UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

TRABAJO FIN DE MÁSTER

Reconocimiento de eventos acústicos mediante el uso de *deep learning*

Máster Universitario en Ingeniería de Telecomunicación

Autor: Carlