final se hizo mediante un paquete de *Python* con una carpeta para poner los audios a analizar (input) y otra carpeta en la que se almacenan los resultados de analizar estos audios (output).

## 3. Descripción del modelo/sistema/herramienta

### 3.1. Arquitecturas

Una vez decidido el método general a seguir y emplear para el proyecto (redes neuronales convoluciones para clasificar Espectrogramas de Mel de los audios), se planteó el dilema de cuál de todas las arquitecturas disponibles para este tipo de técnica daría mejores resultados a la hora de implementar el proyecto [4]. Es por ello que en el proyecto se llevó a cabo un análisis de dos arquitecturas distintas para la clasificación de audio, así como de sendas variantes de estas arquitecturas, de manera que se pudiera determinar cuál de las dos daría mejores resultados para el proyecto. Las dos arquitecturas comparadas y estudiadas fueron:

- **Red convolucional completa (2DCNN)**: esta arquitectura consta de una red convolucional construida y entrenada en su totalidad y sin emplear ningún tipo de *transfer learning.* Este tipo de red permitió un control absoluto sobre la totalidad de los parámetros del modelo permitiendo una gran flexibilidad y posibilidades a la hora de experimentar y probar diferentes configuraciones. Un ejemplo de la arquitectura general de este tipo de soluciones se puede ver en la Figura 2.

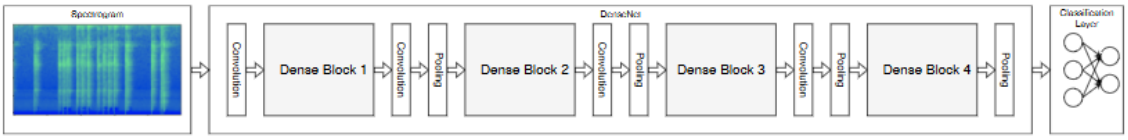



*Figura 2: Arquitectura general de red convolucional para clasificación de audio (Fuente: [1])*

- ***Encoder* más clasificador (E + C)**: esta arquitectura cuenta con un *encoder* pre entrenado en un gran *dataset* de audios seguido de un clasificador propio para llevar a cabo la clasificación de los distintos outputs del *encoder*. En concreto el *encoder* utilizado se trata de *Vggish* de *Google*, el cual ha sido pre entrenado en el dataset de audios *Youtube 8M*. A diferencia de la primera opción, solo se tuvo control absoluto sobre la arquitectura del clasificador, pero se cuenta con la ventaja de usar un modelo cuyo entrenamiento sería imposible con los recursos disponibles. Además, se permiten técnicas de *fine tuning* sobre este modelo. En este caso la estructura de la arquitectura también es similar a la de la Figura 2, solo que las etapas convolucionales se corresponden con una arquitectura más profunda y pre entrenada.

Esta arquitectura está basada en la red neuronal Vgg-19 para clasificación de imágenes, la variante para audio es conocida como Vggish [5].

En la elección del modelo definitivo no se emplearon únicamente los resultados de los KPIs de entrenamiento de las arquitecturas, si no que se tuvieron en cuenta otros factores como: escalabilidad del modelo, flexibilidad a la hora de aumentar y cambiar el número de las clases, tiempo de inferencia o tiempo de reentrenamiento en caso de querer añadir nuevas clases o modificar las existentes.

### 3.2. Datos y pre procesamiento

Antes de entrar en cualquiera de los dos modelos, los audios son sometidos a un proceso previo en el que son convertidos al formato óptimo para su entrada en las arquitecturas. Este proceso consta de las siguientes etapas:

1. Transformación del *input* de audio de formato .wav o .mp3 a su correspondiente espectrograma de Mel con los siguientes parámetros para la transformación:
    - Frecuencia de muestreo: 44100 Hz
    - Duración del audio: 960 ms (fijado por los tamaños por la arquitectura pre entrenada de Vggish)
    - Duración de la ventana de la transformada de Fourier: 25 ms
    - Salto de la ventana móvil para la transformada de Fourier: 10 ms
    - Numero de *mels* en los que separar el espacio de la frecuencia: 64

Los resultados de convertir una onda de audio en espectrograma logarítmico de Mel se pueden ver en la Figura 3.

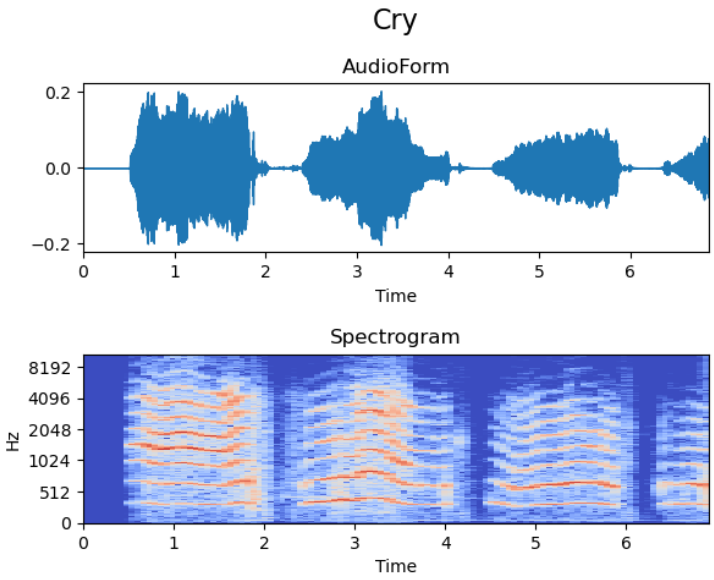



*Figura 3: Ejemplo de onda del audio del lloro de un bebe convertido a espectrograma logarítmico de Mel (elaboración propia)*

2. Segmentación de estos espectrogramas en imágenes del formato adecuado para cada una de las arquitecturas (96 x 64 pixeles). Esto se traduce en fragmentos de audio de 960 ms, ya que el salto de la ventana móvil usada para el Espectrograma de Mel es de 10ms (960 ms / 10 ms/pixel = 96 pixeles). Estos fragmentos se tomaron con un desplazamiento de 480 ms para ir superponiendo diferentes tramos y evitar dejar zonas que permitieran reconocer un audio cortadas y difíciles de clasificar para los modelos.

Otro de los aspectos que se llevaron a cabo fue una limpieza general de los datos antes de cualquier tipo de entrenamiento, ya que en algunas de las primeras pruebas que se llevaron a cabo, se detectó un error que estaban teniendo ambos modelos y que venía derivado de un incorrecto etiquetado de los datos. Este error se debió al hecho de utilizar ventanas móviles de 960 ms como *input* para los modelos y al hecho de que el sonido característico y que daba nombre a la etiqueta del audio no se producía en todas estas ventanas. Esto hacía que podía hubiera algunas de estas ventanas que simplemente fueran un silencio o un simple ruido de fondo que no se correspondía con la clase con la que habían sido etiquetadas. Un ejemplo de esto se muestra en la Figura 4.

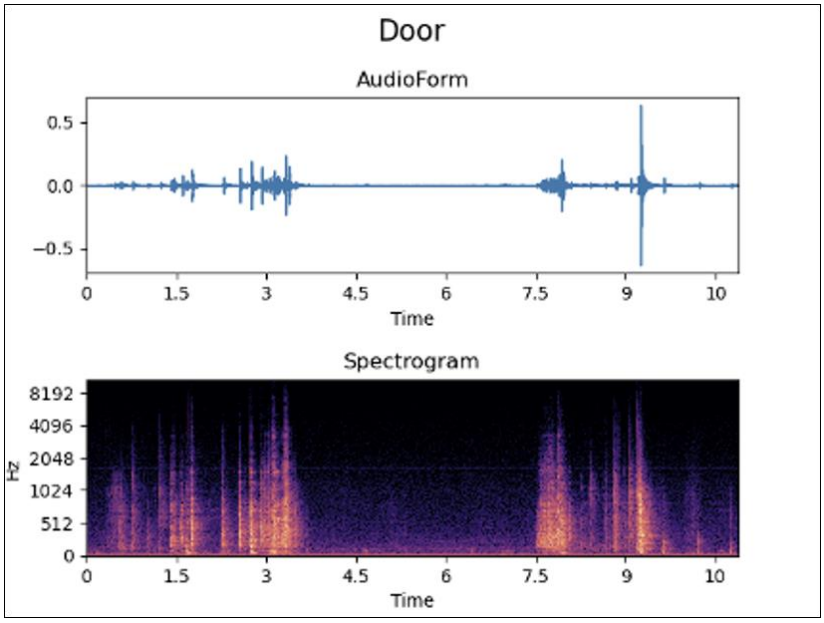



*Figura 4: Espectrograma de audio de una puerta en el que se visualiza la falta de información para cualquier muestra que cuyos 960 ms abarquen el rango desde 3,5 a 7,5 segundos (elaboración propia)*

Para solucionar este problema se aprovechó el codificador de la segunda arquitectura para generar los vectores característicos de cada una de las ventanas de 960 ms. Con estos vectores de dimensión 128 se emplearon técnicas de aprendizaje no supervisado. Para ello se aprovechó que se tenían ejemplos de vectores que se sabe con certeza que pertenecen a la clase buscada y se compararon con una métrica euclídea (norma) los candidatos a esa clase. En la Figura 5 se puede observar como hay una cierta cantidad de ejemplos en la clase de presencia (pasos y puertas) que están muy cercanos al centroide artificial de la clase Silencio, generada para poder hacer esta limpieza. Esto permitió eliminar silencios y ruidos de fondo mal etiquetados en el set de entrenamiento. Esta técnica se usó para limpiar clases tanto de posibles muestras de silencio como de presencia que se pudieran dar en otras clases.

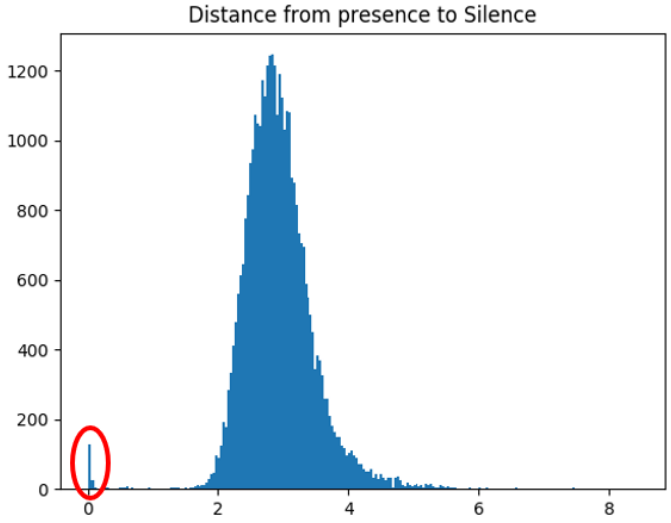



*Figura 5: Distancia euclídea de los vectores de las clases pasos y puertas a un ejemplo de vector de la clase silencio (elaboración propia)*

### 3.3. Entrenamiento

El proceso de entrenamiento se llevó a cabo utilizando la herramienta de procesamiento en la nube de *Google Colab*, ya que permite la utilización gratuita de GPUs, que permiten a su vez el entrenamiento más rápido de cualquier arquitectura que requiera cálculos matriciales como los de la red convolucional 2D completa de la primera metodología. Para la arquitectura del *encoder* pre entrenado se optó por transformar todas las muestras en sus correspondientes vectores característicos primero y usar esos vectores para entrenar la red neural densa directamente. Con esto se logró ahorra un gran tiempo en el apartado de entrenamiento que permitió realizar más iteraciones y mejorar el modelo más rápidamente.

Para ambos métodos se utilizaron técnicas de regularización como el *Dropout*, o *Batchnormalization* para el método 2DCNN y *Early Stopping* para ambos. La ponderación de clases también se utilizó en ambas soluciones para tratar los problemas derivados de tener un conjunto de datos desequilibrado.

Todos los entrenamientos se hicieron usando el optimizador Adam con una tasa de aprendizaje de 0.001 y la entropía cruzada como función de pérdida para el modelo.

Los resultados de los diferentes modelos se obtuvieron para 3 variantes de redes neuronales convolucionales y para el método del *encoder* para un clasificador estándar y para el mismo clasificador aplicando *dropout.*

En la Figura 6 y Figura 7 se pueden observar las curvas de entrenamiento para las variantes definitivas del modelo Encoder + Clasificador (E + C) y 2DCNN respectivamente.

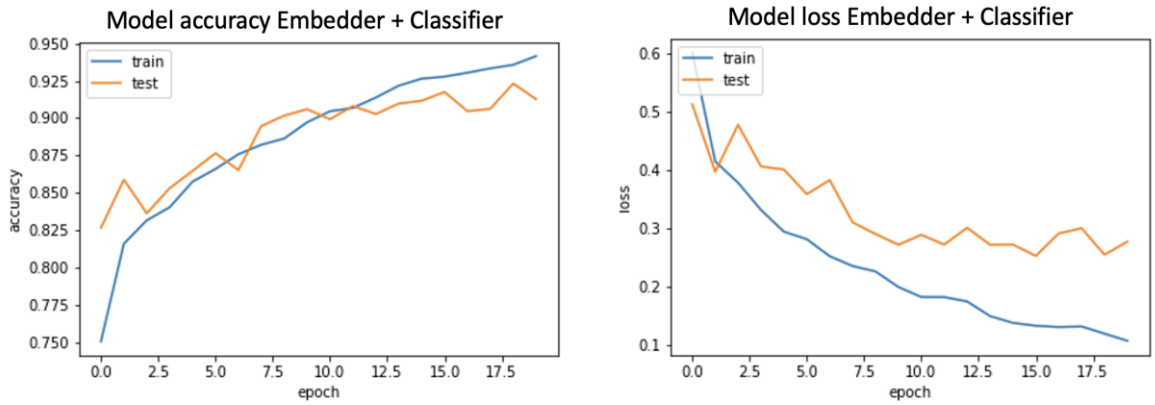



*Figura 6: Curvas de precisión y función de coste del entrenamiento del modelo Embedding (Elaboración propia)*

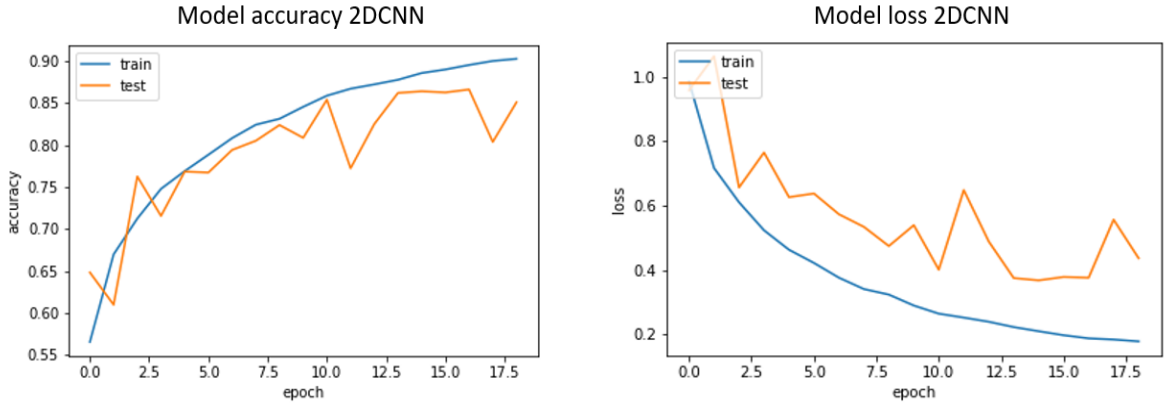



*Figura 7: Curvas de precisión y función de coste del entrenamiento del modelo 2DCNN (Elaboración propia)*

Para mejorar el entrenamiento e ir consiguiendo mejores resultados en cada iteración se emplearon tanto las curvas de la Figura 6 y la Figura 7, como las matrices de confusión del modelo en el *set* de *test*. Un ejemplo de estas matrices se puede ver en la Figura 8. Este

procedimiento permitió el uso de la realimentación para poder aplicar mejoras tanto en el pre procesado de datos, como en el entrenamiento y, basándose en la salida del modelo, poder mejorarlo poco a poco.

## 4. Resultados

El resultado del proyecto se puede dividir en tres outputs diferentes.

### 4.1. Comparación de las técnicas

En primer lugar, una comparación de dos técnicas del estado del arte que son: el entrenamiento de una red convolucional completa y el uso de un *encoder* pre entrenado junto con un perceptrón multicapa para clasificar fragmentos de audio.

En la Tabla 1se pueden observar los KPIs de las versiones finales del modelo E + C y el modelo 2DCNN.

| Classes | Accuracy | | Precision | | Recall | | Specificity | |
|---|---|---|---|---|---|---|---|---|
| | 2DCNN | E + C | 2DCNN | E + C | 2DCNN | E + C | 2DCNN | E+C |
| Alarms | 99.10% | 99.58% | 67.65% | 80.95% | 98.57% | 95.77% | 99.08% | 99.57% |
| Glass | 98.99% | 99.07% | 58.89% | 50.70% | 84.13% | 81.82% | 98.97% | 99.06% |
| Cry | 100.00% | 99.50% | 100.00% | 94.60% | 100.00% | 97.37% | 100.00% | 99.45% |
| Dog | 99.15% | 97.35% | 95.18% | 86.23% | 89.34% | 92.33% | 98.98% | 96.82% |
| Fire | 100.00% | 99.87% | 100.00% | 91.38% | 100.00% | 100.00% | 100.00% | 99.87% |
| Others | 97.83% | 98.94% | 94.14% | 96.82% | 91.30% | 88.45% | 96.68% | 98.43% |
| Presences | 96.02% | 98.52% | 81.69% | 93.97% | 82.53% | 94.38% | 95.17% | 98.07% |
| Shotgun | 98.74% | 98.91% | 84.72% | 86.10% | 89.79% | 89.75% | 98.64% | 98.83% |
| Total (macro average) | 98.73% | 98.97% | 85.28% | 85.09% | 91.96% | 92.48% | 98.44% | 98.76% |
| Total (micro average) | | | 89.83% | 91.73% | 89.83% | 91.73% | 89.83% | 91.73% |

*Tabla 1: Comparativa de los KPIs de las versiones definitivas de los modelos 2DCNN y Encoder + Clasificador (E+C)*

Para la comparación final se establecieron 3 factores diferentes:

- **Precisión en las clases establecidas**: esto se puede analizar mediante los KPIs de la Tabla 1. En esta tabla se aprecia que ambos modelos tienen grandes resultados y no hay ninguno de los dos que supere al otro en todos los aspectos. La conclusión final del análisis de precisión es que el modelo E + C debe ser el elegido si se atiende únicamente a los KPIs de los modelos, ya que es el que mejor se ajusta al objetivo original y principal del proyecto, que es el de detectar posibles situaciones de robo. La clase Presencia es de vital importancia en este aspecto y una diferencia tan grande en el *recall* de esta clase hace que sea la mejor opción a escoger en base a los requerimientos y el objetivo final del proyecto.
- **Tiempos de inferencia y entrenamiento**: en la Tabla 2 se ve como el modelo 2DCNN es lógicamente más rápido en la inferencia, ya que es una versión simplificada, pero más flexible del modelo E + C. También cabe resaltar que para el resultado del proyecto el tiempo realmente importante es el de inferencia, ya que va a ser lo que facilite una mejor puesta en producción posterior del modelo. Tener buen tiempo y flexibilidad en el entrenamiento es relevante si se quieren tener un modelo flexible y rápidamente adaptable, sin embargo, la inferencia es más determinante en el entorno de producción.

| • **Audio** | **Samples (#)** | **2DCNN** | | **Embeddings** | |
|---|---|---|---|---|---|
| | | **Time (s)** | **T/sample (s)** | **Time (s)** | **T/sample (s)** |
| Cry | 13 | 0.2778 | 0.0213 | 0.6028 | 0.0463 |
| Dog | 9 | 0.0456 | 0.0035 | 0.6973 | 0.0774 |
| Door | 21 | 0.0921 | 0.0043 | 0.5014 | 0.0238 |
| Fire | 1 | 0.0171 | 0.0171 | 0.2299 | 0.2299 |
| Glass | 5 | 0.1102 | 0.0220 | 0.3497 | 0.0699 |
| Gun | 2 | 0.0103 | 0.0051 | 0.2756 | 0.1378 |
| Others | 7 | 0.0418 | 0.0059 | 0.2946 | 0.0420 |
| Steps | 59 | 0.2056 | 0.0034 | 0.9035 | 0.0153 |
| Average time | 117 | 0.0068 | | 0.0329 | |

*Tabla 2: Comparación de tiempos de inferencia de cada uno de los modelos para 8 ejemplos de audios*

- **Escalabilidad**: puesto que el número y el tipo de clases puede crecer en el futuro, según las necesidades de la industria o de los clientes, es necesario que la solución fuera lo más escalable posible. Esto se consigue más fácilmente con el modelo E+C, ya que el modelo 2DCNN es una solución más *ad hoc* y que saturaría más rápidamente con un aumento del número de clases. Mientras que el modelo *vgg*, sobre el que se basa la otra arquitectura, está pensado para poder clasificar sin problemas miles de clases.

### 4.2. Modelo elegido

En segundo lugar, el modelo elegido e implementado para resolver el problema planteado en los requerimientos del proyecto. Este modelo fue el modelo E + C, ya que logró unos KPIs algo mejores para el problema planteado y una mayor flexibilidad en caso de querer aumentar o cambiar el número y tipo de clases de salida. En cuanto al tiempo de inferencia lo verdaderamente importante en este análisis es el estudio de estos tiempos frente al mundo real donde se va a implantar la solución. Cada muestra analizada se corresponde con un audio de 960 ms de longitud, sin embargo, hay que tener en cuenta que cada muestra contiene 480 ms de audio aun no analizado y otros 480 ms de audio que solapa con la muestra anterior y que por tanto no se corresponden con audio aún no "escuchado" por el modelo. De esta reflexión se puede concluir que el tiempo máximo que el tiempo por muestra máximo del modelo para que este pueda funcionar en tiempo real debe ser inferior a 0.48 segundos, lo cual cumplen ambos modelos para su peor tiempo por muestra (ambos marcados en rojo en la Tabla 2).

En la Figura 8 se muestra la matriz de confusión final del modelo implementado con las clases finales seleccionadas.

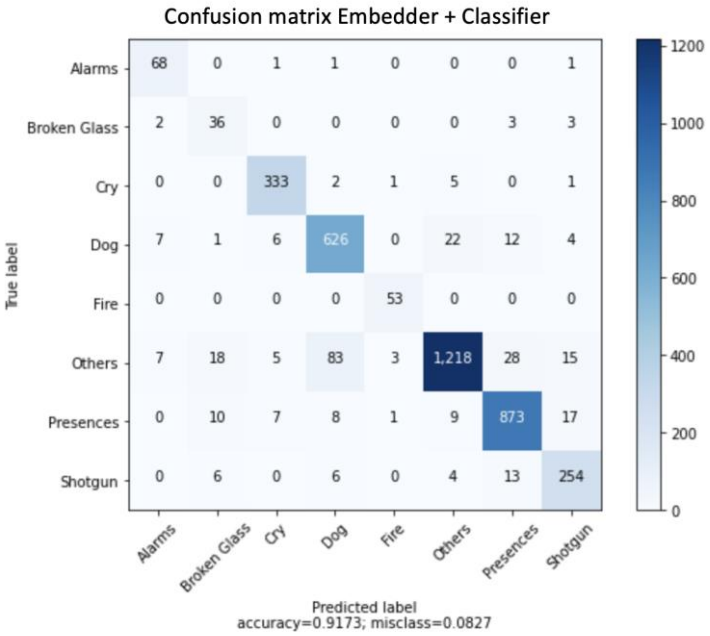



*Figura 8: Matriz de confusión final para el modelo elegido (E + C)*

### 4.3. Recomendaciones finales para la implementación

Conjuntamente con el estudio de las arquitecturas y sus resultados para el problema propuesto se elaboró un estudio de sensibilidad ROC, ya que se trata de un proyecto en el que un umbral para reducir mejorar los KPIs Verdadero Positivo frente a Falso Positivo resulta realmente interesante. Esto se debe al contexto en el que se enmarca el proyecto, ya que en las clases que posteriormente van a representar un robo nos interesa que no se nos escape ningún Verdadero Positivo cuando el modelo esté activo, aunque ello pueda derivar en un aumento de los Falsos Positivos. Se aplicaría la filosofía de “más vale prevenir que curar”.

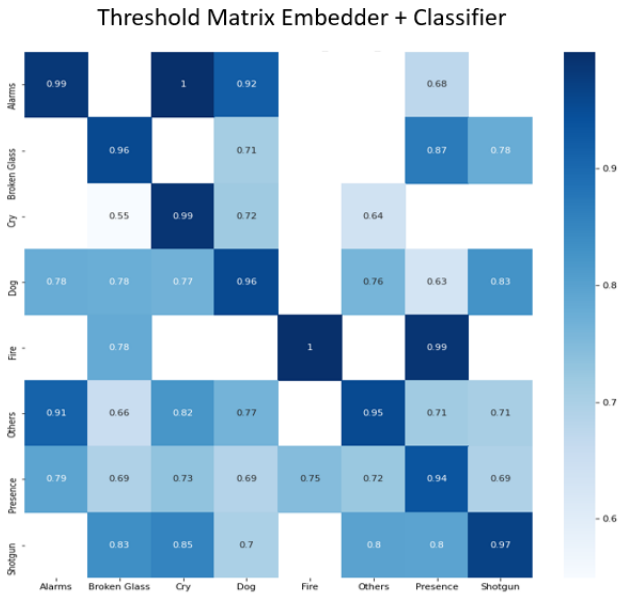



*Figura 9: Matriz de niveles de confianza medios del modelo para cada una de las clases*

La matriz de la Figura 9 muestra como el nivel de confianza medio en las predicciones del modelo para los Verdaderos Positivos (diagonal principal) es mucho mayor que para los Falsos Positivos. Esto implica que simplemente añadiendo un umbral como parámetro a la implementación final del modelo se obtendrían mucho mejores resultados. Este umbral se puede ajustar dependiendo de la clase, pero un nivel general aceptable estaría en torno al 90%.

### 4.4. Herramienta para pruebas y visualización

Por último, se elaboró una herramienta web y un script de Python para hacer análisis y clasificación de audios de manera sencilla y poder estudiar cómo se comporta el modelo ante distintos inputs. Además, durante todo el proyecto se elaboraron diferentes *Notebooks*, para poder realizar los análisis pertinentes, obtener las visualizaciones necesarias para entender los problemas y hacer un estudio detallado de los resultados y de aquellos audios que requiriesen de un análisis más en profundidad. La gran mayoría de las imágenes y resultados mostrados en este documento vienen de estos *Notebooks* o del proceso de entrenamiento de los modelos.

En el Código 1 se presenta un ejemplo de la salida de la aplicación con los resultados del modelo y el tiempo de procesamiento para el audio de entrada.


```
{
    "predictions_every_960_ms": [
        "Presence",
        "Presence",
        "Presence",
        "Presence",
        "Presence"
    ],
    "possible_classes": [
        "Alarms",
        "Broken_Glass",
        "Cry",
        "Dog",
        "Fire",
        "Others",
        "Presence",
        "Shotgun"
    ],
    "final_result": "Presence",
    "time_to_predict_(s)": 0.08271312713623047
}
```


*Código 1: Ejemplo de la salida del modelo (Elaboración propia)*

En este código se puede observar como la salida consta de 4 apartados:

- Clase predicha para cada uno de las ventanas de 960 ms en las que se divide el audio
- Las posibles clases de las que consta el modelo
- El resultado final: es el resultado más significativo de los detectados en el conjunto de las ventanas de 960 ms.
- Tiempo de predicción: tiempo transcurrido desde que se ha empezado a tratar el audio hasta la salida de su predicción final.

## 5. Conclusiones

El proyecto permitió la obtención de un modelo a la altura del nivel del estado del arte en materia de clasificación de audio, pero con unas aplicaciones concretas y una solución orientada a un problema específico de la industria de la seguridad. El modelo y la aplicación desarrollados resultan una innovación en esta industria.

En el momento de redactar esta memoria los *stakeholders* del sector de la seguridad han comenzado a integrar el modelo en sus sistemas y realizado sus propias pruebas con el modelo con resultados muy satisfactorios. Cabe destacar que se cumplieron todos los requerimientos solicitados y además se amplió el alcance del proyecto incluyendo la variable del *comfort* como valor añadido para los clientes finales del sistema.

## 6. Referencias


[1] A. Guzhov, F. Raue, J. Hees y A. Dengel, «Esresnet: Environmental sound classification based on visual domain models,» 2020.

[2] K. Palanisamy, D. Singhania y A. Yao, «Rethinking CNN Models for Audio Classification,» 2020.

[3] D. Gartzman, «Getting to Know the Mel Spectrogram,» August 2019. [En línea]. Available: https://towardsdatascience.com/getting-to-know-the-mel-spectrogram-31bca3e2d9d0. [Último acceso: September 2021].

[4] L. Nannia, G. Maguoloa, S. Brahnamb y M. Pacic, «An Ensemble of Convolutional Neural Networks for Audio Classification,» Julio, 2020.

[5] S. H. e. at, «CNN architectures for large-scale audio classification,» de *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2017.

[6] K. Simonya y A. Zisserman, «Very Deep Convolutional Neural Networks for Large Scale Image Recognition,» de *Conference Paper at ICLR*, 2015.

[7] A. Agrawal, «Towards Data Science,» 6 Junio 2020. [En línea]. Available: https://medium.com/dataseries/the-current-state-of-the-art-in-natural-language-processing-nlp-5c440f889e15. [Último acceso: November 2020].

[8] J. Salamon y J. P. Bello, «Deep Convolutional Neural Networks and DataAugmentation for Environmental SoundClassification,» *IEEE SIGNAL PROCESSING LETTERS*, 2016.

# IMPROVEMENT OF ALARM AND VIDEO SURVEILLANCE SYSTEMS BY INCORPORATING AUDIO ANALYSIS

**Author: Menendez Ruiz de Azua, Alberto**
Supervisor: López López, Álvaro; Güitta López, Lucia
Collaborating Entity: ICAI – Universidad Pontificia Comillas

## ABSTRACT

The project has succeeded in creating, training and implementing various artificial intelligence models for audio detection and classification in the context of security and video surveillance. State-of-the-art results have been achieved and a study has been carried out comparing different architectures and types of models in order to select those that best fit the specific problems of the project.

**Keywords**: Spectrogram, Artificial Intelligence, Neural Networks, Audio Analysis, Security

### 1. Introduction

This project arose as a response to a limitation in the security industry, more specifically in the alarm and video surveillance sector. The need for a solution is based on the need to offer a better service to customers in this sector and to increase the reliability and efficiency of the systems that currently exist.

The central idea of the project, and the response to it, arose from an exhaustive analysis by experts in the security sector of the cases of failure of their systems and the limitations they usually encounter. After a study of the cases in which these systems fail or do not achieve adequate efficiency, it was determined that a fraction of the thefts or incidents that were omitted by video surveillance could have been detected and, consequently, could have been avoided if an analysis of the audio at the specific moment of the incident had simply been carried out. This analysis would be complementary to the CCTV video and in no way a substitute for it.

Therefore, the main purpose of this project is to satisfy this need and to develop an audio classification method that works in a complementary way to the current security cameras, providing them with a capacity that they have lacked until now and allowing them to improve their efficiency and contribute to the security of the end users of the product.

Finally, it has been decided not to limit the project to the exclusive field of security and to take advantage of the enormous possibilities that audio classification offers in terms of

comfort for the end user of the security system. These possibilities are limited only by the classes in which the audio is classified, as there are classes such as a child crying or a fire crackling, which can alert the user to incidents that do not necessarily imply a burglary, but make the system completer and more attractive for the end consumer.

The project seeks to add value to current technology in two ways:

- By improving the security efficiency of the systems.
- By providing comfort capabilities for the end user.

The overall objective of the project was determined as: the development and implementation of state-of-the-art audio classification techniques, aimed at detecting security incidents and user comfort.

## 2. Project Definition

The first step in completing the project's objectives was to investigate the state of the art of both pioneering security techniques and audio processing and classification technologies.

Several interesting insights emerged from this analysis. Firstly, that the most pioneering techniques in the state of the art regarding audio classification were mostly *machine learning* technologies and more specifically in the field of *deep learning* and neural networks. [1] and [2] represent the current state of the art in audio classification, achieving the best KPIs in the GITZAN, ESC50 and UrbanSound8K datasets. Once within this field, the second important aspect is the scarcity of data in the context of security, as it is a sensitive field and data in this respect is very restricted. This makes it very difficult to train and implement *machine learning* techniques. Finally, the third important point that was detected in the state-of-the-art study was the fact that working with audio signals is complicated and that is why most of the state-of-the-art solutions are based on the transformation of audio waves into images and the subsequent application of state-of-the-art image classification techniques, such as inception, vgg19 models.... This idea represents the central point of the two state-of-the-art methods mentioned above and is the one used throughout this project.

In order to employ these techniques, the first step is to convert the audio into images that can be easily classified by the models. This is achieved in this project by means of a sound wave conversion known as Mel Spectrogram [3], which employs in the audio conversion ideas

such as the use of logarithmic scales, which simulate the way in which the human ear deals with the processing of this information. Figure 1 shows examples of the Mel spectrograms generated from 4 audios used throughout the project for training and validation of the models.

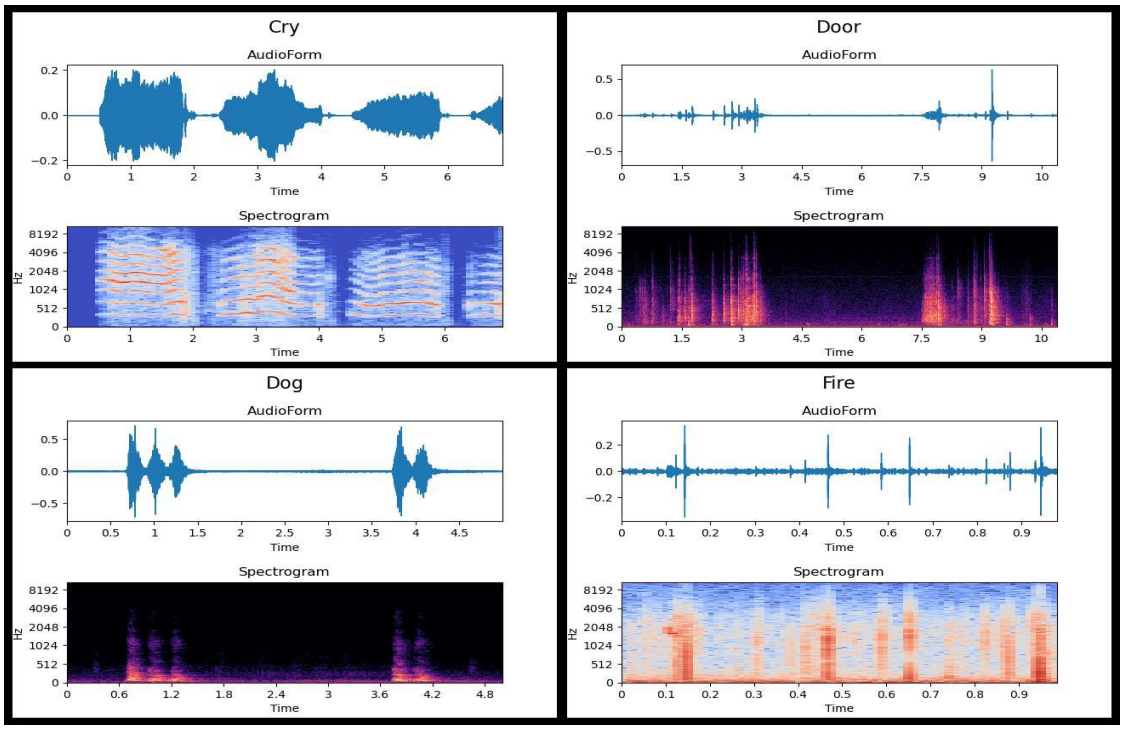



*Figure 1: Example of the Mel Spectrogram of 4 of the audios used in the project (Own elaboration)*

Once the state of the art was reviewed, the use of artificial intelligence techniques was determined and moved on to the central part of the project, where the development and training of various model architectures started. This process consisted of three stages:

1. Collection of audio data for training the models.
2. Analysis, pre-processing and cleaning of the data for training optimization.
3. Training and analysis of the KPIs of the different architectures.
4. Comparison of the results of the different models to determine the best solution for our project.

Steps 3 and 4 were repeated in loops and by changing architectures and details until the desired results were obtained for some of the models and a proper comparison could be made. The improvement process was based on the analysis of the confusion matrices of the

model and the training graphs, followed by a study of the most problematic classes in the matrix and thus being able to carry out the relevant actions to improve the KPIs: cleaning of the data, modification of hyper parameters, training techniques...

After these stages, and once the model that best suited the requirements of the project had been selected, it was implemented and put into production. The classes to be used were defined in the requirements stage and prior to the training of the architectures, as well as the minimum requirements of the models to be put into production. It will be seen later on how these classes changed throughout the project to address certain difficulties and refine the final result.

The deployment of the model was done in the first instance through a web App that allowed uploading audios and obtaining a result of these in real time, so that the client could test and get an idea of the subsequent implementation on its platform. Code 1 presents an example of the output of the application with the results of the model and the processing time for an input audio, in order to get a sense not only of the accuracy, but also of the inference time, which is key in the final applications of the model. The final delivery was done through a Python package with a folder to put the audios to be analysed (input) and another folder in which the results of analysing these audios are stored (output).

## 3. Model/system/tool description

### 3.1. Architectures

Once the general method to be followed and used for the project (convolutional neural networks for classifying Mel spectrograms of audio) had been decided, the dilemma arose as to which of all the architectures available for this type of technique would give the best results when implementing the project [4]. Therefore, the project carried out an analysis of two different architectures for audio classification, as well as two variants of these architectures, in order to determine which of the two would give the best results for the project. The two architectures compared and studied were:

- Full convolutional network (2DCNN): this architecture consists of a convolutional network built and trained in its entirety and without using any type of *transfer learning*. This type of network allowed absolute control over all the parameters of the model, allowing great flexibility and possibilities when experimenting and testing

different configurations. An example of the general architecture of this type of solution can be seen in Figure 2.

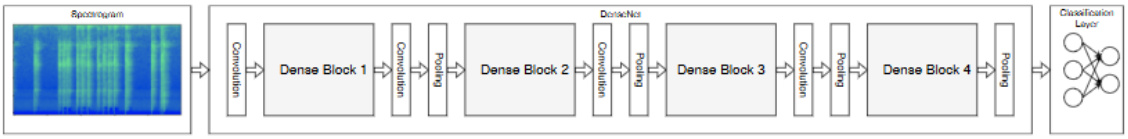



*Figure 2: General Arquitecture of a convolutional neural network for audio classification (source: [1])*

- Encoder plus classifier (E + C): this architecture has an encoder pre-trained on a large audio *dataset*, followed by its own classifier to classify the different outputs of the *encoder*. Specifically, the *encoder* used is *Google*'s Vggish, which has been pre-trained on the Youtube 8M audio dataset. Unlike the first option, we only had absolute control over the architecture of the classifier, but we have the advantage of using a model whose training would be impossible with the available resources. In addition, *fine tuning* techniques are allowed on this model. In this case the structure of the architecture is also similar to that of Figure 2, only that the convolutional stages correspond to a deeper architecture pre-trained by *Google*, based on the Vgg-19 neural network for image classification, the audio variant is known as Vggish [5].

The choice of the final model was not only based on the results of the training KPIs of the architectures, but also considered other factors such as: scalability of the model, flexibility in increasing and changing the number of classes, inference time or retraining time in case of adding new classes or modifying existing ones.

### 3.2. Data and pre-processing

Before entering either of the two models, the audios are subjected to a prior process in which they are converted to the optimal format for their entry into the architectures. This process consists of the following stages:

1. Transformation of the audio input from .wav or .mp3 format to its corresponding Mel spectrogram with the following parameters for the transformation:
    - Sampling frequency: 44100 Hz

- Audio duration: 960 ms (set by the sizes by the pre-trained Vggish architecture)
- Fourier transform window duration: 25ms
- Moving window jump for the Fourier transform: 10 ms
- Number of mels to separate the frequency space into: 64

The results of converting an audio waveform into a logarithmic spectrogram of Mel can be seen in Figure 3.

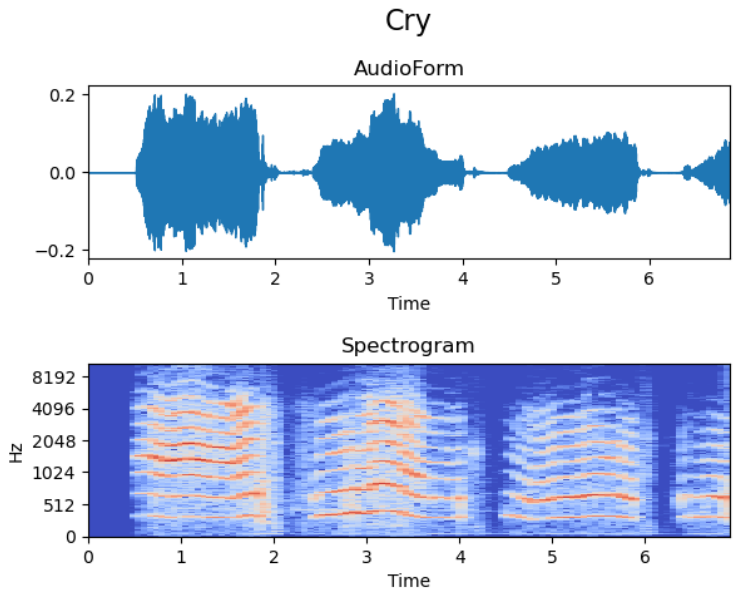



*Figure 3: Example of an audio waveform of a baby crying converted to a logarithmic spectrogram of Mel (own elaboration).*

2. Segmentation of these spectrograms into images in the appropriate format for each of the architectures (96 x 64 pixels). This translates into audio fragments of 960 ms, since the moving window jump used for the Mel Spectrogram is 10ms (960 ms / 10 ms/pixel = 96 pixels). These fragments were taken with an offset of 480 ms in order to overlap different sections and avoid leaving areas that would allow audio to be recognised as cut off and difficult for the models to classify.

Another aspect that was carried out was a pre-processing of the data before any type of training, since in some of the first tests that were carried out, an error was detected in both models that was derived from an incorrect labelling of the data. This error was due to the fact of using 960 ms moving windows as input for the models and to the fact that the

characteristic sound that gave name to the audio label was not produced in all these windows and there could be some that were simply a silence or any background noise that did not correspond to the class to which it had been labelled, but to another of the available classes. An example of this is shown in Figure 4.

.

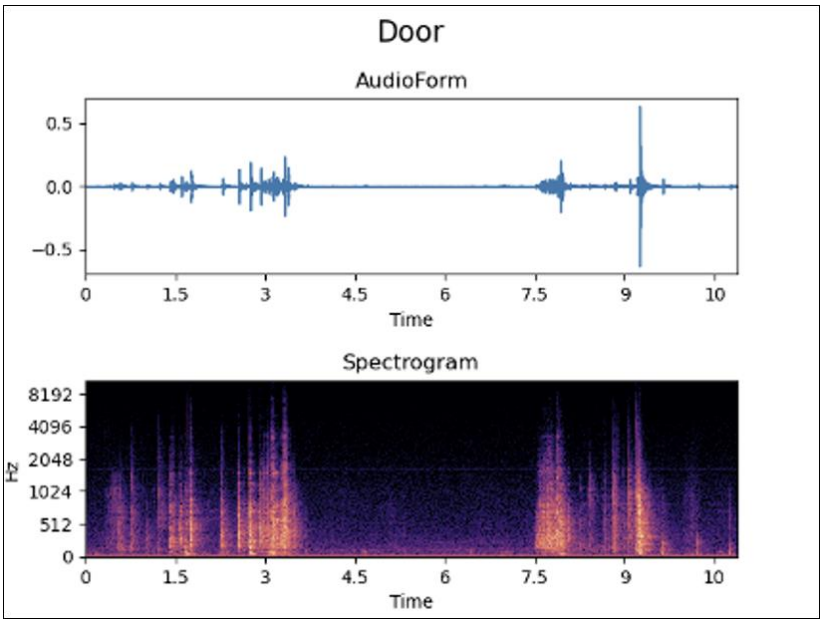



*Figure 4: Audio spectrogram of a gate showing the lack of information for any sample whose 960 ms span the range from 3.5 to 7.5 seconds (own elaboration).*

To solve this problem, the encoder of the second architecture was used to generate the characteristic vectors of each of the 960 ms windows. With these vectors of dimension 128, unsupervised learning techniques were used, taking advantage of the fact that there were examples of vectors that were known with certainty to belong to the class sought, and the candidates for that class were compared with a Euclidean metric (norm). In Figure 5 it can be seen that there are a number of examples in the presence class (steps plus gates) that are very close to the artificial centroid of the Silence class, generated in order to do this cleaning. This allowed for the removal of mislabelled silences and background noises in the training set. This technique was used to clean classes of both possible silence and presence samples that could occur in other classes due to the technique used for the pre-processing of the data.

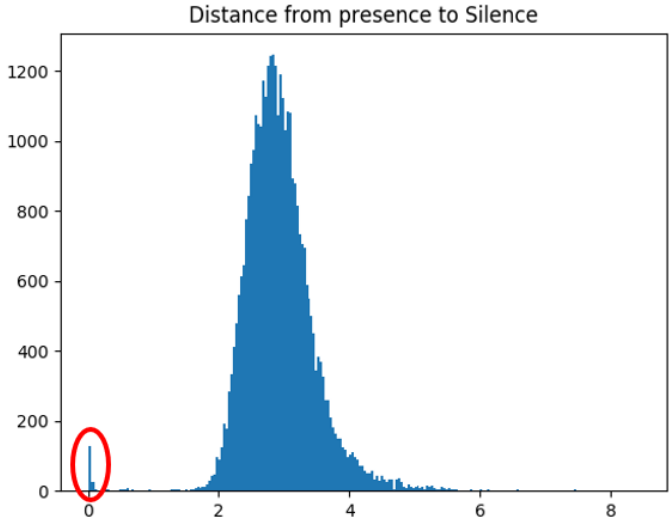



*Figure 5: Euclidean distance of the vectors of the classes Steps and Doors to an example vector of the class silence.*

### 3.3. Training

The training process was carried out using the *Google Colab* cloud processing tool, as it allows the free use of GPUs, which enables faster training of any architecture that requires matrix computations such as those of the full 2D convolutional network of the first methodology. For the pre-trained encoder architecture, we opted to transform all samples into their corresponding characteristic vectors first and use these vectors to train the dense neural network directly. This saved a lot of time in the training section, which allowed for more iterations and faster model improvement.

For both methods regularisation techniques were used such as *Dropout*, or *Batchnormalization* for the 2DCNN method and *Early Stopping* for both. Class weighting was also used in both solutions to deal with the problems arising from having an unbalanced dataset.

All training was done using the Adam optimiser with a learning rate of 0.001 and a categorical cross-entropy as a loss function for the model.

In Figure 6 and Figure 7 the training curves for the final variants of the Encoder + Classifier and 2DCNN model respectively can be seen.

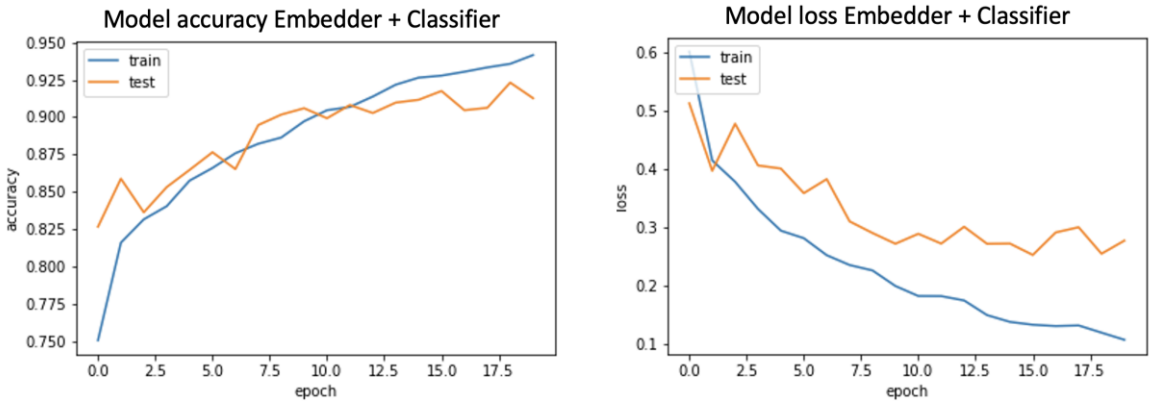



*Figure 6: Accuracy curves and training cost function of the Embedding model training (Own Elaboration)*

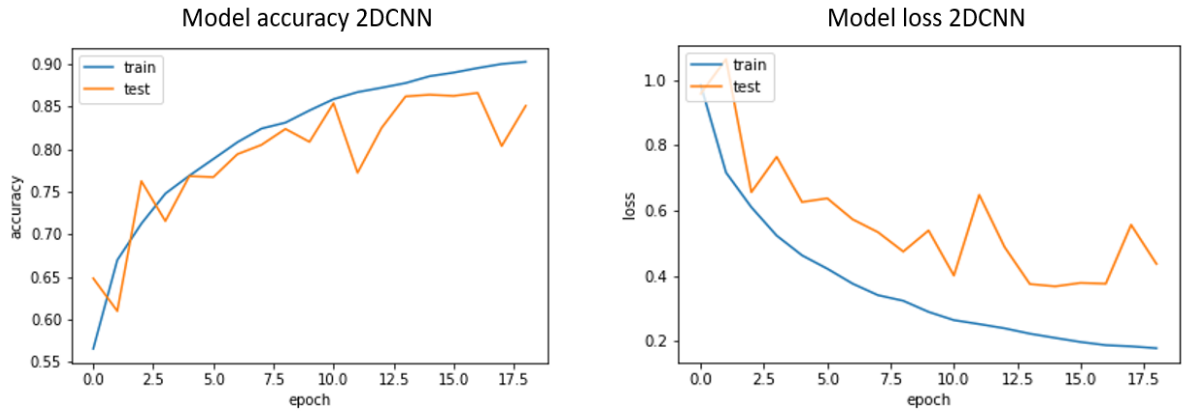



*Figure 7: Accuracy curves and training cost function of the 2DCNN model training (Own elaboration)*

To improve the training and achieve better results in each iteration, both the curves in Figure 6 and Figure 7 were used, as well as the confusion matrices of the model in the test set. An example of this confusion matrices can be seen in Figure 8. This procedure allowed the use of feedback to apply improvements in both the data pre-processing and training processes and, based on the model output, gradually improve the model.

## 4. Results

The outcome of the project can be divided into three different outputs.

### 4.1. Comparison of the techniques

Firstly, a comparison of two state-of-the-art techniques, namely, the training of a full convolutional network and the use of a pre-trained encoder together with a multilayer perceptron to classify audio fragments.

Table 1 shows the KPIs of the final versions of the model (E + C) and the 2DCNNN model.

| Classes | Accuracy | | Precision | | Recall | | Specificity | |
|---|---|---|---|---|---|---|---|---|
| | 2DCNN | E + C | 2DCNN | E + C | 2DCNN | E + C | 2DCNN | E+C |
| Alarms | 99.10% | 99.58% | 67.65% | 80.95% | 98.57% | 95.77% | 99.08% | 99.57% |
| Glass | 98.99% | 99.07% | 58.89% | 50.70% | 84.13% | 81.82% | 98.97% | 99.06% |
| Cry | 100.00% | 99.50% | 100.00% | 94.60% | 100.00% | 97.37% | 100.00% | 99.45% |
| Dog | 99.15% | 97.35% | 95.18% | 86.23% | 89.34% | 92.33% | 98.98% | 96.82% |
| Fire | 100.00% | 99.87% | 100.00% | 91.38% | 100.00% | 100.00% | 100.00% | 99.87% |
| Others | 97.83% | 98.94% | 94.14% | 96.82% | 91.30% | 88.45% | 96.68% | 98.43% |
| Presences | 96.02% | 98.52% | 81.69% | 93.97% | 82.53% | 94.38% | 95.17% | 98.07% |
| Shotgun | 98.74% | 98.91% | 84.72% | 86.10% | 89.79% | 89.75% | 98.64% | 98.83% |
| Total (macro average) | 98.73% | 98.97% | 85.28% | 85.09% | 91.96% | 92.48% | 98.44% | 98.76% |
| Total (micro average) | | | 89.83% | 91.73% | 89.83% | 91.73% | 89.83% | 91.73% |

*Table 1: Comparison of the KPIs of the final versions of the 2DCNN and Encoder + Classifier (E+C) models.*

For the comparison, 3 different factors were established:

- **Accuracy in the established classes**: this can be analysed by the KPIs in Table 1. In this table it can be seen that both models have great results and there is no one of the two that outperforms the other in all aspects. The final conclusion of the accuracy analysis is that the E + C model should be the one chosen if only the KPIs of the models are considered, since it is the one that best fits the original and main objective of the project, which is to detect possible theft situations. The Presence class is of vital importance in this aspect and such a big difference in the recall of this class

makes it the best option to choose based on the requirements and the final objective of the project.

- **Inference and training times**: Table 2 shows how the 2DCNN model is logically faster in the inference, since it is a simplified but more flexible version of the E + C model. It is also worth noting that for the outcome of the project the really important time is the inference time, since it will be the one that will facilitate a better subsequent production of the model. Having good time and flexibility in training is relevant if you want to have a flexible and quickly adaptable model, however, inference is more determinant in the production environment.

| • **Audio** | **Samples (#)** | **2DCNN** | | **Embeddings** | |
|---|---|---|---|---|---|
| | | **Time (s)** | **T/sample (s)** | **Time (s)** | **T/sample (s)** |
| Cry | 13 | 0.2778 | 0.0213 | 0.6028 | 0.0463 |
| Dog | 9 | 0.0456 | 0.0035 | 0.6973 | 0.0774 |
| Door | 21 | 0.0921 | 0.0043 | 0.5014 | 0.0238 |
| Fire | 1 | 0.0171 | 0.0171 | 0.2299 | 0.2299 |
| Glass | 5 | 0.1102 | 0.0220 | 0.3497 | 0.0699 |
| Gun | 2 | 0.0103 | 0.0051 | 0.2756 | 0.1378 |
| Others | 7 | 0.0418 | 0.0059 | 0.2946 | 0.0420 |
| Steps | 59 | 0.2056 | 0.0034 | 0.9035 | 0.0153 |
| Average time | 117 | 0.0068 | | 0.0329 | |

*Table 2: Comparison of inference times of each of the models for 8 examples of audios.*

- **Scalability**: since the number and type of classes may grow in the future, depending on industry or customer needs, it is necessary that the solution be as scalable as possible. This is more easily achieved with the Encoder + Classifier model, as the 2DCNN model is a more ad hoc solution and would saturate more quickly with an increase in the number of classes. Whereas the vgg model, on which the other architecture is based, is designed to be able to classify thousands of classes without problems.

### 4.2. Chosen model

Secondly, the model chosen and implemented to solve the problem posed in the project requirements. This model was the model with the encoder, since it achieved somewhat better KPIs for the problem posed and greater flexibility in case you want to increase or change the number and type of output classes. As for the inference time, what is really important in this analysis is the study of these times against the real world where the solution is going to be implemented. Each analysed sample corresponds to an audio of 960 ms in length, however, it must be considered that each sample contains 480 ms of audio not yet analysed and another 480 ms of audio that overlaps with the previous sample and therefore does not correspond to audio not yet "heard" by the model. From this reflection it can be concluded that the maximum time per sample for the model to run in real time must be less than 0.48 seconds, which both models meet for their worst time per sample (both marked in red in Table 2).

Figure 8 shows the final confusion matrix of the implemented model with the final selected classes, where the Steps and Gates classes have been merged to form a single Presence class.

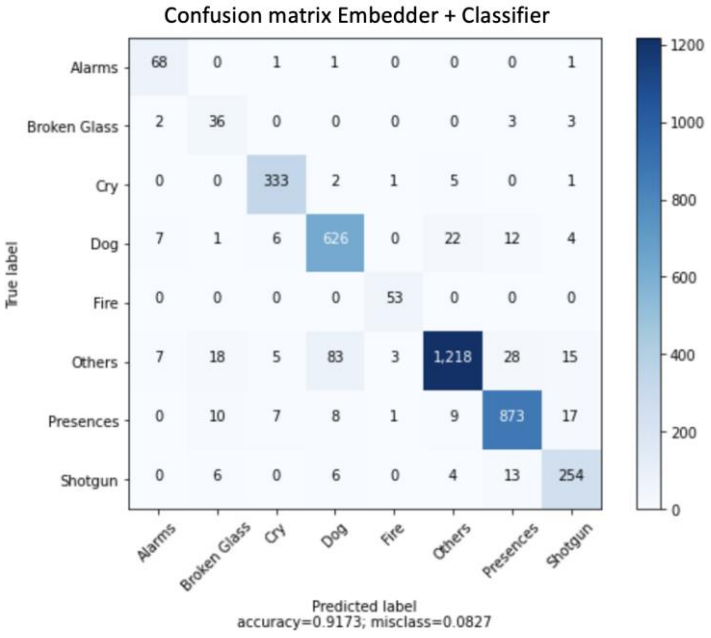



*Figure 8: Final confusion matrix for the chosen model (Embedder + Classifier)*

### 4.3. Final recommendations for implementation

Together with the study of the architectures and their results for the proposed problem, a ROC sensitivity study was carried out, as this is a project in which a threshold to reduce the True Positive versus False Positive KPIs is really interesting. This is due to the context in which the project is framed, since in the classes that will later represent a theft, we are interested in not missing any True Positive when the model is active, although this may lead to an increase in False Positives. The philosophy of "Prevention is better than cure" would apply.

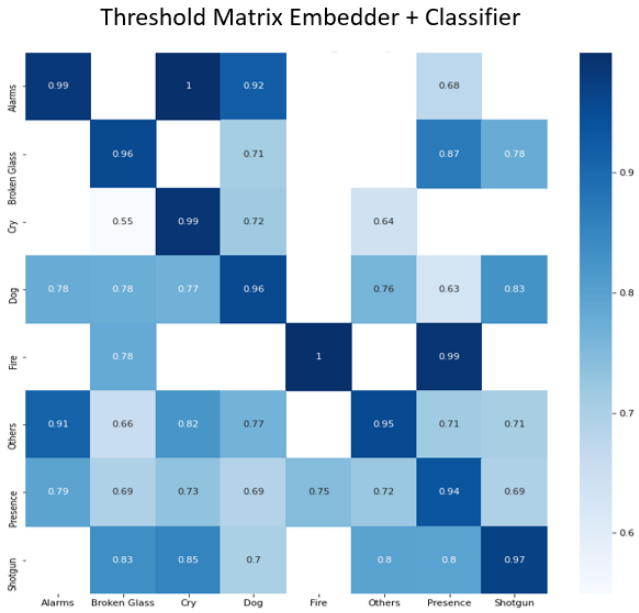



*Figure 9: Matrix of average confidence levels of the model for each of the classes*

The matrix in Figure 9 shows how the average confidence level in the model's predictions for True Positives (main diagonal) is much higher than for False Positives. This implies that simply adding a threshold as a parameter to the final implementation of the model would give much better results. This threshold can be adjusted depending on the class, but a general acceptable level would be around 90%.

### 4.4.Testing and visualisation tool

Finally, a web tool and a Python script were developed to perform audio analysis and classification in a simple way and to be able to study how the model behaves in response to different inputs. In addition, throughout the project, different *Notebooks* were created in order to carry out the relevant analyses, obtain the visualisations necessary to understand the problems and make a detailed study of the results and of those audios that required a more in-depth analysis. The vast majority of the images and results shown in this document come from these Notebooks or from the training process of the models.

An example of the output of the application with the model results and the processing time for the input audio is presented in Code 1.


```
{
    "predictions_every_960_ms": [
        "Presence",
        "Presence",
        "Presence",
        "Presence",
        "Presence"
    ],
    "possible_classes": [
        "Alarms",
        "Broken_Glass",
        "Cry",
        "Dog",
        "Fire",
        "Others",
        "Presence",
        "Shotgun"
    ],
    "final_result": "Presence",
    "time_to_predict_(s)": 0.08271312713623047
}
```


*Code 1: Example of the model output (Own elaboration)*

In this code you can see how the output consists of 4 sections:

- Predicted class for each of the 960 ms windows into which the audio is divided.
- The possible classes of which the model consists
- The final result: the most significant result of those detected in the set of 960 ms windows.
- Prediction time: time elapsed from the start of the audio processing to the output of its final prediction.

## 5. Conclusions

The project resulted in a state-of-the-art model for audio classification, but with concrete applications and a solution oriented to a specific problem in the security industry. The model and the application developed are an innovation in this industry.

At the time of writing, security industry stakeholders have started to integrate the model into their systems and have conducted their own tests with the model with very satisfactory results. It should be noted that all the requirements requested were met and the scope of the project was extended to include the comfort variable as an added value for the system's end customers.

## 6. References


[1] A. Guzhov, F. Raue, J. Hees y A. Dengel, «Esresnet: Environmental sound classification based on visual domain models,» 2020.

[2] K. Palanisamy, D. Singhania y A. Yao, «Rethinking CNN Models for Audio Classification,» 2020.

[3] D. Gartzman, «Getting to Know the Mel Spectrogram,» August 2019. [En línea]. Available: https://towardsdatascience.com/getting-to-know-the-mel-spectrogram-31bca3e2d9d0. [Último acceso: September 2021].

[4] L. Nannia, G. Maguoloa, S. Brahnamb y M. Pacic, «An Ensemble of Convolutional Neural Networks for Audio Classification,» Julio, 2020.

[5] S. H. e. at, «CNN architectures for large-scale audio classification,» de *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2017.

[6] K. Simonya y A. Zisserman, «Very Deep Convolutional Neural Networks for Large Scale Image Recognition,» de *Conference Paper at ICLR*, 2015.

[7] A. Agrawal, «Towards Data Science,» 6 Junio 2020. [En línea]. Available: https://medium.com/dataseries/the-current-state-of-the-art-in-natural-language-processing-nlp-5c440f889e15. [Último acceso: November 2020].

[8] J. Salamon y J. P. Bello, «Deep Convolutional Neural Networks and DataAugmentation for Environmental SoundClassification,» *IEEE SIGNAL PROCESSING LETTERS*, 2016.

# *Índice de la memoria*

# *Índice de figuras*

# *Índice de tablas*

# Capítulo 1. INTRODUCCIÓN

## 1.1 MOTIVACIÓN DEL PROYECTO

Este proyecto surge como respuesta a una limitación en la industria de la seguridad, más concretamente al sector de las alarmas y la video vigilancia. La exigencia de una solución viene fundamentada por la necesidad de ofrecer un mejor servicio a los clientes de este sector y aumentar la fiabilidad y eficacia de los sistemas que existen actualmente.

Hay muchos sectores donde un fallo en un sistema puede ser fácilmente solventado y no conlleva grandes pérdidas más allá del coste de volver a poner en funcionamiento ese sistema. Sin embargo, eso no ocurre en la seguridad y los sistemas de video vigilancia, donde un fallo en las cámaras o un detalle omitido por estas podría traducirse en un robo y en una consiguiente pérdida de confianza y credibilidad enorme, por parte, no solo de ese cliente, sino de todos aquellos que tengan instalado el mismo sistema.

La idea central del proyecto, y a la que se ha dado respuesta, surge de un análisis exhaustivo de los expertos del sector de la seguridad sobre los casos de fallo de sus sistemas y las limitaciones con las que se suelen encontrar. Tras un estudio de los casos en los que fallan o no se alcanza la eficiencia adecuada con los actuales sistemas de video vigilancia, se determinó, que una fracción de los robos o incidencias que fueron omitidos por la vídeo vigilancia se podrían haber detectado y, en consecuencia, haber sido evitados, si simplemente se hubiera llevado a cabo un análisis de audio. Este análisis sería complementario al vídeo de las cámaras de seguridad y en ningún caso sustitutivo del mismo.

Por tanto, la finalidad principal de este proyecto es lograr satisfacer esta necesidad y elaborar un método de clasificación de audio que funcione de forma complementaria a las actuales cámaras de seguridad, dotándolas de una capacidad de la que hasta ahora carecían y que les permita mejorar su eficacia y contribuir a la seguridad de los usuarios finales del producto. Es importante remarcar que en ningún momento se plantea que esta tecnología funcione como sustitutivo a las cámaras de seguridad, sino que se trataría de un complemento a las mismas. La metáfora de lo que se pretende buscar con este proyecto sería pensar que actualmente se cuenta con vigilantes que simplemente pueden ver y analizar aquello que reciben por el sentido de la vista (imágenes captadas por las cámaras), pero con la tecnología desarrollada en este proyecto se busca dotarlas de la capacidad de escuchar.

Finalmente, se ha optado por no limitar el proyecto al ámbito exclusivo de la seguridad y aprovechar las enormes posibilidades que la clasificación de audio aporta en temas de confort para el usuario final del sistema de seguridad. Estas posibilidades vienen limitadas únicamente por las clases en las que se elija clasificar el audio, ya que hay clases como el lloro de un niño o el chisporroteo de fuego, que pueden alertar al usuario de incidencias que no implican necesariamente un robo, pero hacen el sistema más completo y atractivo para un consumidor final.

En el proyecto se busca crear un valor añadido con respecto a la tecnología actual en dos aspectos:

- Mediante la mejora de la eficiencia en seguridad de los sistemas.
- Otorgándole capacidades de confort para el usuario final.

El objetivo global del proyecto se determinó como: el desarrollo e implementación de técnicas de vanguardia en materia de clasificación de audio, orientadas a la detección de incidencias de seguridad y comodidad de usuarios.

## 1.2 REQUERIMIENTOS

Las primeras reuniones con los *stakeholders* del proyecto definieron el alcance del proyecto y dieron lugar a los siguientes requerimientos:

- Se buscaría elaborar un modelo con una precisión superior al 90% para la clasificación de fragmentos de audio.
- Se establecieron como clases prioritarias a reconocer por el modelo todas aquellas relaciones con la detección de actos intrusivos en una vivienda, como pueden ser ruidos de presencia en una casa vacía (puertas, cajones, pasos …), rotura de cristales al tratar de irrumpir en la vivienda o ladrido de perros al tratar de irrumpir un asaltante en la propiedad.
- El tiempo de procesamiento del modelo debería ser lo menor posible, sin superarse nunca los 3 segundos.

El alcance del proyecto se determinó de la siguiente manera:

- Todo lo referente al modelo, su puesta a punto y los diferentes *tests* para cumplir los puntos expuestos en los requerimientos.
- Elaboración de una estrategia sencilla para que los *stakeholders* pudieran probar el modelo e integrarlo en sus sistemas para hacer sus propias pruebas.

## 1.3 OBJETIVOS DEL PROYECTO

A continuación, se establecieron los objetivos del proyecto en base a los requerimientos de la industria y el alcance determinado. Los objetivos se fijaron de cara a obtener la mejor solución posible al problema planteado originalmente y se resumieron en los siguientes puntos:

1. Análisis y comparación de diferentes técnicas para la clasificación de audio, orientado siempre a las necesidades del proyecto.
2. Experimentación y estudio de las diferentes técnicas sin perder de vista el uso final que se les va a dar y analizando su desempeño.
3. Selección e implementación de las técnicas que mejores resultados muestren en base a los objetivos del proyecto.
4. Puesta en producción de la solución y testeo de su funcionamiento acorde a los objetivos buscados.

## 1.4 OBJETIVOS DE DESARROLLO SOSTENIBLE

Este proyecto está alineado con dos de los objetivos de desarrollo sostenible propuestos por los líderes mundiales el 25 de septiembre de 2015. Los objetivos que aborda el proyecto son:

### 1.4.1 OBJETIVO 3: GARANTIZAR UNA VIDA SANA Y PROMOVER EL BIENESTAR PARA TODAS LAS EDADES

Este proyecto pretende mejorar el bienestar de las personas a través del confort y tranquilidad que van a entregar los sistemas de seguridad gracias al complemento de la clasificación de audio. Esto supone un gran aumento en el bienestar de los principales grupos consumidores de estos sistemas, que son familias o personas mayores, que se van a ver ampliamente beneficiados de las capacidades extra de estos sistemas, ya sea a través de tranquilidad o confort al saber que la tecnología está ahí y que no les va a fallar.

### 1.4.2 OBJETIVO 9: CONSTRUIR INFRAESTRUCTURAS RESILIENTES, PROMOVER LA INDUSTRIALIZACIÓN SOSTENIBLE Y FOMENTAR LA INNOVACIÓN

Este proyecto representa un claro ejemplo de innovación al introducir técnicas pioneras en una industria en la que no se han empleado todavía. También supone la apertura de puertas para que esta tecnología pueda ser utilizada en campos en los que nunca se había imaginado, permitiendo posibilidades que hasta ahora no se planteaban.

# Capítulo 2. ESTADO DE LA CUESTIÓN

El estudio del estado del arte se hizo con el alcance de las tecnologías y técnicas de clasificación de fragmentos de audio. Es importante resaltar el hecho de que se trata de fragmentos de audio independientes, en contraposición al caso del análisis continuo de audio como una secuencia, que suele conformar un problema del estilo NLP (*Natural Language Processing)* [6]. En este proyecto esto se tendrá en cuenta de otra manera, sin la necesidad de recurrir a técnicas de análisis secuencial. Esto viene fundamentado en gran parte por el contexto del problema, ya que, si se aproxima desde su resolución por una persona, no es estrictamente necesario conocer el contexto anterior o posterior para poder clasificar un audio y decir que es lo que se oye en el mismo. Por ejemplo, un ladrido de perro es fácilmente clasificable independientemente de si lo que se ha oído antes han sido otros ladridos de perro o cualquier otro sonido.

Una vez establecido el alcance de la solución a implementar y sobre qué tipo de solución centrar la búsqueda, se procedió al correspondiente análisis de las tecnologías disponibles al respecto.

En un primer análisis se detectó una serie de ideas interesantes. En primer lugar, que las técnicas más pioneras del estado del arte pasaban en su mayoría por tecnologías de *machine learning* (ML) y más concretamente por el campo del *deep learning*. Las redes neuronales de los estudios [1] y [2] representan el estado del arte actual en materia de clasificación de audio, logrando los mejores KPIs en los *datasets* de GITZAN, ESC50 y UrbanSound8K.

| Model | GTZAN | ESC-50 | UrbanSound8K |
|---|---|---|---|
| Choi, Keunwoo, et al.[45] | 89.80% | - | 69.10% |
| Multi-Stream Network[18] | - | 84.90% | - |
| Attention-Based CRNN[11] | - | 86.10% | - |
| ES-ResNet [32] | - | 91.50% | 85.42% |
| GTZAN [60] | 94.50% | - | - |
| DenseNet (Random) | 88.50% | 72.50% | 76.32% |
| DenseNet (Pretrained) | 91.39% | 91.16% | 85.14% |
| DenseNet (Pretrained Ensemble) | 90.50% | **92.89%** | **87.42%** |

*Figura 10: Tabla de resultados de la solución propuesta en el estudio [2] en los diferentes datasets*

| | Model | Source | Representation | ESC-10 | ESC-50 | US8K official | US8K unofficial |
|---|---|---|---|---|---|---|---|
| Others | Human (2015) | [3] | – | 95.70 | 81.30 | – | – |
| | **Raw waveform and 1D-CNN** | | | | | | |
| | EnvNet (2017) | [5] | raw | 88.10 | 74.10 | 71.10 | – |
| | EnvNet v2 (2017) | [6] | raw | 91.30 | 84.70 | 78.30 | – |
| | Multiresolution 1D-CNN (2018) | [7] | raw | – | 75.10 | – | – |
| | Gammatone 1D-CNN (2019) | [8] | raw | – | – | – | 89.00 [1] |
| | **Learnable filterbank and 2D-CNN** | | | | | | |
| | Piczak-CNN + ConvRBM (2017) | [9] | FBE | – | 86.50 | – | – |
| | **Time-frequency representation and 2D-CNN** | | | | | | |
| | Piczak-CNN (2015) | [10] | Mel-spec | 90.20 | 64.50 | 73.70 | – |
| | SB-CNN (2017) | [12] | Mel-spec | – | – | 79.00 | – |
| | GoogLeNet (2017) | [15] | Mel-spec, MFCC, CRP | 86.00 | 73.00 | – | 93.00 [2] |
| | Piczak-CNN (2017) | [18] | (TEO-)GT-spec | – | 81.95 | – | 88.02 [3] |
| | Piczak-CNN (2017) | [19] | (PE)FBE | – | 84.15 | – | – |
| | VGG-like CNN + mix-up (2018) | [21] | Mel-, GT-spec | 91.70 | 83.90 | 83.70 | – |
| | VGG-like CNN + Bi-GRU + att. (2019) | [22] | GT-spec | 94.20 | 86.50 | – | – |
| | TSCNN-DS (2019) | [24] | Mel-spec, MFCC, CST | – | – | – | 97.20 |
| | LMCNet (2019) | [24] | Mel-spec, CST | – | – | – | 95.20 |
| | LMCNet (no aug.) | *reproduced* [4] | Mel-spec, CST [5] | – | – | *74.04* | *94.00* |
| | TFNet (2019) | [27] | Mel-spec | 95.80 | 87.70 | – | 88.50 |
| | TFNet (no aug.) (2019) | [27] | Mel-spec | 93.10 | 86.20 | – | 87.20 |
| | TFNet (no aug.) | *reproduced* [6] | Mel-spec [7] | – | *79.45* | *78.50* | *96.69* |
| Ours | **ESResNet** | | | | | | |
| | from scratch | | log-power spec | 92.50 | 81.15 | 81.31 | (96.74) |
| | ImageNet pre-trained | | log-power spec | 96.75 | 90.80 | 84.90 | (98.18) |
| | **ESResNet-Attention** | | | | | | |
| | from scratch | | log-power spec | 94.25 | 83.15 | 82.76 | (96.83) |
| | ImageNet pre-trained | | log-power spec | **97.00** | **91.50** | **85.42** | (**98.84**) |

*Figura 11: Tabla de resultados de la solución propuesta en el estudio [1], sobre el que se ha basado una de las soluciones del proyecto*

Una vez dentro de este campo, el segundo aspecto importante es la escasez de datos en el contexto de la seguridad, ya que es un campo sensible y los datos al respecto están muy restringidos. Esto dificulta en gran medida el entrenamiento e implementación de técnicas de ML.

Por último, el tercer punto importante que se detectó en el estudio del estado del arte fue el hecho de que trabajar con señales de audio es complicado y es por eso que la mayor parte de las soluciones en la literatura se basan en la transformación de las ondas de audio en imágenes y en la posterior aplicación de técnicas del estado del arte de clasificación de imágenes, como los modelos *inception*, *vgg19*... Esta idea representa el punto central de los dos métodos del estado del arte mencionados anteriormente y es la que se empleó a lo largo de este proyecto Para ello, el primer paso es el de convertir el audio en imágenes y, a continuación, aplicar variantes de los modelos mencionados anteriormente. Esto se logra en este proyecto mediante una conversión de ondas sonoras conocida como *Espectrograma de Mel* [3], que emplea en la conversión del audio ideas que simulan la forma en la que el oído humano se enfrenta al tratamiento de esta información. En la Figura 12 se pueden observar ejemplos de los espectrogramas de Mel generados de 4 audios utilizados a lo largo del proyecto para el entrenamiento y validación de los modelos. El espectrograma de Mel se basa en cierta manera en el funcionamiento del oído humano, ya que como es bien sabido, nuestro sistema auditivo percibe el sonido de forma logarítmica y no lineal. De ahí que el sonido se suela trabajar en decibelios, los cuales son una escala logarítmica de la intensidad del sonido. Es por eso que la variante del espectrograma de Mel que se emplea para proyectos de clasificación de audio y la que se va a emplear en este proyecto es el Espectrograma de Mel logarítmico (más conocido por su nombre en inglés *log-Mel spectrogram)*.

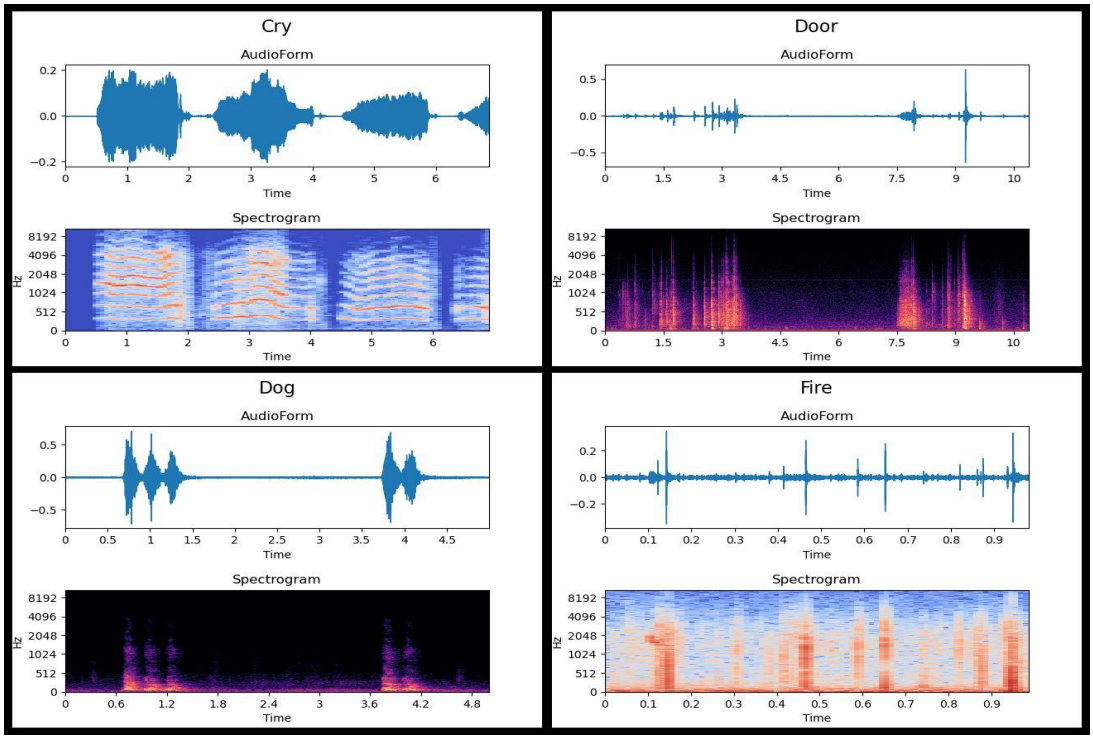



*Figura 12: Ejemplo del Espectrograma de Mel de 4 de los audios empleados en el proyecto (Elaboración propia)*

A continuación, se procede a una explicación detallada de estas tecnologías y métodos pertenecientes al estado del arte y que se han empleado en este proyecto.

## 2.1 EL ESPECTROGRAMA DE MEL

Fuente de las imágenes empleadas en este apartado: https://towardsdatascience.com/getting-to-know-the-mel-spectrogram-31bca3e2d9d0

Una vez explicada la necesidad del Espectrograma de Mel y de donde viene su utilidad a la hora de clasificar sonidos vamos a pasar a explicar su creación y como se logra esta representación del audio.

Los sonidos pueden ser representados de diversas formas, siendo la más simple la amplitud en el tiempo. Esto se debe a que el sonido puede definirse como una secuencia de vibraciones que varían la presión de las partículas en un medio, por lo que se suele usar la amplitud para representarlos, como se puede observar en la Figura 13.

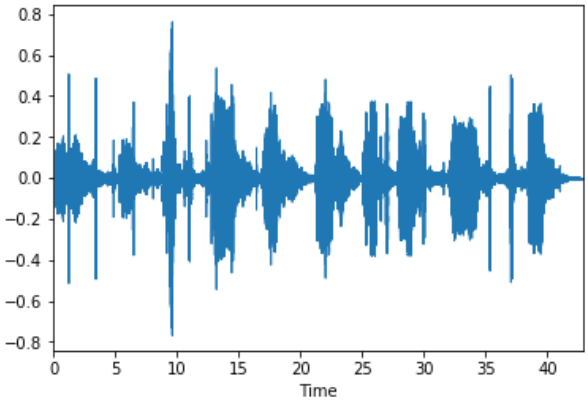



*Figura 13: Representación temporal de una onda de audio con la amplitud en el eje Y y el tiempo en el eje X*

Una manera simple de tratar la amplitud en el tiempo es usando la transformada de Fourier, la cual permite trabajar en el dominio de las frecuencias en vez del tiempo. Usando esta simple formulación, se puede representar la magnitud de onda normalizada sobre la frecuencia. Ahora bien, para simplificar el análisis se parte el grafico de amplitud sobre tiempo en pequeñas partes y se aplica la transformada de Fourier sobre cada una de ellas. Un ejemplo de una de estas representaciones se puede ver en la Figura 14

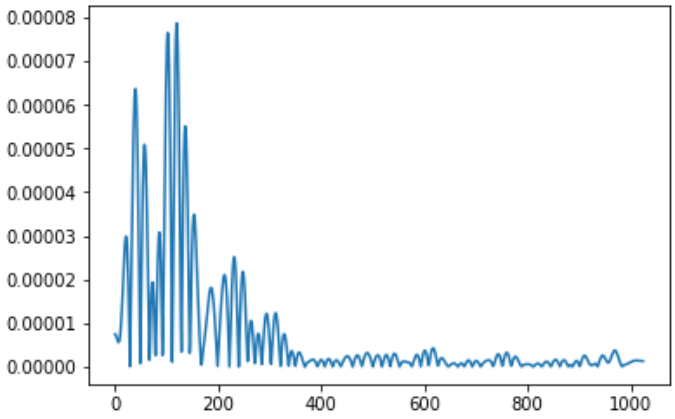



*Figura 14: Representación de un fragmento de una onda sonora usando la transformada de Fourier*

Con toda esta información se procede a representar cada parte en serie en una gráfica de frecuencia sobre tiempo, representando a su vez los dB (unidad que se utiliza para expresar la relación entre dos valores de presión sonora, o tensión y potencia eléctrica) en una escala de color. Sin embargo, la representación de estos datos en escalas normales lleva a una solución como la que se puede observar en la Figura 15, en la cual no se puede apreciar ninguna variabilidad que permita distinguir unos audios de otros, pero esto no queda aquí.

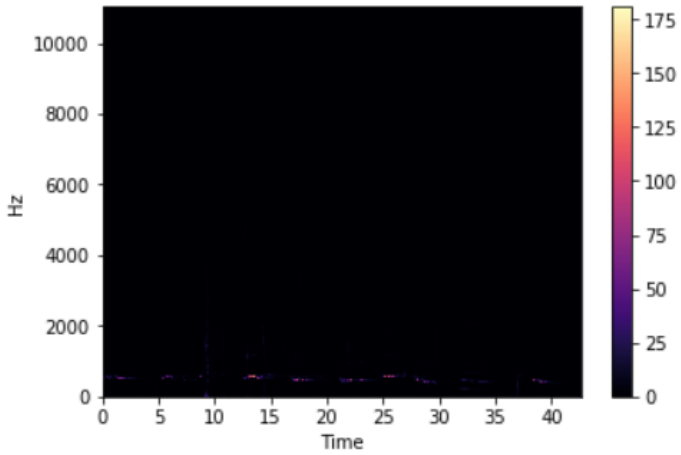



*Figura 15: Representación en decibelios del diagrama de frecuencia de una onda sonora*

El problema es que, para tener una representación en la que se vean más detalles sobre la información del sonido representado, se tiene que usar una escala logarítmica tanto en la representación de la frecuencia como la de los dB. A esto se le llama espectrograma (ver Figura 16).

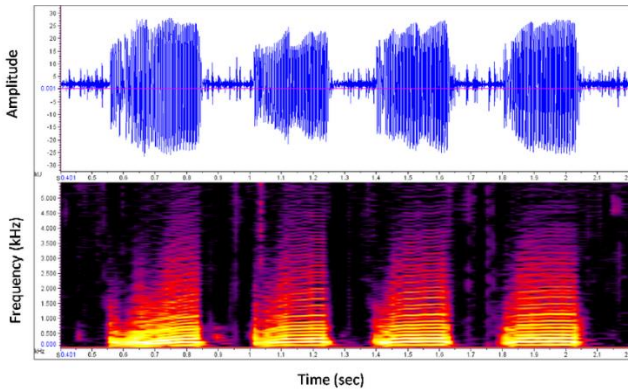



*Figura 16: Representación de una onda sonora en escala logarítmica*

El revés de esta representación es que, para altas frecuencias y altos decibelios, la diferencia en la representación es prácticamente imperceptible, por lo que le resultará más difícil a la red neuronal analizar y aprender de estos datos. Es aquí donde entra en juego la escala de Mel, la cual, desde un punto de vista matemático, es el resultado de aplicar una transformación no lineal a la escala de frecuencias, haciendo más obvias las diferencias a altas frecuencias, frente a lo que ocurría en la escala logarítmica [3].

Para aplicar este concepto, se pasa a la escala de Mel cada parte antes dividida del sonido representado en el dominio de las frecuencias y se continúa aplicando los mismos pasos que al utilizar la escala logarítmica común, dando resultado al espectrograma de Mel (ver Figura 17). El espectrograma representa la frecuencia en el Eje y en escala logarítmica, el tiempo en el Eje X y la amplitud de las frecuencias en decibelios por medio de colores.

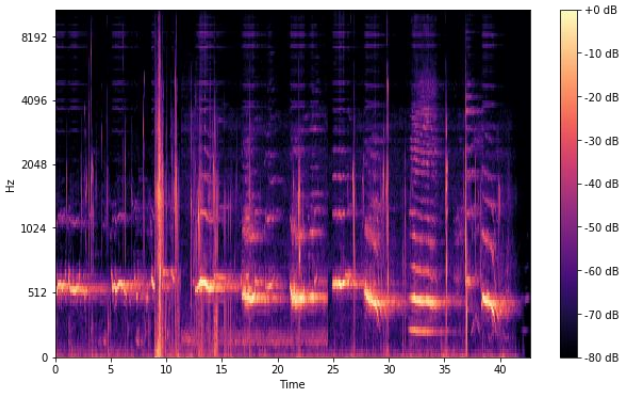



*Figura 17: Representación del espectrograma de Mel de una onda sonora*

Los espectrogramas de Mel suponen la conversión de la onda sonora en imágenes que permiten distinguir fácilmente diferencias en el contenido espectral a todas las frecuencias. Esto abre la puerta a emplear de manera eficaz técnicas de clasificación de imágenes, que es uno de los campos más robustos y en los que se han logrado mayores avances en la inteligencia artificial. Las distintas técnicas recogidas en el estado del arte, en lo que a clasificación de audio se refiere, emplean los espectrogramas de Mel como *input*. Las técnicas principales, al igual que con la clasificación de imágenes convencional, pasan por el uso de redes neuronales convolucionales [7].

Esta técnica, junto con el análisis previo del estado del arte, determinó el uso de técnicas de inteligencia artificial y se avanzó a la parte central del proyecto, donde se procedió a desarrollar y entrenar diversas arquitecturas de modelos del estado del arte basadas en el uso del espectrograma de Mel.

## 2.2 REDES CONVOLUCIONALES

Las redes convolucionales son una de las tecnologías más robustas y consolidadas dentro del campo del *machine learning* y representan el estado del arte en la resolución de problemas de clasificación o segmentación de imágenes. La idea central de este tipo de redes fue creada por el informático francés Yann Lecun y ha dado lugar a todo tipo de arquitecturas (como *inception, vgg, Alexnet…)* y una gran variedad de aplicaciones en muchos campos diferentes, entre ellas la clasificación de audio.

La característica principal de este tipo de redes es que las neuronas están formadas por "matrices" o *kernels* y los pesos que se entrenan usando el algoritmo de *backpropagation* son los valores que toman las celdas de estas estructuras. Varios *kernels* de la misma dimensión se agrupan para formar una capa convolucional, que se acompaña de una etapa conocida como *pooling* para poder comprimir la salida de estas capas, que constituirá la entrada para la siguiente etapa convolucional y así sucesivamente. Varias de estas combinaciones de convolución más *pooling* dan lugar a las capas ocultas de una red neuronal convolucional, como se puede ver en la Figura 18. En su variante más tradicional y para la mayor parte de las aplicaciones para las que se suelen emplear este tipo de redes, estas capas ocultas van seguidas de una combinación de capas correspondientes a una red neuronal densa que permite la clasificación del resultado de las capas de convolución.

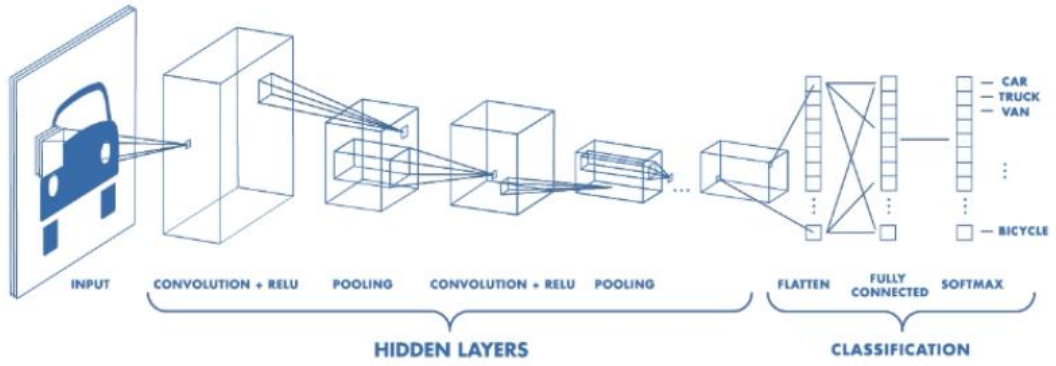



*Figura 18: Estructura de una red convolucional (Fuente [8])*

La idea principal detrás del uso de los *kernels* es que estos permiten realizar una operación matemática sobre la entrada de su capa llamada convolución. Esta operación consiste en un producto escalar que se repite a lo largo y ancho sobre los *pixeles* de toda la estructura de entrada de la capa. De esta forma se consigue comprimir y extraer ciertas características en cada etapa convolucional en función de la tarea para la que haya sido entrenada la red. Por lo general, las capas más cercanas a la entrada inicial capturan características más sencillas, mientras que las capas finales son capaces de recoger estructuras más complejas.

Una cosa importante a tener en cuenta la hora de entender las redes neuronales convolucionales y su aplicación en este proyecto es que la palabra convolucional hace referencia a la arquitectura de la red, pero no al objetivo de la red. Lo que hace referencia al objetivo de la red y el problema que trata de resolver es la función de coste. Los tipos de función de coste para los retos que se resuelven con este tipo de arquitecturas son muy variados, siendo la más común la *cross-entropy*, ya que la mayor parte de los casos de uso de estas arquitecturas son de clasificación y esta función es la referencia en este campo. Otras funciones también utilizadas con estas arquitecturas son el Error Mínimo Cuadrado, para problemas de regresión, la divergencia de Jensen-Shannon para GANs, el "*Expected*

*discounted accumulated future reward*" para problemas de *Reinforcement Learning* o *Triplet Loss* para técnicas de aprendizaje no supervisado.

A continuación, se explican las funciones de perdida de *Cross-Entropy Loss* y *Triplet Loss* en profundidad, que son las más relevantes para el contexto de este proyecto.

### 2.2.1 CROSS-ENTROPY LOSS

Referencias de este apartado: [9]

La idea detrás de las funciones de perdida es la de poder medir y cuantificar la salida de un modelo con respecto al objetivo o resultado esperado por el mismo. Las funciones de perdida son la base del aprendizaje supervisado. La creatividad en el diseño de estas funciones y en su uso ha dado lugar a variantes para los problemas de aprendizaje no supervisado y aprendizaje por refuerzo.

La función *cross-entropy los* (entropía cruzada) es la función de referencia en el campo de los problemas de clasificación y por tanto la más utilizada en las aplicaciones que atañen a los modelos con redes neuronales convolucionales. La entropía cruzada es un concepto muy utilizado en el campo de la probabilidad para medir la diferencia entre dos distribuciones de probabilidad o un grupo de eventos aleatorios. En el campo de *machine learning* y de los modelos de clasificación se utiliza para medir la diferencia entre el vector de probabilidades que el modelo otorga para una entrada dada, en el que cada posición representa la probabilidad que el modelo otorga a una clase de salida, y el verdadero vector con la clase a la que pertenece dicha muestra de entrada. Sirve en resumen para medir la precisión de los modelos que tiene una salida para las clases con valores entre 0 y 1.

En función del número de clases de salida para el que se calcule esta entropía cruzada se puede hablar de dos tipos de entropía cruzada:

- *Binary cross-entropy:* si el número de clases de salida es dos, generalmente esto suelen ser problemas de detección. Por ejemplo, detectar si hay un gato en una imagen.
- *Categorical cross entropy:* si el número de clases de salida es mayor de dos, esto son los problemas clásicos de clasificación. Un buen ejemplo de esto es el problema que se aborda en este proyecto, la clasificación de audios en diferentes clases.

La entropía cruzada calcula la diferencia entre el vector de probabilidades y el vector objetivo usando la siguiente ecuación:

$$Loss = -\sum_{i=1}^{\substack{longitud\ de \\ salida}} y_i * \log \hat{y}_i$$

Donde:

- $\hat{y}_i$ es el valor de salida que el modelo ha otorgado a la clase i para una muestra dad
- $y_i$es el valor de salida esperado para la clase i (generalmente 0 o 1) para una muestra dada.

Esta ecuación da un valor que es el que posteriormente se emplea para el algoritmo de *backpropagation* que se emplea para entrenar los parámetros de la red neuronal.

### 2.2.2 TRIPLET LOSS

Referencias de este apartado: [10]

*Triplet Loss* o pérdida de triplete es una función de coste que se emplea en problemas de aprendizaje no supervisado y que se emplea en el entrenamiento de redes neuronales que buscan crear un espacio vectorial de aprendizaje propio con una cantidad de dimensiones reducida y que supone una especia de mapeado entre una entrada y su salida correspondiente en este espacio. El ejemplo más reciente en la creación de este tipo de espacios es el modelo

GPT3, que como paso previo a las redes *transformer* que constituyen la base de su tecnología mapea las palabras de entrada a un espacio vectorial propio.

La técnica de pérdida del triplete busca minimizar la distancia euclídea entre las salidas correspondientes a dos muestras similares, dos audios de la misma clase en el caso de este proyecto, al tiempo que busca aumentar la distancia euclídea con respecto al vector salida correspondiente a una muestra de una clase diferente. De esta manera se entrena a la red no solo para que realice un proceso de *clustering* con las muestras similares, sino que también está entrenada para separar lo máximo posible las muestras distintas, construyendo un espacio propio de aprendizaje.

El error que se propaga por la red neuronal para su entrenamiento se obtiene de la siguiente ecuación:

$$L = max(d(a,p) - d(a,n) + margen, 0)$$

En esta ecuación "a" es el ancla o vector normalizado que se ha obtenido de la red al introducir una muestra que se va a usar para comparar con el vector de entrada; "p" se corresponde con el vector normalizado obtenido de otra muestra de la persona de la imagen del ancla y "n" representa el vector normalizado obtenido con una muestra que se sabe que no se corresponde con la muestra del ancla. "d (a, p)" y "d (a, n)"se corresponden con las distancias euclídeas entre el ancla y el positivo y el ancla y el negativo respectivamente. El margen se fija y se pone para evitar que se cometa sobre aprendizaje por parte de la red y que el proceso no tienda a hacer que la distancia entre el ancla y el positivo se iguale a la distancia entre el ancla y el negativo.

El funcionamiento se representa de forma esquemática en la Figura 19.

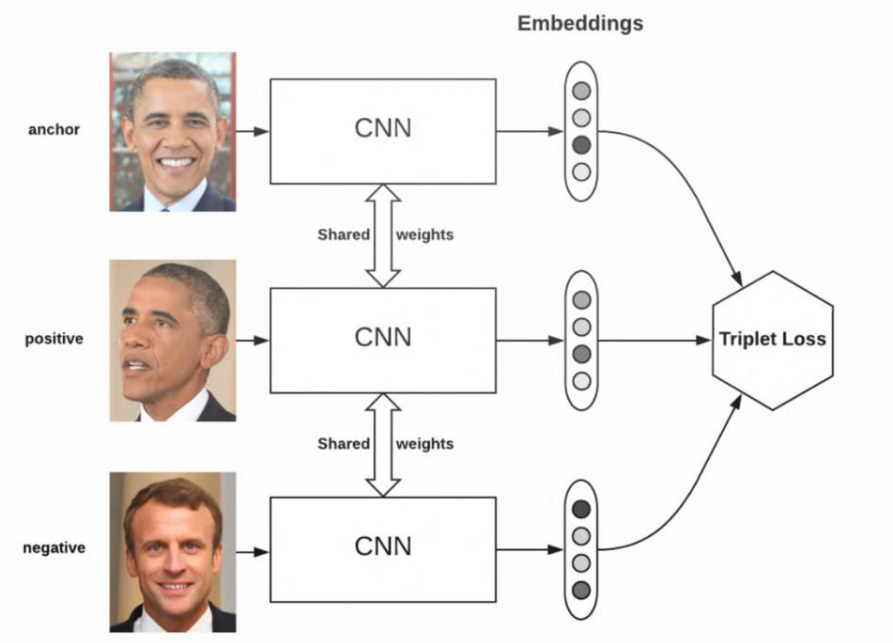



*Figura 19: Ejemplo de funcionamiento de la técnica de Perdida del Triplete para una red convolucional de reconocimiento facial.*

Esta técnica no se ha empleado como tal en el trabajo, pero se incluye, puesto que se empleó un modelo (*vggish)* que resulta de hacer *tranfer learning* con una arquitectura de una red neuronal y se emplea el uso de un espacio de aprendizaje artificial creado por esa arquitectura. Esta red tiene como salida un vector de 128 elementos para cada muestra de audio que recibe como *input* y ha sido originalmente entrenada utilizando *categorical cross-entropy* como función de perdida, ya que contaba con su propio clasificador. Si se quisiera re entrenar solo esta red para obtener mejores resultados en un problema concreto, lo que se conoce como *fine tuning*, sería necesario emplear la función de *triplet loss*, lo cual resulta muy interesante desde el punto de vista de mejoras para el modelo.

## 2.3 TRANSFER LEARNING

*Transfer learning* es una técnica empleada en *machine learning* en la cual un modelo entrenado y usado para una tarea es reutilizado como punto de partida para otro modelo desarrollado para otra tarea distinta.

Haciendo una comparación con la manera de aprender que tenemos las personas, sería como si cogiéramos las habilidades de una persona que sabe jugar al tenis y se las pusiéramos a una persona que queremos que aprenda a jugar al ping-pong y que no ha practicado ningún deporte jamás. De esta forma se conseguiría que esta persona necesite mucho menos entrenamiento para adquirir las habilidades necesarias para poder realizar la segunda tarea de manera satisfactoria.

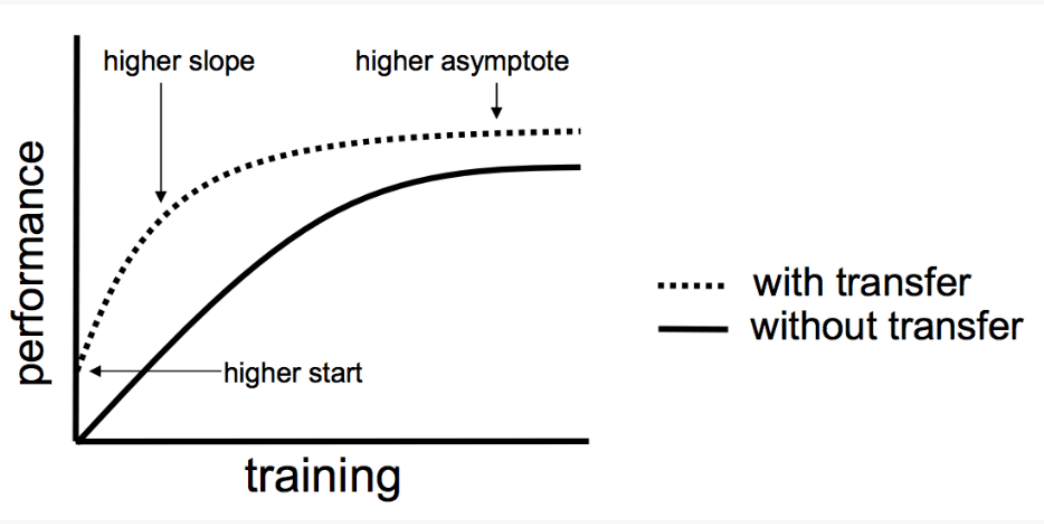



*Figura 20: Representación gráfica de las ventajas del uso de Transfer Learning (fuente [11])*

Para entender la utilidad del *transfer learning* resulta muy interesante fijarse en el ejemplo de este proyecto. Para este proyecto se ha utilizado *vggish*, que es una modificación de la famosa arquitectura para clasificación de imágenes *vgg19* con ligeras modificaciones en el

tamaño de salida y una reducción del número de capas utilizadas. Esta arquitectura ha sido entrenada por investigadores de *Google* en el dataset Youtube 8M., que contiene millones de segmentos de audio correspondientes a 1000 clases diferentes extraídos de videos de *youtube*. Entrenar un modelo de estas características en un *dataset* tan grande es el sueño de cualquier ingeniero de *machine learning*, sin embargo, hay que tener en cuenta que es una tarea computacionalmente muy costosa, que requiere de un *hardware* muy potente y, aun así, los modelos requieren de muchas horas de entrenamiento. Esto queda evidenciado en el trabajo [5], donde se entrenan diferentes modelos, entre ellos *vgg* en *datasets* similares al de el modelo *vggish* y utilizando para este proceso entre 10 y 20 GPUs y más de 400 horas de entrenamiento por modelo.

Son este tipo de casos los que convierten al *transfer learning* en una herramienta tan potente, ya que permiten a proyectos como el que se presenta en este trabajo, emplear modelos con resultados a nivel del estado del arte y poder aprovechar estos recursos para generar otro tipo de arquitecturas y soluciones que también obtengan grandes resultados.

El modelo *vggish* lo que permite es utilizar la arquitectura de las capas ocultas y convolucionales del modelo mostrado en la Figura 18 correspondientes a las pre entrenadas por los investigadores de *Google* en *datasets* y con *hardware* que no están al alcance del público en general. Permitiendo de esta manera tener una base sobre la que entrenar otros modelos o usar otras técnicas para resolver problemas similares a los de la arquitectura que se usa para hacer *transfer learning.*

## 2.4 TÉCNICAS PARA ENTRENAMIENTO DE MODELOS

En este capítulo se describirán brevemente algunas de las técnicas utilizadas en el entrenamiento de los modelos de *deep learning* de este proyecto, ya que su explicación ayudara en gran medida al entendimiento del porqué de su uso y que beneficios se han obtenido.

### 2.4.1 DROPOUT

El *dropout* es una técnica de regularización para redes neuronales desarrollada en el laboratorio de Geoffrey Hinton en 2012 [6]. Consiste en desactivar de manera aleatoria ciertas conexiones o neuronas de las capas ocultas de la red. Con esto se consigue modular la potencia para aproximar de la red, lo que lleva de manera eficaz a que no se sobreentrene (i.e., que no se produzca el famoso *overfitting).* En la Figura 21 se puede ver un ejemplo visual del funcionamiento de esta técnica en una red neuronal densa.

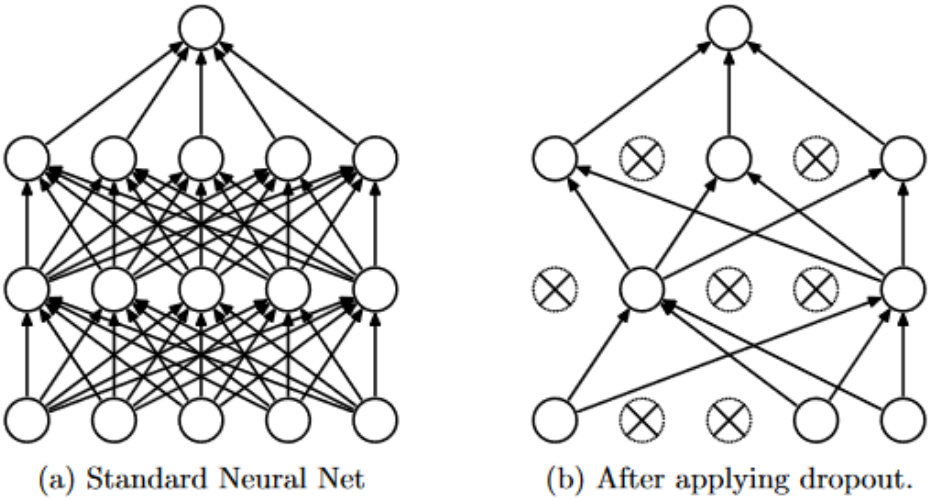



*Figura 21: Ejemplo del funcionamiento de la técnica dropout (fuente [12])*

### 2.4.2 BATCHNORMALIZATION

La normalización por lotes, o como se la conoce más comúnmente en inglés *Batchnormalization,* es una técnica que consiste en añadir una etapa extra entre las neuronas y la función de activación, con el objetivo de normalizar las activaciones de salida. Esta técnica de regularización es una ayuda al entrenamiento. Esta técnica se suelte combinar con la idea de un momento, de manera que la media y desviación estándar que se usen para

normalizar una muestra, no sean muy diferentes de las utilizadas para normalizar la muestra anterior.

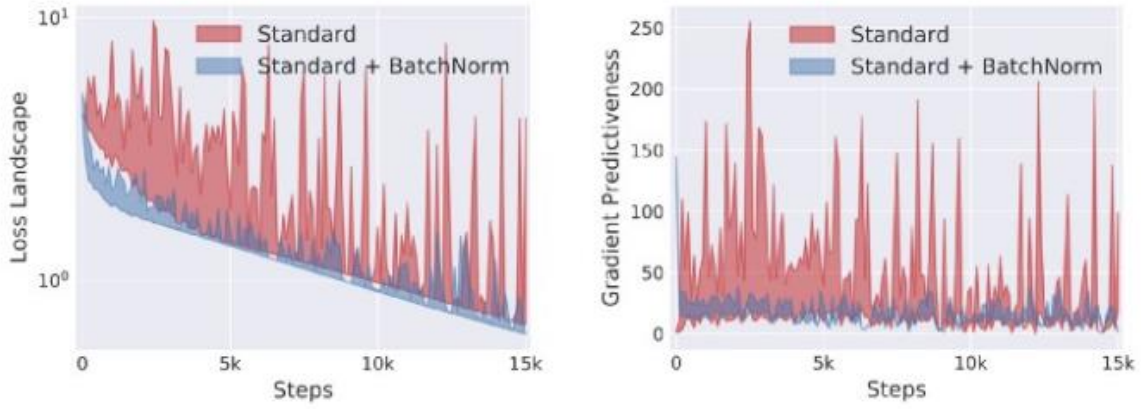



*Figura 22: Muestra de los ejemplos del Batchnormalization (fuente [13])*

### 2.4.3 EARLY STOPPING

Esta técnica se usó durante el proceso de entrenamiento para evitar el *overfitting.* Consiste en establecer un umbral de épocas para alguno de los KPIs del entrenamiento (*test accuracy, test loss…)* de manera que, si ese KPI lleva más de las épocas establecidas sin mejorar su valor, se pare el entrenamiento y se escoja como modelo final el existente para la 'última época que ese KPI mejoró.

En la Figura 23 se puede ver un ejemplo de uno de los entrenamientos finales del modelo 2DCNN, donde se ve como tras 3 épocas sin mejorar el parámetro *val_loss* que es la función de coste en el set de validación, se procede a terminar el entrenamiento y quedarse con la versión de la época 17.

```
Epoch 00017: val_loss improved from 0.38043 to 0.36380, saving model to ../Models_Checkpoint/Conv2D_1.h5
1446/1446 [==============================] - 33s 22ms/step - loss: 0.1884 - acc: 0.8981 - val_loss: 0.3638 - val_acc: 0.8721
Epoch 18/30
1445/1446 [============================>.] - ETA: 0s - loss: 0.1776 - acc: 0.9012
Epoch 00018: val_loss did not improve from 0.36380
1446/1446 [==============================] - 33s 22ms/step - loss: 0.1775 - acc: 0.9012 - val_loss: 0.5090 - val_acc: 0.8193
Epoch 19/30
1445/1446 [============================>.] - ETA: 0s - loss: 0.1714 - acc: 0.9050
Epoch 00019: val_loss did not improve from 0.36380
1446/1446 [==============================] - 33s 23ms/step - loss: 0.1714 - acc: 0.9050 - val_loss: 0.3782 - val_acc: 0.8689
Epoch 20/30
1446/1446 [==============================] - ETA: 0s - loss: 0.1698 - acc: 0.9083
Epoch 00020: val_loss did not improve from 0.36380
1446/1446 [==============================] - 33s 23ms/step - loss: 0.1698 - acc: 0.9083 - val_loss: 0.4158 - val_acc: 0.8523
Epoch 21/30
1444/1446 [============================>.] - ETA: 0s - loss: 0.1617 - acc: 0.9127
Epoch 00021: val_loss did not improve from 0.36380
1446/1446 [==============================] - 33s 23ms/step - loss: 0.1616 - acc: 0.9128 - val_loss: 0.3779 - val_acc: 0.8754
dict_keys(['loss', 'acc', 'val_loss', 'val_acc'])
```

*Figura 23: Ejemplo de la actuación del early stopping durante uno de los entrenamientos de este proyecto (Elaboracion propia)*

### 2.4.4 CLASS WEIGHT

Esta técnica se usa para reducir el posible efecto que puede tener sobre nuestro modelo el hecho de tener unos datos fuertemente desbalanceados entre clases. Esto es exactamente el caso de este proyecto, como se puede ver en la Figura 26 y la Figura 27. La idea principal de *class weight* es la de otorgar pesos a las predicciones de las clases, de manera que los errores en aquellas clases que tienen menor cantidad de datos en el set de entrenamiento penalicen más (proporcionalmente) que los errores en aquellas clases que tienen mayor cantidad de datos.

# Capítulo 3. MEMORIA DEL PROYECTO

## 3.1 ESTRUCTURA Y DEFINICIÓN DEL PROYECTO

Al tratarse de un proyecto que se enmarca en el ámbito del *Machine Learning* (ML) y la Inteligencia Artificial, se siguió la estructura típica de este tipo de proyectos. Las fases y el orden de trabajo para seguir en estos proyectos son los siguientes:

1. Análisis del problema
    a. Determinar el tipo de solución/es de ML
2. Recopilación de datos
3. Preparación y análisis de los datos
    a. Guardado y limpieza
    b. Transformación
    c. Preprocesamiento
4. Modelado de algoritmos y arquitecturas
    a. Selección de arquitecturas más adecuadas
    b. Construcción de los modelos
5. Entrenamiento de modelos
    a. Entrenamiento
    b. Evaluación de resultados
6. Selección de soluciones
    a. Comparación de resultados
    b. Elección de modelos

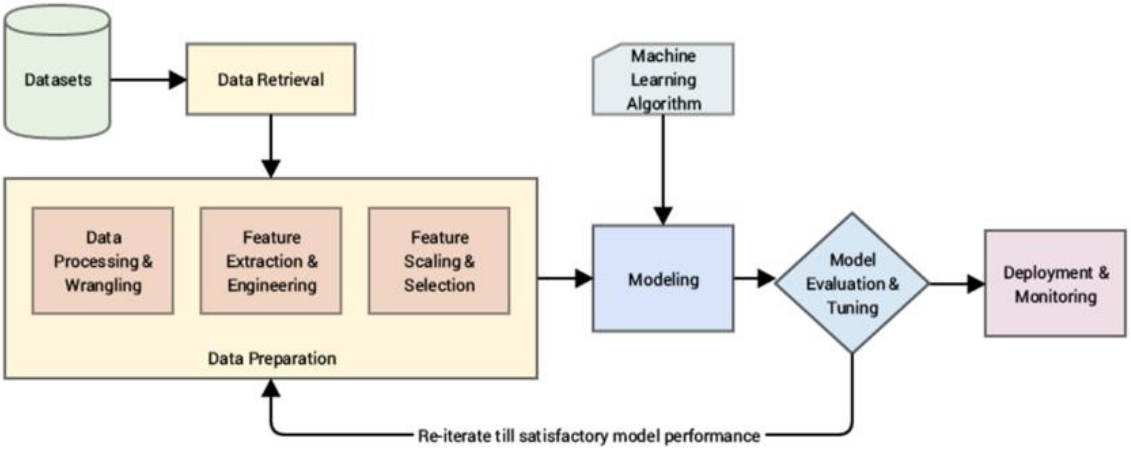



*Figura 24: Estructura de un proyecto clásico de ML*

A continuación, se expone el trabajo realizado en cada uno de las fases de un proyecto de ML en este proyecto concreto.

## 3.2 ANÁLISIS DEL PROBLEMA

En esta etapa se determinó, junto con los *stakeholders* del proyecto, que el objetivo final de este fuera la clasificación de fragmentos de audio para la detección de anomalías de seguridad o confort en los mismos.

Uno de los grandes errores que se producen en el campo de Aprendizaje Automático es la aplicación de soluciones de ML o Inteligencia Artificial a problemas que realmente no requieren de este tipo de alternativas y que podrían ser resueltos con mayor facilidad, en menos tiempo y de manera más eficiente con otro tipo de algoritmos que no pertenezcan al campo del aprendizaje automático. Es por ello se valoró cuidadosamente el empleo de todas las posibles técnicas, para no caer en este error y aplicar una solución desmesurada para el problema y los requisitos del proyecto.

Teniendo en cuenta los requerimientos, y atendiendo al estado del arte, se puso de manifiesto la conveniencia de usar una aplicación de ML, ya que son los algoritmos que mejores

resultados han presentado en este tipo de problemas de detección y clasificación de audio, como se menciona en la revisión del Estado del Arte. Se irá viendo a lo largo del proyecto como este tipo de soluciones se van a adaptando y cobrando más sentido a medida que el avanza y se exponen sus dificultades y retos.

### 3.2.1 SELECCIÓN DEL TIPO DE SOLUCIÓN

Tras el análisis del estado del arte y ante el éxito de las soluciones de redes neuronales convoluciones para afrontar este tipo de problemas se optó por el estudio del comportamiento de dos arquitecturas relacionadas con este tipo de redes- y que representan el estado del arte para diferentes *datasets* a través de los estudios [1] y [2]. En primer lugar, una arquitectura de *encoder* convolucional pre entrenado junto con un clasificador y, en segundo lugar, la creación y entrenamiento de una red neuronal convolucional *ad hoc* en la que todos los parámetros eran configurables.

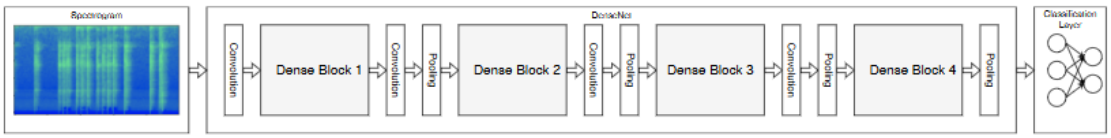



*Figura 25: Estructura general de las arquitecturas del proyecto (Fuente: [1])*

La Figura 25 representa el estilo de red que se va a utilizar en las dos arquitecturas propuestas para el proyecto. Las 3 etapas clave de la imagen y que se aplican en ambos modelos son:

- Se recibe como input un audio convertido a espectrograma y que por tanto es una imagen.
- A continuación, se procede a pasar los espectrogramas por una serie de etapas convolucionales que extraen las características de los mismos. Esta etapa es la mayor diferencia entre ambos modelos, siendo para la arquitectura del modelo 2DCNN totalmente configurable y con la capacidad de añadir o quitar etapas, y para la

arquitectura del Encoder + el Clasificador, se trata de una red convolucional pre entrenada, basada en una variante de las arquitecturas *vgg* para clasificación de imágenes.

- Por último, una etapa de clasificación que se encarga de decidir a cuál de las clases seleccionas corresponde la muestra del espectrograma que ha sido el *input* de la red. Para el modelo 2DCNN esta etapa es una continuación de la propia red, mientras que en la otra arquitectura es una red propia y creada para clasificar los vectores de de características que son el *output* de la red *vggish.*

## 3.3 RECOPILACIÓN DE DATOS

En esta etapa se llevó a cabo la recopilación de datos que permitieran el entrenamiento de las soluciones de ML seleccionadas. Dentro del contexto de un proyecto de ML esta suele ser una de las etapas más tediosas y que determinan en gran medida el alcance y resultado final del proyecto. En concreto, para este trabajo la recopilación de datos marcó el tipo y número de clases que se emplearán más tarde para el entrenamiento y creación de los modelos de Inteligencia Artificial.

Los proyectos de ML requieren de grandes cantidades de datos para obtener resultados satisfactorios y en concreto en proyectos de clasificación se requieren grandes cantidades de datos etiquetados correctamente, lo cual suele ser más complejo de encontrar y lleva en muchos casos a emplear complejas técnicas de preprocesamiento e ingeniería de datos para conseguir sacar el mayor partido de los datos disponibles, como ocurrió en este proyecto.

Estos audios se recogieron de diferentes fuentes entre las que destacan:

- El conjunto de datos de UrbanSound (Link: UrbanSound Dataset).
- El conjunto de datos Freesound (Link: Freesound Dataset)
- Clips del sitio web de la BBC (Link: BBC Dataset)

- Audios extraídos de videos de Youtube: estos audios se extrajeron una vez determinadas las clases de interés para proyecto y para aumentar el número de muestras de entrenamiento de clases con menor número de muestras.

La recopilación de los datos se hizo de manera ajena al contexto de la seguridad, ya que los datos en este ámbito están muy restringidos, y por otra parte esto no resultaba una limitación para el objetivo final del proyecto. Es decir, no se hacía necesario que los datos provinieran directamente de una situación en la que la seguridad se viera comprometida, ya que, por ejemplo, un ladrido de perro en cualquier otro contexto serviría para entrenar el modelo para situaciones en las que un perro esté ladrando debido a que un asaltante ha entrado en una propiedad privada.

No todos los datos se pudieron emplear directamente, sino que, como se mencionara en los puntos siguientes, la mayor parte fueron sometidos a distintos procesos de limpieza, etiquetado y preprocesamiento, para poder adecuarlos a las necesidades del proyecto. Muchos de los audios de los *Datasets* de *Freesound* y BBC eran algo ambiguos y tuvieron que ser clasificados manualmente para eliminarlos o recortarlos y de esta forma obtener mejores resultados.

Tras un primer acercamiento a los datos disponibles, se hizo una hipótesis de las posibles clases que se podrían extraer de estos datos y que serían de utilidad al proyecto. Las clases que se decidieron emplear en primera instancia y a raíz de los datos descargados fueron: Cristales rotos, Lloros, Ladridos de perro, Fuego, Disparos, Portazos, Pasos. Esta selección de clases se hizo en base a aquellos sonidos de los que se disponía mayor cantidad de audios y que guardaban una relación con aspectos relacionados con la seguridad o el confort de los clientes de equipos de seguridad.

Cabe destacar que estas clases no son las definitivas, sino que se emplearon como guía sobre la que ir trabajando en las siguientes etapas y poder ir avanzando el proyecto.

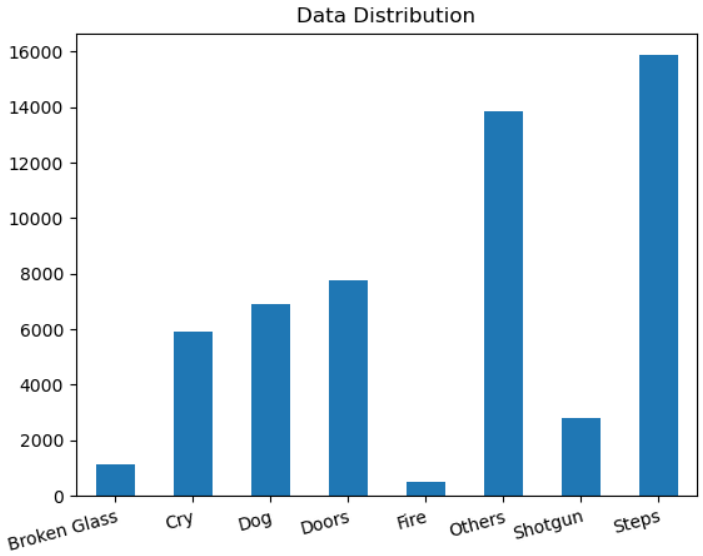



*Figura 26: Distribución inicial de los datos obtenidos para entrenar el modelo*

En la Figura 26 se muestra la distribución de las distintas clases que se emplearon originalmente para el proyecto y sobre las que se hizo el primer análisis de las arquitecturas propuestas. Unos de los principales aspectos a destacar es la gran diferencia en cuanto a número de muestras para cada una de las clases, se comentará más adelante una técnica que se empleó para lidiar con este hecho durante el entrenamiento y otorgar la misma importancia a todas las clases independientemente de la cantidad de muestras. Esta técnica se conoce como *class_weight* y consiste, en resumen, en ponderar en mayor medida los errores que el modelo pueda tener en aquellas clases que tienen menos muestras.

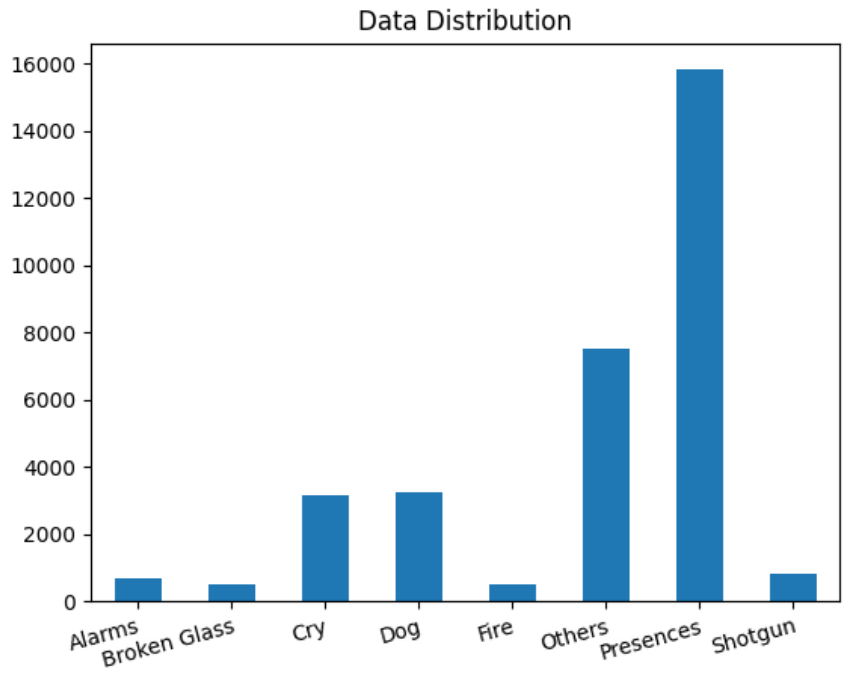



*Figura 27: Distribución final de las clases empleadas para el proyecto (elaboración propia)*

En la Figura 27 se muestra la distribución final y tipo de clases que se emplearon para la solución definitiva del proyecto una vez se hubieron llevado a cabo todos los cambios y procesos que se detallaran más adelante en esta memoria.

### 3.3.1 EXPLICACIÓN DE LAS CLASES

Las clases iniciales para el proyecto se seleccionaron en base a un *trade-off* entre dos criterios:

- Utilidad en el contexto de la seguridad: en base al criterio de los expertos para poder identificar situaciones de robo o que puedan comprometer la seguridad del cliente.
- Calidad y disponibilidad de datos de la clase: este aspecto hace referencia a la calidad de los datos encontrados en diferentes data sets para las clases propuestas.

Atendiendo a estos dos criterios se establecieron las siguientes clases:

1. **Cristales rotos** (*Broken Glass):* esta clase hace referencia a sonidos que tengan que ver con cualquier tipo de cristal rompiéndose, desde ventanas hasta copas o vasos. Esta clase serviría para detectar posibles intrusiones rompiendo cristales u rotura de objetos cuando el cliente final no se encuentre en la residencia.
2. **Lloros** (C*ry*): esta clase se incluyó como iniciativa de la adición de la variable *confort* al sistema. Su uso está pensado para sustituir a las clásicas soluciones de monitorización de bebes.
3. **Ladrido de perro** (*Dog*): clase para detectar ladridos de perros. Su utilidad reside en la detección del animal puede hacer de ciertos intrusos cuando el dueño no se encuentre en el domicilio.
4. **Puertas** *(Doors)*: hace referencia a cualquier sonido de puertas, cajones … abriéndose y cerrándose. Sirve para detectar la presencia de gente en el lugar de las cámaras cuando no debería haber nadie en el lugar.
5. **Fuego** (*Fire*): clase que permite detectar el clásico chisporroteo de llamas que hace el fuego. Al igual que lloros, se incluyó como iniciativa para la variable *confort* del modelo. Esta clase se pensó como complemento o sustitutivo para las alarmas antiincendios y se pensó mejorarla añadiéndole explosiones o sonidos que puedan causar otras situaciones de peligro relacionadas con el fuego.
6. **Otros** (*Others*): esta clase hace referencia a todos aquellos sonidos que no tengan nada que ver con situaciones que comprometan la seguridad, pero que puedan ser captadas por el sistema. Se incluyo como clase 0 y ayuda a dar robustez al sistema. Está compuesta principalmente por sonidos urbanos (ambulancias, obras…), sonidos de naturaleza y silencios.
7. **Disparos** (*Shotgun*): esta clase hace referencia a disparos de armas de fuego. Serviría para alertar de incidencias relacionadas con pistolas u otro tipo de armas. Aunque esta clase pueda parecer sorprendente en el contexto español, hay que tener en cuenta que este sistema puede ser implementado en otros países o regiones donde el uso de

armas está permitido y no es tan extraño, como pueden ser Estados Unidos o Latinoamérica.

8. **Pasos** (*Steps*): hace referencia a sonidos relacionados con el caminar de una persona y, al igual que Puertas, sirve para detectar la presencia de gente en el lugar de las cámaras cuando no debería haber nadie en el lugar.

Sobre las clases iniciales se realizaron los siguientes cambios a lo largo del proyecto:

- Las clases Pasos y Puertas se combinaron para formar una sola clase, **Presencia**, ya que eran las clases que tenían peores KPIs para todas las arquitecturas. Esto es bastante lógico ya que son clases que pueden resultar muy parecidas también al oído humano. Puesto que ambas clases tenían la misma función final de cara a los requerimientos propuestos por los *stakeholders*, este cambio no supuso ninguna modificación de los mismos y permitió alcanzar el mismo resultado en la solución final.
- Se añadió la clase **Alarmas**, esta clase representaba todo aquello relacionado con ruidos de alarmas que se puedan dar en la vivienda, como puede ser la alarma de dejarse un fuego encendido en la vitrocerámica. Esta clase se englobaría dentro del apartado confort de la solución.

Las distribuciones de las clases se pueden ver en la Figura 26 y en la Figura 27.

## 3.4 PREPARACIÓN Y ANÁLISIS DE LOS DATOS

Una vez se hubieron recopilado los datos para la consecución del proyecto se procedió a su limpieza, clasificación y pre procesamiento para facilitar el trabajo en las etapas de entrenamiento y desarrollo de los modelos y arquitecturas de ML que representan el punto central del proyecto. Esta parte se dividió en las siguientes etapas:

### 3.4.1 GUARDADO Y LIMPIEZA

Los audios se guardaron en formato .wav, ya que algunas librerías que se emplearon para su estudio y empleo posterior funcionan mejor con este formato. Es por eso por lo que uno de los primeros pasos que se llevó a cabo fue la implementación de un *script* para convertir los archivos de audio de formato .mp3 a .wav.

La mayoría de los audios extraídos de las fuentes mencionadas en los apartados anteriores se clasificaron en función de su etiqueta de origen en carpetas con el nombre de las clases que se emplearán durante el proyecto. De esta forma se realizó una primera etapa de clasificación para adaptar los audios a las necesidades y requerimientos establecidos en las fases previas del proyecto.

### 3.4.2 PREPROCESAMIENTOS LLEVADOS A CABO

En este apartado se llevaron a cabo dos grandes tipos de preprocesamientos previos a la fase de entrenamiento de los modelos elegidos, uno para cada una de las arquitecturas.

- Modelo 2DCNN: se llevó a cabo la conversión de los audios en sus correspondientes espectrogramas logarítmicos de Mel, que son el input que requiere la red neuronal.
- Modelo *Encoder* + Clasificador (E + C): a la conversión llevada a cabo para el modelo 2DCNN para conseguir los espectrogramas de cada uno de los audios, se le añadió la conversión de estos espectrogramas por el *Encoder* convolucional pre entrenado *vggish.* De esta forma, se obtuvieron los vectores característicos de 128 elementos correspondientes a cada una de las muestras para el entrenamiento del clasificador. Estos vectores se almacenaron en archivos con el nombre de la clase a la que corresponden.

Estas dos iniciativas permitieron ahorrar un gran tiempo en los procesos posteriores de entrenamiento y prototipado de las arquitecturas, ya que especialmente la conversión de las muestras de audio en vectores en un espacio euclídeo de 128 elementos usando la red *vggish*

es un proceso que conlleva un gran consumo de recursos computacionales y tiempo. Por lo tanto, el hecho de tener que hacerlo una sola vez es una gran ventaja a la hora de encarar las etapas posteriores. A continuación, se explican ambas iniciativas en detalle.

#### *3.4.2.1 Generación de los Espectrogramas*

Antes de entrar en cualquiera de los dos modelos los audios son sometidos a un proceso previo en el que son convertidos al formato óptimo para su entrada en las arquitecturas. Este proceso consta de las siguientes etapas:

1. Transformación del input de audio de formato .wav a su correspondiente espectrograma de Mel con los siguientes parámetros para la transformación:
    - Frecuencia de muestreo: 44100 Hz
    - Duración del audio: 960 ms (fijado por los tamaños por la arquitectura pre entrenada de Vggish)
    - Duración de la ventana de la transformada de Fourier: 25 ms
    - Salto de la ventana móvil para la transformada de Fourier: 10 ms
    - Numero de *mels* en los que separar el espacio de la frecuencia: 64

Los resultados de convertir una onda de audio en espectrograma logarítmico de Mel para varios de los audios del proyecto se pueden ver en el apartado 8.1.

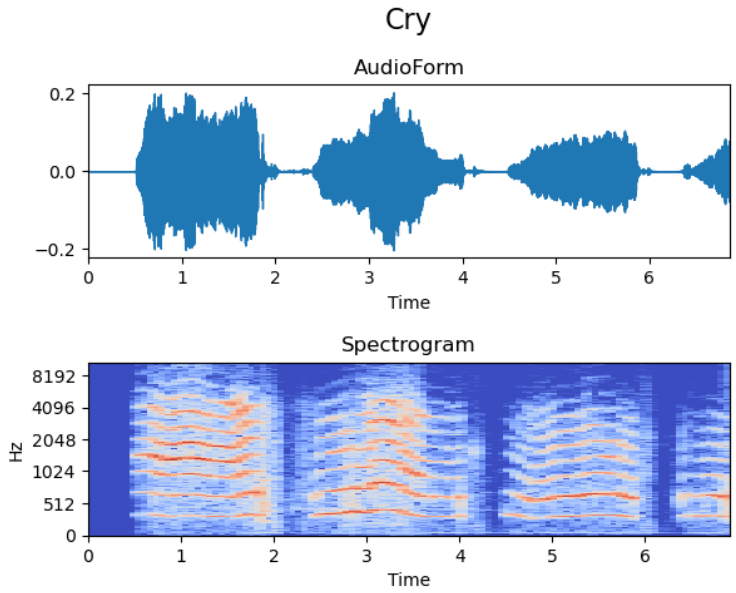



*Figura 28: Ejemplo de onda del audio del lloro de un bebe convertido a espectrograma logarítmico de Mel (elaboración propia)*

2. Segmentación de estos espectrogramas en imágenes del formato adecuado para cada una de las arquitecturas (96 x 64 pixeles). Esto se traduce en fragmentos de audio de 960 ms, ya que el salto de la ventana móvil usada para el Espectrograma de Mel es de 10ms (960 ms / 10 ms/pixel = 96 pixeles). Estos fragmentos se tomaron con un desplazamiento de 480 ms para ir superponiendo diferentes tramos y evitar dejar zonas que permitieran reconocer un audio cortadas y difíciles de clasificar para los modelos.

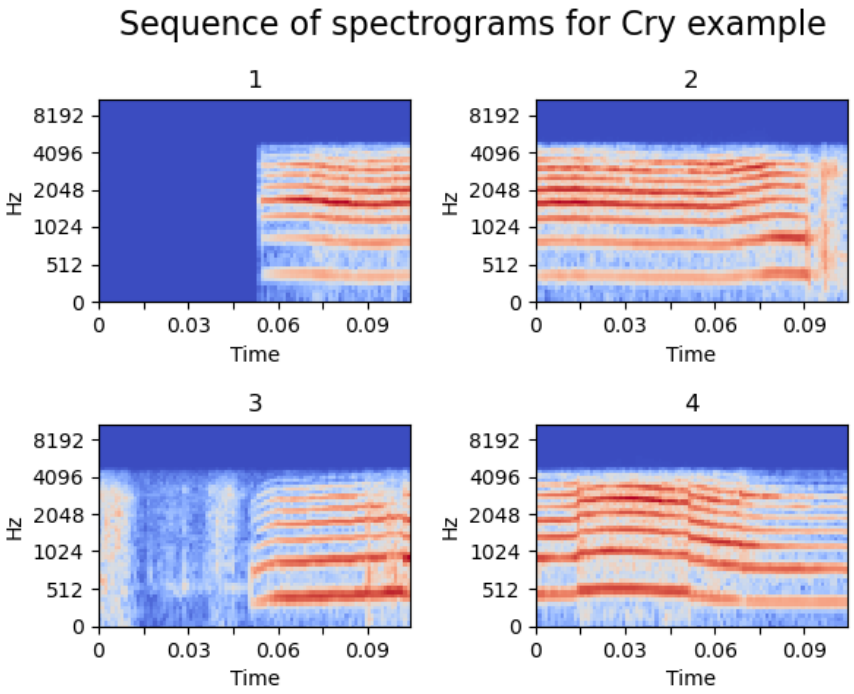



*Figura 29: Ejemplo de las 4 primeras muestras obtenidas del audio de la Figura 28*

Este proceso permitió generar cada una de las muestras necesarias para el entrenamiento en el formato adecuado para que el proceso fuera lo más rápido y eficiente posible. En la Figura 29 se puede ver un ejemplo del tipo de muestras que se obtuvieron para cada fragmento de audio.

### 3.4.3 LIMPIEZA DE LAS CLASES

Este otro pre procesamiento se incluyó a raíz de análisis posteriores y problemas detectados en el entrenamiento, es decir, como parte de la realimentación llevada a cabo en el proceso de entrenamiento (ver Figura 24). Este pre procesamiento no fue detectado al poner el proceso en marcha y se hizo palpable al recibir los primeros resultados en las fases de entrenamiento.

Este error venía derivado de un incorrecto etiquetado de los datos y se debió al uso de ventanas móviles de 960 ms como input para los modelos y al hecho de que el sonido característico y que daba nombre a la etiqueta del audio no se producía en todas estas ventanas y podía haber algunas que simplemente fueran un silencio o cualquier ruido de fondo que no se correspondía con la clase a la que había sido etiquetada, si no con otra de las clases disponibles. Un ejemplo de esto se muestra en la Figura 30, donde se aprecia como las muestras generadas entre los segundos 3,5 y 7 de esa muestra puerta, van a ser en realidad silencios, y no van a tener ningún ruido característico de una puerta.

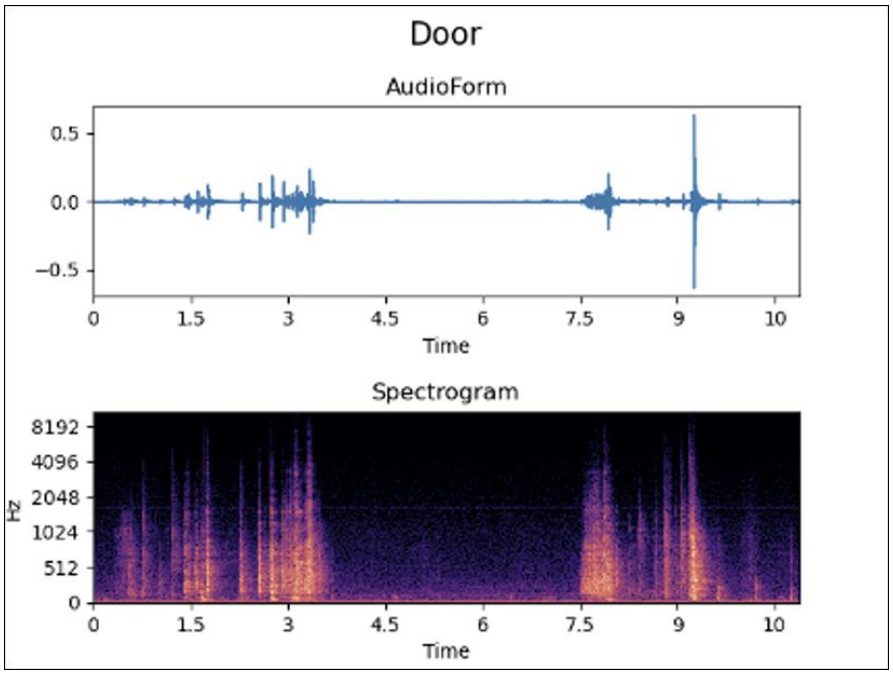
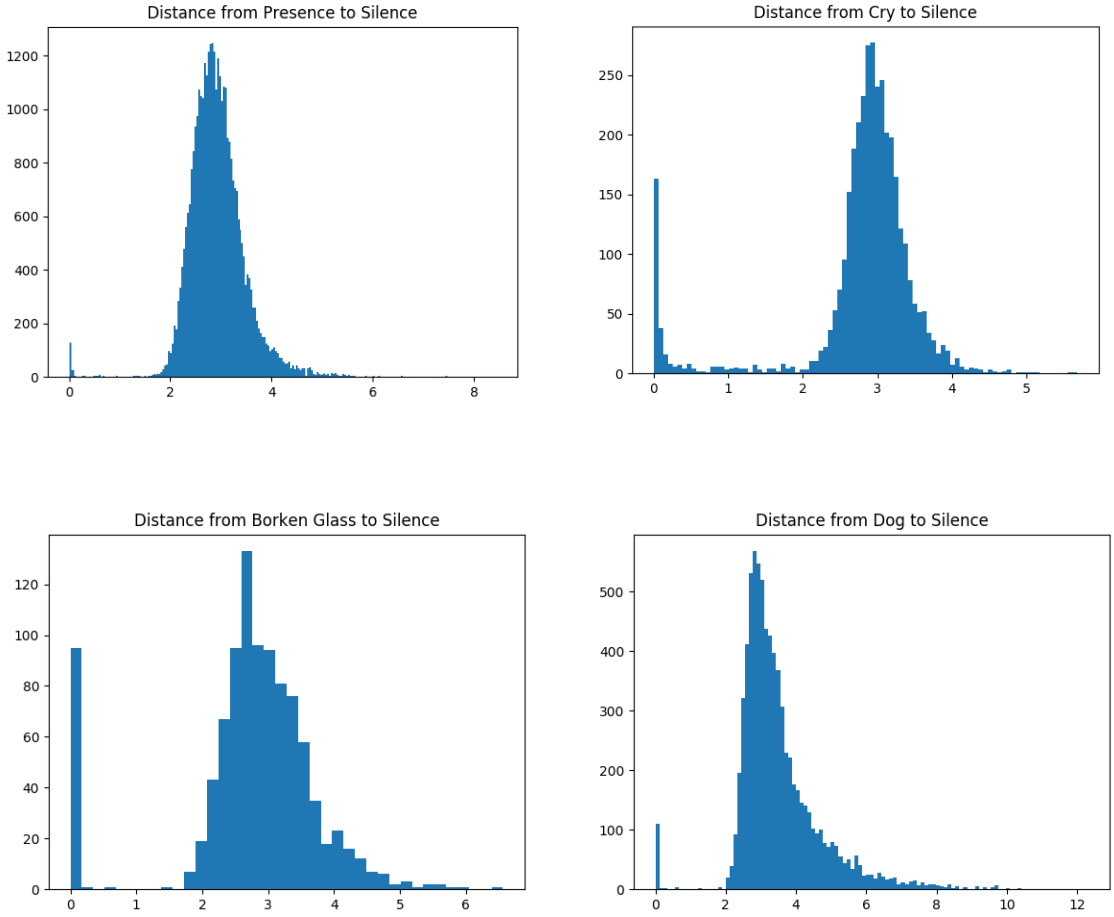



*Figura 30: Espectrograma de audio de una puerta en el que se visualiza la falta de información para cualquier muestra que cuyos 960 ms abarquen el rango desde 3,5 a 7,5 segundos (elaboración propia)*

Para solucionar este problema se aprovechó el codificador *vggish* de la segunda arquitectura para generar los vectores característicos de cada una de las ventanas de 960 ms. Con estos vectores de dimensión 128 se emplearon técnicas de aprendizaje no supervisado,

aprovechando que se tenían ejemplos de vectores que se sabe con certeza que pertenecen a la clase buscada, y se compararon los candidatos a esa clase con el centroide generado en el espacio euclídeo. En la Figura 31 se puede observar como hay una cierta cantidad de ejemplos en la clase de presencia (pasos más puertas) que están muy cercanos al centroide artificial de la clase Silencio, generada para poder hacer esta limpieza. Esto permitió eliminar silencios y ruidos de fondo mal etiquetados en el set de entrenamiento. Esta técnica se usó para limpiar clases tanto de posibles muestras de silencio y presencia que se pudieran dar en otras clases debido a la técnica empleada para el pre procesamiento de los datos.



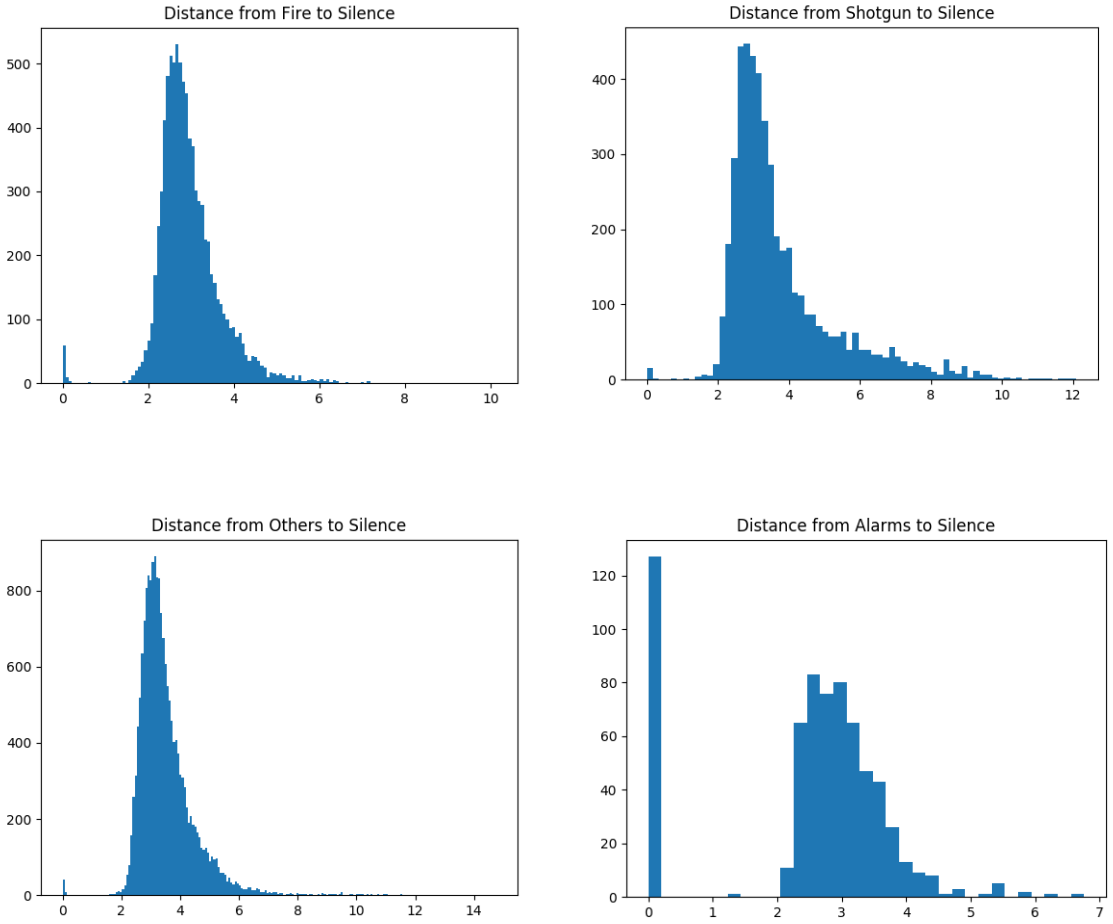



*Figura 31: Imágenes de la distancia euclídea de las muestras de cada clase a una muestra de Silencio (elaboración propia)*

Las muestras que se "eliminaron" con esta limpieza se reciclaron y se añadieron a otras clases más convenientes y representativas de su sonido. Los silencios se trasladaron a la clase Otros y las muestras que eran más parecidas a pasos o puertas a la clase Presencia.

### 3.4.4 CREACIÓN DE LA CLASE *PRESENCIA*

En un primer momento la clase presencia no existía y su creación fue el resultado de otro pre procesamiento que se llevó a cabo con los datos tras los análisis de las etapas de entrenamiento.

Esta clase es el resultado de unir las muestras de las clases Pasos y Puertas en una sola clase: Presencia. La decisión vino fundamentada por los primeros resultados obtenidos con los modelos, en los cuales las clases de Pasos y Puertas eran las que tenían perores KPIs en cuanto a precisión. Mas concretamente, esto se debía a que se estaban produciendo confusiones en muchos fragmentos de audios entre ellas, es decir Pasos que eran clasificados como Puertas y viceversa. Esto resultó bastante lógico, ya que tras varios análisis y pruebas de los casos en los que el modelo no acertaba se determinó que esas muestras eran difíciles de clasificar incluso para el oído humano. De esta manera y tras deliberarlo con los *stakeholders* se determinó que la mejor solución sería la unión de ambas clases en una sola, ya que esto no comprometía el uso final de la solución, ya que el fin último de ambas clases era el de detectar si había presencia humana en la zona de actuación del sistema de video-vigilancia.

## 3.5 MODELADO DE ALGORITMOS Y ARQUITECTURAS

### 3.5.1 DESCRIPCIÓN DE LAS ARQUITECTURAS

Como se ha mencionado en apartados anteriores se optó por entrenar dos algoritmos diferentes pero que se engloban dentro del mismo tipo de algoritmos para la clasificación de audio. Esto se hizo ya que los algoritmos que se van a presentar están considerados como el estado de la cuestión para diversos sets de datos de audio. Son algoritmos similares pero que presentan ciertas ventajas y desventajas uno con respecto al otro

#### *3.5.1.1 Red convolucional 2D ad hoc (2DCNN)*

Esta arquitectura se basa en la mostrada en la Figura 25, donde el espectrograma se hace pasar por una serie de campas convolucionales y después son las capas densas las que clasifican el resultado de las convoluciones.

Esta arquitectura se presenta en el estudio [1] y se escogió como uno de los algoritmos a probar en este proyecto, ya que permite una gran flexibilidad a la hora de crear la arquitectura frente a la otra solución propuesta. Esta solución tiene la ventaja de que es más *ad hoc* y por lo tanto permite más control y adaptabilidad a la problemática del problema, sin embargo, conlleva el riesgo de que su curva de escalabilidad frente a nuevas clases o muestras es mucho peor que para el otro algoritmo. Esto se debe a que en caso de querer añadir nuevas clases puede que sea necesario también ampliar la red y re entrenar todas las capas, con el coste que ello conlleva.

En la Tabla 3 se puede observar un resumen de los parámetros de la variante más representativa de cada una de las arquitecturas que se probaron. Las variaciones más significativas entre las arquitecturas 2DCNN son las siguientes:

- Las arquitecturas 2DCNN_1 y 2DCNN_3 solamente difieren en el tamaño y disposición de los *kernels.* En concreto los *kernels* de la arquitectura 2DCNN_1 son bastante especiales y fuera de lo común en lo que respecta a las arquitecturas convolucionales.
- La arquitectura 2DCNN_2 es una variación con *dropout* de la arquitectura 2DCNN_1. Nótese, que ambas tienen el mismo número de parámetros, ya que la acción del *dropout* no afecta al número total de parámetros, si no a la manera de entrenarlos

Dentro de la idea general de cada una de estas arquitecturas se exploraron diferentes variaciones, pero las que se muestran en la tabla corresponden con la variación que obtuvo mejores resultados.

| Method | Trainable Params | Non-trainable Params | Total Params |
|---|---|---|---|
| 2DCNN_1 | 149.832 | 384 | 150.216 |
| 2DCNN_1 | 149.832 | 384 | 150.216 |
| 2DCNN_1 | 51.848 | 576 | 52.424 |
| Classifier | 826.121 | 3.840 | 829.961 |
| Classifier Dropout | 826.121 | 3.840 | 829.961 |

*Tabla 3: Resumen del número de parámetros de los ejemplos más representativos de cada una de las arquitecturas.*

#### *3.5.1.2 Encoder convolucional más clasificador (E + C)*

Esta arquitectura se basa en la solución del estado del arte presentada en [2] y se compone de dos arquitecturas diferentes, de manera que la salida de una es la entrada de la siguiente. Dichas arquitecturas son:

- **Arquitectura pre entrenada *vggish*:** la estructura de este modelo se puede ver la Figura 32 y para este proyecto se empleó el modelo pre entrenado por Google en el set de datos de Youtube 8M. Esta técnica se englobaría dentro del contexto del *transfer learning* y tiene la ventaja de que permite aplicar arquitecturas y modelos entrenados en sets muy grandes con tecnologías muy punteras, que de otra forma serían muy costosos de crear y poner a punto. La salida de esta arquitectura es un vector que engloba la representación que la red hace de la imagen recibida en un espacio latente euclídeo de 128 dimensiones. Esta red ha sido previamente entrenada con el objetivo de separar espectrogramas de audios en este espacio, por lo que es perfecta para el objetivo de este proyecto. Esta salida se usó como entrada para la siguiente parte de la arquitectura.

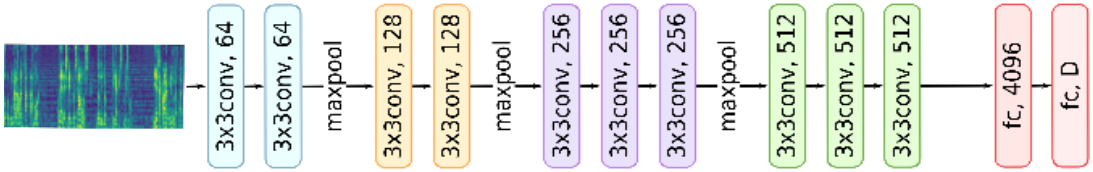



*Figura 32: Arquitectura de la red pre entrenada Vggish, basada en el modelo de [14]*

- **Red densa *fully-connected***: esta parte de la arquitectura toma la salida de la red *vggish* y la hace pasar por una serie de capas de red neuronal conectadas que dan como salida final una predicción de las clases del audio. Esta red al no ser como *vggish*, que ya viene con la arquitectura establecida al aplicar *transfer learning*, permite algunas variaciones y modificación de su estructura. Para este proyecto se estudiaron dos de estas posibles arquitecturas, una red densa *fully-connected* y una variante de esta red con *dropout.* Un resumen del número de parámetros de las mejores versiones de estas dos arquitecturas se puede ver en la Tabla 3. Nótese, que ambas tienen el mismo número de parámetros, ya que la acción del *dropout* no afecta al número total de parámetros, si no a la manera de entrenarlos. Para ambas arquitecturas se probaron distintas variantes con menor y mayor número de neuronas y capas y en el caso de la variante de *dropout*, con distinta probabilidad de desconexión.

### 3.5.2 CONSTRUCCIÓN DE LOS MODELOS

La construcción de los modelos se llevó a cabo usando las librerías de *Tensorflow* y *Keras* en el lenguaje de programación *Python.*

Cabe destacar que todas estas arquitecturas no se crean partiendo de cero, si no que existen numerosos estudios y trabajos previos al respecto sobre los que apoyarse para saber qué combinaciones de hiper parámetros y buenas prácticas son las que producen *a priori* mejores resultados, como por ejemplo el uso de *kernels* convolucionales con tamaño de números

primos (3x3, 5x5…) o el uso de un número de *kernels* de la forma $2^n$. Un ejemplo de estas prácticas extraídas de un caso real y para redes convolucionales se puede ver en el estudio [15].

De la misma manera también existen significativos estudios y catálogos de buenas prácticas en relación a las herramientas de regulación empleadas en las diferentes variantes de los modelos y a los hiper parámetros usados para el entrenamiento. Un ejemplo de esto se muestra en la referencia [16]

## 3.6 ENTRENAMIENTO DE MODELOS

### 3.6.1 ENTRENAMIENTO

El proceso de entrenamiento se llevó a cabo empleando la herramienta de procesamiento en la nube de *Google Colab*, ya que permite la utilización gratuita de GPUs, *hardware* que permite a su vez el entrenamiento más rápido de cualquier arquitectura que requiera cálculos matriciales como los de la red convolucional 2D completa de la primera metodología. Para la arquitectura del *encoder* pre entrenado se optó por transformar todas las muestras en sus correspondientes vectores característicos primero y usar esos vectores para entrenar la red neural densa directamente. Con esto se logró ahorrar un gran tiempo en el apartado de entrenamiento que permitió realizar más iteraciones y mejorar el modelo más rápidamente.

Para ambos métodos se utilizaron algunas técnicas de regularización como el *Dropout*, o *Batchnormalization* para el método 2DCNN y *Early Stopping* para ambos. La ponderación de clases (*class weight)* también se utilizó en ambas soluciones para tratar los problemas derivados de tener un conjunto de datos desequilibrado.

Todos los entrenamientos se hicieron usando el optimizador Adam con una tasa de aprendizaje de 0.001 y la entropía cruzada como función de pérdida para el modelo.

El entrenamiento se hizo para 3 variantes de redes neuronales convolucionales con el método 2DCNN y para un clasificador estándar y el mismo clasificador aplicando capas de "Dropout" para el método del *encoder*.

En la Figura 33 y Figura 34 se pueden observar las curvas de entrenamiento para las variantes que mejores resultados dieron del modelo E + C y 2DCNN respectivamente.

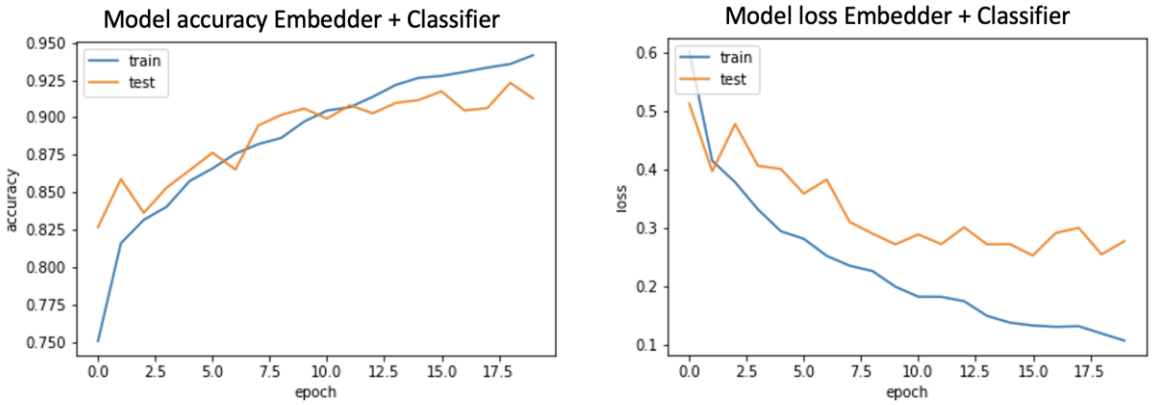



*Figura 33: Curvas de precisión y función de coste del entrenamiento del modelo Embedding (Elaboración propia)*

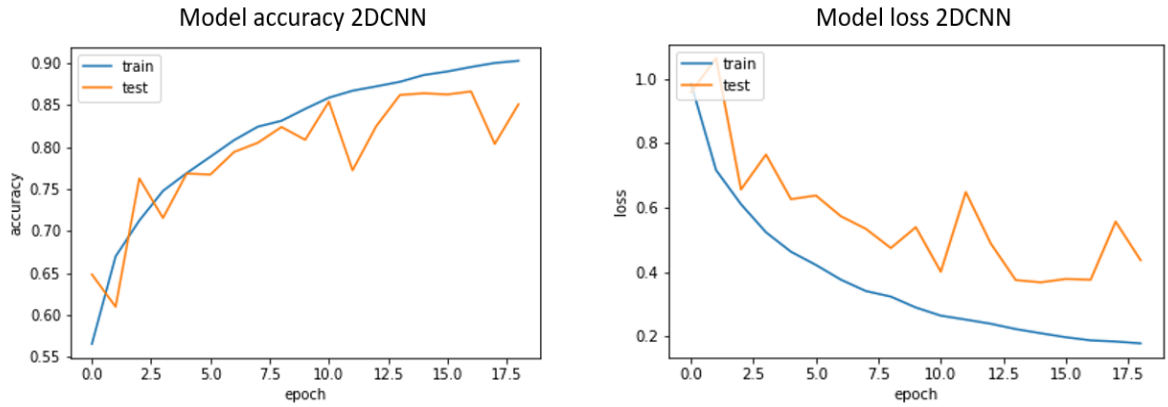



*Figura 34: Curvas de precisión y función de coste del entrenamiento del modelo 2DCNN (Elaboración propia)*

### 3.6.2 ANÁLISIS DE RESULTADOS

Para mejorar el entrenamiento e ir consiguiendo mejores resultados en cada iteración, se emplearon tanto las curvas de entrenamiento como las matrices de confusión del modelo en el set de test, un ejemplo de estas matrices se puede ver en la Figura 35. Este procedimiento permitió el uso de la realimentación para poder aplicar mejoras tanto en los procesos de pre procesado de datos, como de entrenamiento y basándose en la salida del modelo ir poco a poco mejorándolo.

El análisis de las matrices de confusión y de las curvas de entrenamiento permitió la mejora de los modelos, de las técnicas de pre procesado y del set de datos. Combinado con el entrenamiento se elaboró un código que recogía aquellas muestras que habían sido mal clasificadas por el modelo y posteriormente eran analizadas con un *notebook* para analizar ciertas muestras concretas.

Esta metodología permitió detectar fallos como el que se menciona en la sección 3.4.3, donde se explica como a raíz de las matrices de confusión, y un posterior estudio de los casos problemáticos, se mejoró el set de entrenamiento.

# Capítulo 4. RESULTADOS Y ELECCIÓN DE MODELOS

## 4.1 ANÁLISIS GENERAL

### 4.1.1 PRIMER ANÁLISIS (CLASES ORIGINALES)

En esta fase se compararon las 3 variantes del modelo 2DCNN y las variantes con y sin *dropout* del modelo E + C. Se obtuvieron los resultados mostrados en la Tabla 4.

| Method | Accuracy (%) | Loss | Val accuracy (%) | Val Loss |
|---|---|---|---|---|
| 2DCNN_1 | 88,567 | 0,210 | 86,394 | 0,368 |
| 2DCNN_2 | 75,558 | 0,478 | 78,168 | 0,596 |
| 2DCNN_3 | 84,293 | 0,285 | 86,491 | 0,368 |
| Classifier | 87,469 | 0,234 | 85,419 | 0,446 |
| Classifier Dropout | 80,776 | 0,377 | 84,639 | 0,445 |

*Tabla 4: Resultados del entrenamiento en la primera etapa del proyecto*

Estos resultados iniciales están alejados de la solución final, ya que se obtuvieron sobre un set de datos que luego terminaría sufriendo bastantes modificaciones. Sin embargo, fueron muy importantes para analizar cuál de las arquitecturas demostraba un mejor rendimiento sobre el set de datos puro, y lo que es más importante, combinados con el análisis de las matrices de confusión permitieron una mejora posterior de todo el proceso. La Figura 35 muestra las matrices de confusión de entrenar el modelo E + C sin Dropout y el primer

modelo 2DCNN respectivamente. Se puede ver como las clases se corresponden con las anteriores a todos los pre procesamientos y cambios que se produjeron después a raíz de los análisis.

A partir de este punto se presentan únicamente los resultados de la mejor solución para cada uno de los métodos, que fueron el primer modelo 2DCNN y el modelo E+C en su versión de clasificador sin *dropout*. Esto se hizo, ya que en las siguientes etapas la diferencia se agrando entre las distintas variantes y el objetivo final del proceso era quedarse con uno de los dos métodos.

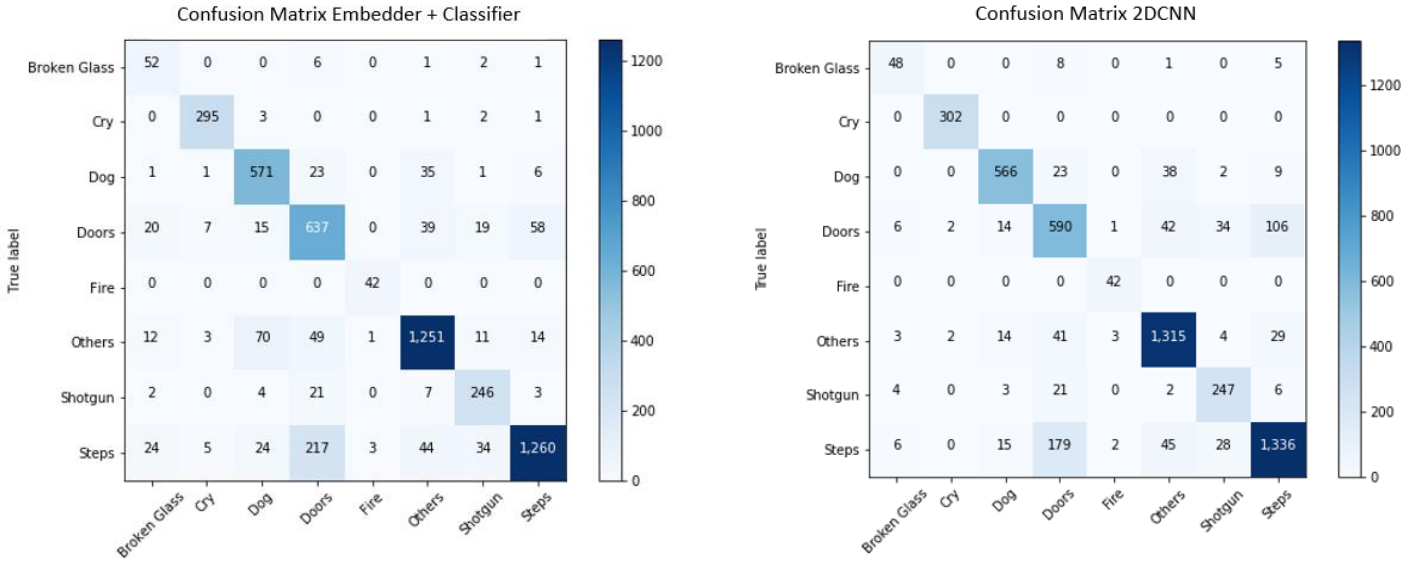



*Figura 35: Matriz de confusión del entrenamiento de los modelos finales sobre los datos originales (elaboración propia)*

### 4.1.2 SEGUNDO ANÁLISIS (CLASES FINALES)

Este segundo análisis se llevó a cabo tras un gran proceso de iteraciones en el cual se fueron refinando tanto los modelos como los procesos de pre procesamiento y la calidad de los datos. La idea de este segundo análisis es la de comparar los modelos con la mira puesta en elegir uno de los dos para ser implementado. En este apartado se presenta únicamente la

comparación final entre la mejor versión del modelo 2DCNN y del modelo E + C, sobre las clases y los datos definitivos.

| Method | Accuracy (%) | Loss | Val accuracy (%) | Val Loss |
|---|---|---|---|---|
| 2DCNN_1 | 93,86 | 0,124 | 89,83 | 0,321 |
| Classifier | 92,77 | 0,132 | 91,73 | 0,251 |

*Tabla 5: Resultados del entrenamiento para las versiones definitivas de ambos modelos*

En la Tabla 5 se ven los resultados para las dos variantes. En los resultados se puede observar cómo, aunque el modelo 2DCNN presenta una mejor precisión en el set de entrenamiento, es el modelo del Clasificador el que presenta un mejor KPI en el set de test. En el apartado 574.2.1 se hace un estudio más detallado de la precisión de los modelos y una comparación empírica de ambos.

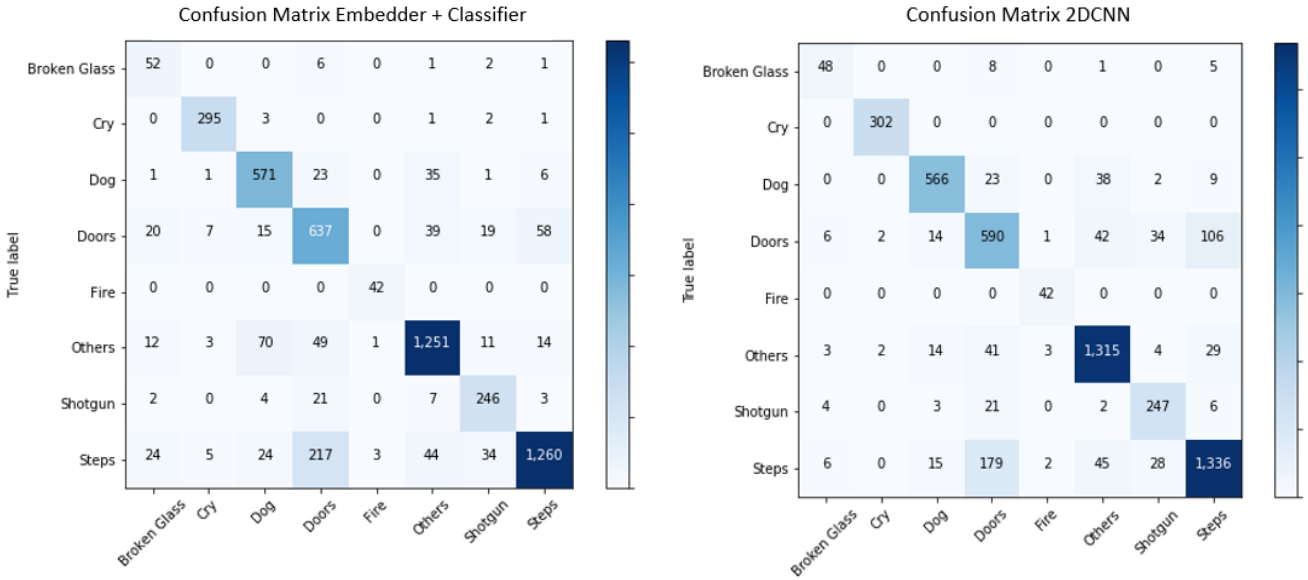



*Figura 36: Matriz de confusión del entrenamiento de los modelos finales (elaboración propia)*

En la Figura 36 se pueden observar las matrices de confusión finales para cada una de las mejores variantes de los modelos, donde ya se puede ver la configuración final de las clases, con las clases Presencia y Alarmas. Se puede observar también como los resultados obtenidos ya son mucho mejores y los errores son residuales o se dan en clases en los que es entendible que se puedan producir confusiones, como son Perros, Presencia y Otros.

## 4.2 ESTUDIO DE RESULTADOS Y ELECCIÓN DE MODELOS

En vista de los resultados planteados en el apartado anterior se planteó el un *Framework* basado en los siguientes puntos para la elección del modelo

### 4.2.1 PRECISIÓN

En las clases establecidas: esto se puede analizar mediante las matrices de confusión. Donde se aprecia que ambos modelos tienen grandes resultados y sus principales problemas vienen en muestras muy puntuales, lo cual es normal, ya que, haciendo un estudio más profundo de los fallos de estas clases, se corresponden con fragmentos que son muy complicados de distinguir incluso para el oído humano. En este apartado ambos modelos están muy parecidos, como muestra la Tabla 6.

| Classes | Accuracy | | Precision | | Recall | | Specificity | |
|---|---|---|---|---|---|---|---|---|
| | 2DCNN | E + C | 2DCNN | E + C | 2DCNN | E + C | 2DCNN | E+C |
| Alarms | 99.10% | 99.58% | 67.65% | 80.95% | 98.57% | 95.77% | 99.08% | 99.57% |
| Glass | 98.99% | 99.07% | 58.89% | 50.70% | 84.13% | 81.82% | 98.97% | 99.06% |
| Cry | 100.00% | 99.50% | 100.00% | 94.60% | 100.00% | 97.37% | 100.00% | 99.45% |
| Dog | 99.15% | 97.35% | 95.18% | 86.23% | 89.34% | 92.33% | 98.98% | 96.82% |
| Fire | 100.00% | 99.87% | 100.00% | 91.38% | 100.00% | 100.00% | 100.00% | 99.87% |
| Others | 97.83% | 98.94% | 94.14% | 96.82% | 91.30% | 88.45% | 96.68% | 98.43% |
| Presences | 96.02% | 98.52% | 81.69% | 93.97% | 82.53% | 94.38% | 95.17% | 98.07% |
| Shotgun | 98.74% | 98.91% | 84.72% | 86.10% | 89.79% | 89.75% | 98.64% | 98.83% |
| Total (macro average) | 98.73% | 98.97% | 85.28% | 85.09% | 91.96% | 92.48% | 98.44% | 98.76% |
| Total (micro average) | 89.83% | 91.73% | 89.83% | 91.73% | 89.83% | 91.73% | | |

*Tabla 6: Comparativa de los KPIs de las versiones definitivas de los modelos 2DCNN y Encoder + Clasificador (E+C)*

La Tabla 6 muestra un resumen de los diferentes KPIs de las versiones finales de ambos modelos. En cuanto a la fila correspondiente a los valores totales, el *macro average* está calculado como la media de todas las medidas de ese KPI para las clases, mientras que el *micro average*, es el cálculo de ese KPI para el modelo global. De estos resultados se pueden sacar las siguientes conclusiones:

- La precisión global del modelo es mejor para el modelo E + C que para el modelo 2DCNN, superando el modelo E + C el 90% de precisión en el test de validación.
- El modelo E + C es superado única y ligeramente por el modelo 2DCNN en la precisión media por clase.
- Dependiendo del tipo de clase resulta más interesante analizar la precisión o el *recall.* Para aquellas clases que no nos podamos permitir que el modelo pase por alto una muestra de ese estilo será más importante el *recall (*este es el caso de la mayor parte de los casos críticos en el sector de la seguridad, responde a la pregunta: ¿De todos aquellos que eran robos cuántos hemos detectado?) y para aquellas clases en los que lo importante sea no estar molestando constantemente al usuario con falsas alarmas,

pero no sea tan crítico detectarlos bien, miraremos la precisión (generalmente estas son las clases de *comfort)*. Atendiendo a este criterio se puede hacer el siguiente análisis:

- La clase más importante de cara al *recall* sería Presencia, ya que es la clase clave que permite detectar robos cuando la capacidad visual de las cámaras de seguridad es burlada por posibles asaltantes. En esta clase la ventaja del modelo E + C es abrumadora y prácticamente decanta la balanza del lado de este modelo, ya que es el caso más crítico de estudio para el proyecto.
- Otras clases interesantes para el *recall* son las clases Perro y Cristales, en las cuales el resultado está muy parejo.
- En cuanto a las principales clases de *comfort* (Alarmas, Lloros y Fuego) el resultado en la precisión es algo ventajoso para el modelo 2DCNN en las clases Lloros y Fuego y existe una diferencia bastante grande (aprox. 12%) a favor del modelo E + C en la clase Alarmas.

La conclusión final de este análisis es que el modelo E + C debe ser el elegido si se atiende únicamente a los KPIs de los modelos, ya que es el que mejor se ajusta al objetivo original y principal del proyecto, que es el de detectar posibles situaciones de robo. La clase Presencia es de vital importancia en este aspecto y una diferencia tan grande en el *recall* de esta clase hace que sea la mejor opción a escoger en base a los requerimientos y el objetivo final del proyecto.

### 4.2.2 TIEMPOS DE PROCESAMIENTO

Ambos modelos presentaron tiempos similares en los diferentes entrenamientos y en sus diferentes variantes. Sin embargo, hay que tener en cuenta que del modelo del Encoder más el clasificador solamente se entrenaba el clasificador y que los audios se pre procesaban antes del entrenamiento para acelerarlo. Igualmente, el modelo 2DCNN es lógicamente más rápido en la inferencia, ya que es una versión simplificada, pero más flexible del modelo Encoder + Clasificador. También cabe resaltar que para el resultado del proyecto el tiempo realmente importante es el de inferencia, ya que va a ser lo que facilite una mejor puesta en producción posterior del modelo. Tener buen tiempo y flexibilidad en el entrenamiento es relevante si se quieren tener un modelo flexible y rápidamente adaptable, sin embargo, la inferencia es más determinante en el entorno de producción.

En la

Tabla 7 se muestran los tiempos de inferencia para ocho audios diferentes con longitudes diferentes y correspondientes a una clase distintas. En la sección 8.1 se muestran los espectrogramas y ondas de estos audios para poder hacerse una mejor idea de su longitud y aspecto. La columna *samples* muestra el número de muestras en las que la conversión en espectrogramas de Mel ha dividido ese audio. Las columnas siguientes muestran los tiempos de procesamiento en segundos que ha tardado cada uno de los modelos en procesar todas las muestras. Por último, en la última fila se muestra el tiempo medio de cada uno de los modelos por muestra.

La inferencia para ambos modelos se hizo sobre el mismo *hardware (*Una *GPU* en *Google Colab)* y en las mismas condiciones, midiendo el tiempo desde que se recibe el archivo de audio hasta que se obtiene una predicción del mismo. Se ha de tener en cuenta que la variación de tiempos no es proporcional al número de muestras, ya que la inferencia con *GPU* permite escalar fácilmente el número de muestras a la entrada en las redes convolucionales por su facilidad para paralelizar cálculos matriciales.

| Audio | Samples (#) | 2DCNN | | Embeddings | |
|---|---|---|---|---|---|
| | | Time (s) | T/sample (s) | Time (s) | T/sample (s) |
| Cry | 13 | 0.2778 | 0.0213 | 0.6028 | 0.0463 |
| Dog | 9 | 0.0456 | 0.0035 | 0.6973 | 0.0774 |
| Door | 21 | 0.0921 | 0.0043 | 0.5014 | 0.0238 |
| Fire | 1 | 0.0171 | 0.0171 | 0.2299 | 0.2299 |
| Glass | 5 | 0.1102 | 0.0220 | 0.3497 | 0.0699 |
| Gun | 2 | 0.0103 | 0.0051 | 0.2756 | 0.1378 |
| Others | 7 | 0.0418 | 0.0059 | 0.2946 | 0.0420 |
| Steps | 59 | 0.2056 | 0.0034 | 0.9035 | 0.0153 |
| Average time | 117 | 0.0068 | | 0.0329 | |

*Tabla 7: Comparación de tiempos de inferencia de cada uno de los modelos para 8 ejemplos de audios*

De estos resultados se puede concluir que el modelo 2DCNN_1 tiene un tiempo de procesamiento de un orden de magnitud 5 veces más pequeño que el modelo con el *Encoder*. Esto entra dentro de lo esperado, ya que el número de parámetros y etapas del segundo modelo es mucho mayor que del primero. Lo verdaderamente importante en este análisis es el estudio de estos tiempos frente al mundo real donde se va a implantar la solución. Cada muestra analizada se corresponde con un audio de 960 ms de longitud, sin embargo, hay que tener en cuenta que cada muestra contiene 480 ms de audio aun no analizado y otros 480 ms de audio que solapa con la muestra anterior y que por tanto no se corresponden con audio aún no "escuchado" por el modelo. De esta reflexión se puede concluir que el tiempo máximo que el tiempo por muestra máximo del modelo para que este pueda funcionar en tiempo real debe ser inferior a 0.48 segundos, lo cual cumplen ambos modelos para su peor tiempo por muestra (ambos marcados en rojo en la tabla).

### 4.2.3 ESCALABILIDAD:

puesto que el número y el tipo de clases puede crecer en el futuro, según las necesidades de la industria o de los clientes, es necesario que la solución fuera lo más escalable posible. Esto se consigue más fácilmente con el modelo E + C, ya que el modelo 2DCNN es una solución más *ad hoc* y que saturaría más rápidamente con un aumento del número de clases. Mientras que el modelo *vgg* sobre el que se basa la otra arquitectura, está pensado para poder clasificar sin problemas miles de clases y en esta solución se emplean únicamente ocho. Se ha de tener en cuenta que, aunque pueda parecer un modelo desproporcionado para el problema que se plantea, gracias al uso del *transfer learning* se evita la parte más compleja y costosa de la aplicación de este modelo, el entrenamiento.

### 4.2.4 ELECCIÓN FINAL

Con estos análisis y con la experiencia de haber tenido que realizar diferentes procesos iterativos y de re entrenamiento con los diferentes cambios que se fueron introduciendo, el modelo elegido e implementado para resolver el problema planteado fue el E + C. Se escogió este modelo, ya que logró unos KPIs algo mejores para el problema planteado, especialmente en la clase Presencia, y una mayor flexibilidad en caso de querer aumentar o cambiar el número y tipo de clases de salida, que fue algo que se tuvo que hacer en varias ocasiones a lo largo del proyecto.

A pesar de tener mayor tiempo de inferencia, los valores seguían dentro de los requerimientos establecidos y permiten la total funcionalidad del modelo en un entorno real, lo que lo convirtió en el modelo seleccionado. Obviamente, tener un tiempo de inferencia menor siempre es deseable, pero en esta ocasión se tuvo que hacer un *trade-off* entre los tres aspectos (precisión, escalabilidad y velocidad). El contexto y solución final del proyecto (orientados a la seguridad) decantaron la balanza por el modelo con mayor precisión y fiabilidad de resultados.

Este modelo se trasladó a los *stakeholders* de manera que les fuera sencilla su puesta en producción, como se verá en el apartado de implementación del proyecto.

# Capítulo 5. IMPLEMENTACIÓN Y SISTEMA

De la implementación final del sistema de cara al usuario se desconocen los detalles exactos, pero se elaboraron las herramientas solicitas por los *stakeholders*, para que pudieran probar el modelo y proponer cualquier modificación que creyeran conveniente. En esta línea se implementaron 2 soluciones: una App web para poder hacer pruebas del modelo de manera sencilla y una herramienta sencilla en *Python* para que pudieran integrar el modelo en sus propios sistemas de manera sencilla.

## 5.1 APP

Se desarrolló una pequeña aplicación web utilizando *Docker* y los recursos que ofrece Google Cloud para desplegar este tipo de soluciones. La aplicación estaba constituida por un *frontend* muy sencillo (ver Figura 37). Esta interfaz permitía al usuario subir los archivos de audio que quisiera analizar y obtener los resultados que el modelo diera de esos audios.


*Figura 37: Frontend de la aplicación web desarrollada*

```
<head>
    <link href="{{ url_for('static', path='/style.css') }}"
rel="stylesheet", type="text/css">
    <title>Upload wav file</title>
</head>
<body>
    <h1>Upload an Audio</h1>

    <form action="/" method="POST" enctype="multipart/form-data">
        <label>Select .wav file</label>
        <div class="custom-file">
            <input type="file" class="custom-file-input" name="audio"
id="audio">
            <label class="custom-file-label" for="audio"></label>
        </div>
        <button type="submit" class="btn btn-primary">Upload</button>
    </form>

    <div>
        <button id="myButton" class="float-left submit-button"
>Predict</button>
    </div>

    <script type="text/javascript">
    document.getElementById("myButton").onclick = function () {
        location.href = "/predict";
    };
    </script>
</body>
```

*Código 2: Código HTLML para el Frontend de la aplicación*

El Código 2 se corresponde con el archivo HTML usado para crear el *frontend*. El *backend* de la aplicación estaba construido en Python y es el mismo que se desarrolló para elaborar la herramienta que se comentara en el punto siguiente.

## 5.2 HERRAMIENTA PYTHON

Una vez probado el modelo a través de la App Web descrita en el apartado anterior, se procedió al traslado del modelo desarrollado. Esta entrega se hizo a través de una herramienta en Python, con una pequeña interfaz para poder probar el modelo, pero, sobre todo, con todo el código documentado, de manera que la extracción e implementación del modelo en los sistemas de los *stakeholders* fueran lo más rápido posible.

Para el uso de la herramienta bastaba con situar los audios que se desearan analizar en la carpeta *input* y tras correr el archivo *main.py* en un entorno con los paquetes facilitados en el archivo *requirements.txt* se recibían los resultados para cada uno de los audios en la carpeta *output* en formato *json.* En el Código 3 se muestra el ejemplo de la salida para un audio de la clase Presencia.

En este código se puede observar como la salida consta de 4 apartados:

- Clase predicha para cada uno de las ventanas de 960 ms en las que se divide el audio
- Las posibles clases de las que consta el modelo
- El resultado final: es el resultado más significativo de los detectados en el conjunto de las ventanas de 960 ms.
- Tiempo de predicción: tiempo transcurrido desde que se ha empezado a tratar el audio hasta la salida de su predicción final.


```
    "predictions_every_960_ms": [
        "Presence",
        "Presence",
        "Presence",
        "Presence",
        "Presence"
    ],
    "possible_classes": [
        "Alarms",
        "Broken_Glass",
        "Cry",
        "Dog",
        "Fire",
        "Others",
        "Presence",
        "Shotgun"
    ],
    "final_result": "Presence",
    "time_to_predict_(s)": 0.08271312713623047
}
```


*Código 3: Ejemplo de la salida del modelo (Elaboración propia)*

## 5.3 RECOMENDACIONES EN LA IMPLEMENTACIÓN

Junto con las herramientas de los puntos 5.1 y 5.2, y desde la experiencia de haber trabajado con el modelo y conocer sus limitaciones, se plantearon una serie de recomendaciones a la hora de implementar la versión final. Para ello se tuvo también en cuenta la idea aproximada de como el cliente quería poner el modelo final en producción y con estas recomendaciones poder ayudarle a maximizar el rendimiento del modelo.

- **Uso de un buffer**

A la hora de poner un modelo de este estilo en producción y que funcione de manera continua es recomendable el uso de un *buffer* que vaya acumulando el audio recibido y vaya generando las muestras para ir alimentando al modelo y poder realizar la inferencia de las mismas.

Se debería combinar este *buffer* con un detector de ruido, que permita alimentar el modelo únicamente cuando se produzca una perturbación, ya que así no se saturaría fácilmente el *buffer* y solo se analizarían aquellos audios que de verdad pueden aportar una información extra a la de las cámaras de seguridad.

- **Umbrales para las clases**

Conjuntamente con el estudio de las arquitecturas y sus resultados para el problema propuesto se elaboró un estudio de sensibilidad ROC, ya que se trata de un proyecto en el que un umbral para reducir mejorar los KPIs Verdadero Positivo frente a Falso Positivo resulta realmente interesante. Esto se debe al contexto en el que se enmarca el proyecto, ya que en las clases que posteriormente van a representar un robo nos interesa que no se nos escape ningún Verdadero Positivo cuando el modelo esté activo, aunque ello pueda derivar

en un aumento de los Falsos Positivos. Se aplicaría la filosofía de "más vale prevenir que curar".

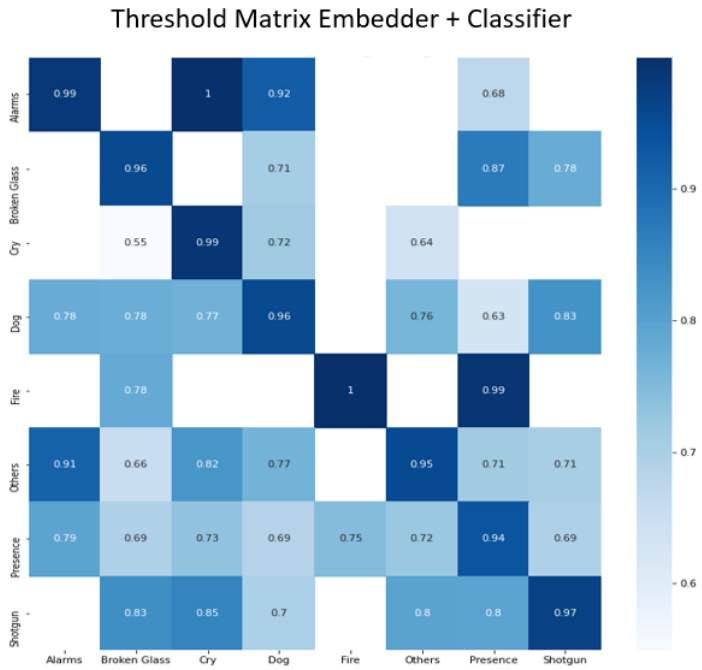



*Figura 38: Matriz de niveles de confianza medios del modelo para cada una de las clases*

La matriz de la Figura 38 muestra como el nivel de confianza medio en las predicciones del modelo para los Verdaderos Positivos (diagonal principal) es mucho mayor que para los False Positives. Esto implica que simplemente añadiendo un umbral una la implementación final del modelo se obtendrían mucho mejores resultados. Este umbral se puede ajustar dependiendo de la clase, pero un nivel general aceptable estaría en torno al 90%.

- **Primer *feedback* recibido**

Una vez se entregó el modelo se recibió un *feedback* por parte de los *stakeholders*, que lo probaron con sus propios datos y elaboraron sus propias matrices de confusión para su set de datos. Los resultados compartidos se pueden ver en la Figura 39, donde las clases Lloros y Fuego no se han tenido en cuenta, ya que hubo un mal entendido al transmitir a que hacían referencia en el caso de Fuego, y por falta de datos, en el caso de Lloros. En el caso de Alarmas también hubo un mal entendido debido a que se pensó por parte del cliente que esta clase hacía referencia a sirenas de ambulancia, mientras que esta clase se encontraba en Otros.

Estos problemas fueron resueltos posteriormente con un resumen como el mostrado en el apartado Explicación de las clases.

Los fallos entre Perros y Otros son completamente comprensibles, ya que en la clase Otros hay gran variedad de ruidos de animales. Al igual que lo son los errores entre Presencia y Disparo, puesto que hay disparos que se pueden confundir fácilmente con portazos fuertes, que son algunos de los ruidos que encontramos en la clase Presencia.

| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| Alarms | 1 | 174 | 25 | 64 | 57 | | 3 | 19 | 1 |
| Broken_Glass | 2 | 0 | 54 | 0 | 3 | | 0 | 10 | 9 |
| Cry | 3 | 18 | 0 | 15 | 15 | | 2 | 1 | 19 |
| Dog | 4 | 67 | 0 | 23 | 740 | | 0 | 6 | 1 |
| Fire | 5 | 1 | 0 | 0 | 0 | | 0 | 0 | 0 |
| Others | 6 | 632 | 0 | 5 | 146 | | 59 | 70 | 7 |
| Presence | 7 | 15 | 29 | 1 | 53 | | 10 | 819 | 166 |
| Shotgun | 8 | 0 | 3 | 0 | 6 | | 6 | 3 | 927 |

*Figura 39: Primeros resultados de test de los stakeholders de la seguridad (elaboración de los clientes)*

# Capítulo 6. CONCLUSIONES Y TRABAJOS FUTUROS

## 6.1 CONCLUSIONES

El proyecto permitió la obtención de un modelo a la altura del nivel del estado del arte en materia de clasificación de audio, pero con unas aplicaciones concretas y una solución orientada a un problema específico de la industria de la seguridad. El modelo y la aplicación desarrollados resultan una innovación en esta industria.

En el momento de redactar esta memoria los *stakeholders* del sector de la seguridad han comenzado a integrar el modelo en sus sistemas y realizado sus propias pruebas con el modelo con resultados muy satisfactorios. Cabe destacar que se cumplieron todos los requerimientos solicitados y además se amplió el alcance del proyecto incluyendo la variable del confort como valor añadido para los clientes finales del sistema.

## 6.2 TRABAJOS FUTUROS

A raíz del proyecto actual y de lo observado en la realización del mismo se plantean los siguientes trabajos futuros.

- **Implementación del modelo**

Este apartado correrá a cargo de los *stakeholders* del sector de la seguridad. El modelo trabajara de manera complementaria a cámaras de seguridad dotándolas de nuevas capacidades.

- **Estudio de la adición de la variable temporal**

El modelo actual cumple perfectamente la función de reconocer fragmentos de audio, sin embargo, resultaría interesante el estudio de la adición de modelos que, sobre el actual modelo elaborado, sean capaces de entender el contexto de ese fragmento. Estos modelos

pueden ser del estilo de los *Transformers* o modelos *NLP*. En el actual proyecto esto se trata de conseguir creando una superposición entre muestras, de manera que cada muestra está formada por un porcentaje del final de la muestra anterior, un porcentaje del inicio de la muestra siguiente y una parte que no tiene por qué pertenecer a ninguna de las dos.

Un ejemplo de una aplicación que utiliza la técnica de este proyecto con una capa extra para poder analizar la variable temporal de los audios se puede ver en [17], donde se ha utilizado el modelo *vggish* combinado con una red recurrente BiGru para capturar esta temporalidad.

- **Análisis de hiper parámetros**

Otra idea interesante, sobre todo relacionada con los puntos 3.3 y 4.1, fue un estudio más amplio de las diferentes variantes y posibilidades. Aunque se ha llevado a cabo un breve estudio, cambiando ambas arquitecturas, y se han probado variantes que no se muestran en el trabajo por haber dado peores resultados que las escogidas; se podría profundizar más en este aspecto y buscar optimizar estos valores para el objetivo propuesto. Resulta especialmente interesante en lo referente al modelo 2DCNN, ya que de los dos es el que más margen de cambio tiene, al haber sido completamente desarrollado para este proyecto.

- **Entrenamiento dinámico usando la capacidad humana**

Esta idea es bastante común en aplicaciones de *machine learning* cuyos resultados son fácilmente contrastables frente la capacidad humana para el mismo problema es el uso de una herramienta para poder entrenar el modelo automáticamente. Con esto se podría hacer todo el proceso que se hizo de pre procesado y etiquetado manual que se hizo con los datos de manera dinámica. La idea sería que para aquellas muestras que se puedan oír los audios más conflictivos o mal etiquetados y decidir si pertenecen a otra clase o si se pueden descartar para el proyecto. Esto permitiría también testear el modelo frente a los KPIs humanos y es básicamente la forma de proceder que se ha seguido a lo largo del proyecto para mejorar el modelo de manera progresiva, ya que la mayor parte de los cambios y mejoras surgieron del análisis exhaustivo de los casos en los que el modelo no clasificaba como debiera.

# Capítulo 7. BIBLIOGRAFÍA

[1] A. Guzhov, F. Raue, J. Hees and A. Dengel, "Esresnet: Environmental sound classification based on visual domain models," 2020.

[2] K. Palanisamy, D. Singhania and A. Yao, "Rethinking CNN Models for Audio Classification," 2020.

[3] D. Gartzman, "Getting to Know the Mel Spectrogram," August 2019. [Online]. Available: https://towardsdatascience.com/getting-to-know-the-mel-spectrogram-31bca3e2d9d0. [Accessed September 2021].

[4] L. Nannia, G. Maguoloa, S. Brahnamb and M. Pacic, "An Ensemble of Convolutional Neural Networks for Audio Classification," Julio, 2020.

[5] S. H. e. at, "CNN architectures for large-scale audio classification," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2017.

[6] A. Agrawal, "The Current State of the Art in Natural Language Processing (NLP)," *Towards Data Science*, 6 Junio 2020.

[7] J. Salamon and J. P. Bello, "Deep Convolutional Neural Networks and DataAugmentation for Environmental SoundClassification," *IEEE SIGNAL PROCESSING LETTERS*, 2016.

[8] M. Mishra, "Convolutional Neural Networks, Explained," *Towards Data Science*, 26 August 2020.

[9] J. Brownlee, "A Gentle Introduction to Cross-Entropy for Machine Learning," *Machine Learning Mastery*, 22 December 2020.

[10] A. Menendez, "Desarrollo de una app de reconocimiento facial con tecnicas de inteligencia artificial," Junio 2019.

[11] J. Brownlee, "A Gentle Introduction to Transfer Learning for Deep Learning," *Machine Learninig Mastery*, 20 December 2017.

[12] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks fromOverfitting," *Journal of Machine Learning Research* , vol. 15, pp. 1929-1958, 2014.

[13] A. Ilyas, A. Mądry, S. Santurkar and D. Tsipras, "How does Batch Normalization Help Optimization?," Gradient Science, 26 November 2018. [Online]. Available: http://gradientscience.org/batchnorm/.

[14] K. Simonya and A. Zisserman, "Very Deep Convolutional Neural Networks for Large Scale Image Recognition," in *Conference Paper at ICLR*, 2015.

[15] D. S. J. C. P. Patrice Y. Simard, "Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis," *Microsoft Research*, September 2003.

[16] P. Radhakrishnan, "What are Hyperparameters ? and How to tune the Hyperparameters in a Deep Neural Network?," *Towards Data Science*, 9 August 2017.

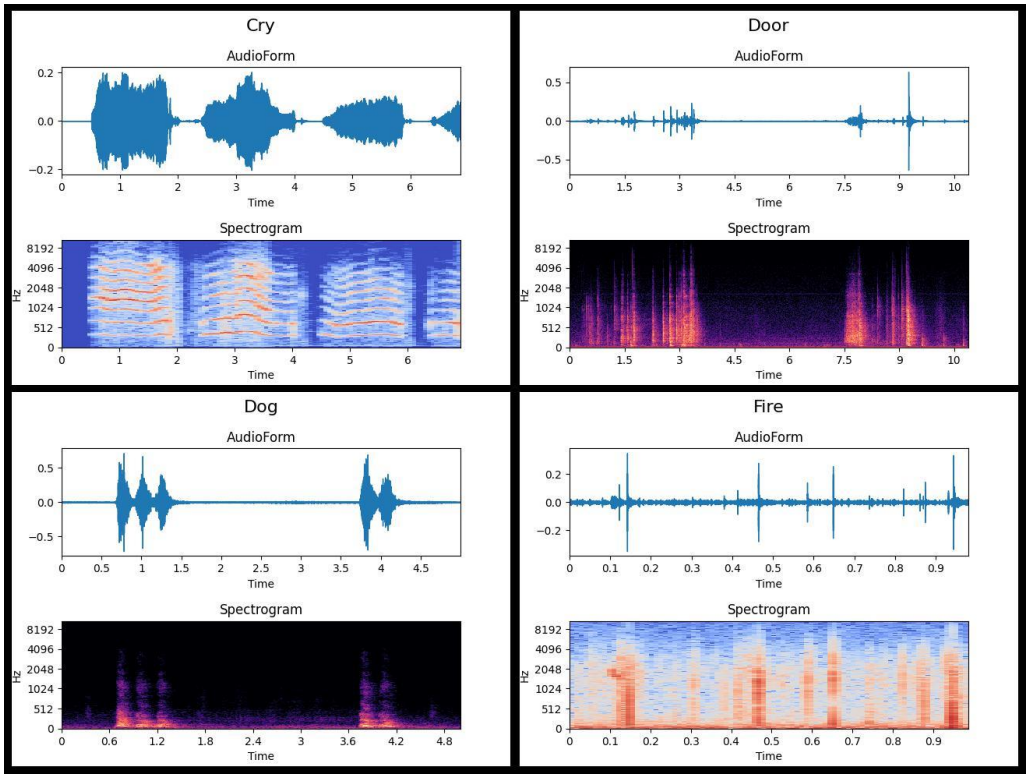
[17] L. SHI, K. DU, C. ZHANG, H. MA and W. YAN, "Lung Sound Recognition AlgorithmBased on VGGish-BiGRU," *IEEE Access*, 7 October 2019.

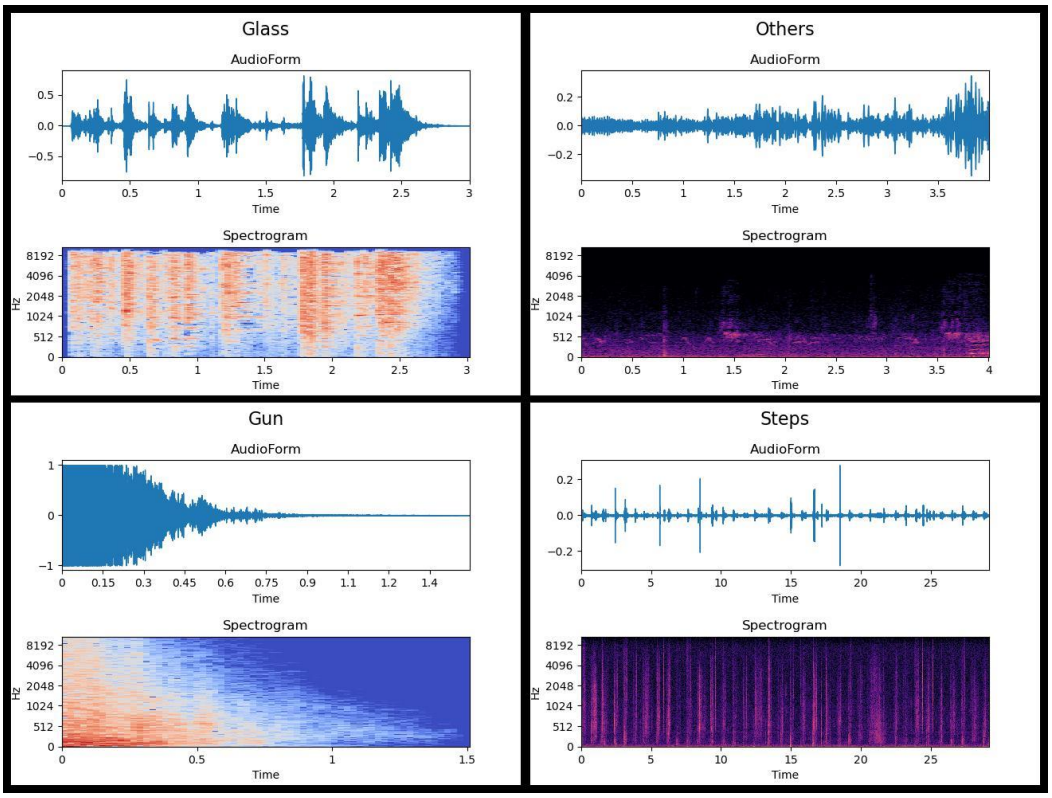
[18] "How to Improve Class Imbalance using Class Weights in Machine Learning," Analytics Vidhya, 6 October 2020. [Online]. Available:

https://www.analyticsvidhya.com/blog/2020/10/improve-class-imbalance-class-weights/.

# Capítulo 8. ANEXO A

## 8.1 EJEMPLOS DE ESPECTROGRAMAS




Glass
AudioForm
Time
Spectrogram
Hz
Time
Others
AudioForm
Time
Spectrogram
Hz
Time
Gun
AudioForm
Time
Spectrogram
Hz
Time
Steps
AudioForm
Time
Spectrogram
Hz
Time

# Capítulo 9. ANEXO B: OBJETIVOS DE DESARROLLO SOSTENIBLE

Este proyecto está alineado con dos de los objetivos de desarrollo sostenible propuestos por los líderes mundiales el 25 de septiembre de 2015. Los objetivos que aborda el proyecto son:

### 9.1.1 OBJETIVO 3: GARANTIZAR UNA VIDA SANA Y PROMOVER EL BIENESTAR PARA TODAS LAS EDADES

Este proyecto pretende mejorar el bienestar de las personas a través del confort y tranquilidad que van a entregar los sistemas de seguridad gracias al complemento de la clasificación de audio. Esto supone un gran aumento en el bienestar de los principales grupos consumidores de estos sistemas, que son familias o personas mayores, que se van a ver ampliamente beneficiados de las capacidades extra de estos sistemas, ya sea a través de tranquilidad o confort al saber que la tecnología está ahí y que no les va a fallar.

### 9.1.2 OBJETIVO 9: CONSTRUIR INFRAESTRUCTURAS RESILIENTES, PROMOVER LA INDUSTRIALIZACIÓN SOSTENIBLE Y FOMENTAR LA INNOVACIÓN

Este proyecto representa un claro ejemplo de innovación al introducir técnicas pioneras en una industria en la que no se han empleado todavía. También supone la apertura de puertas para que esta tecnología pueda ser utilizada en campos en los que nunca se había imaginado, permitiendo posibilidades que hasta ahora no se planteaban.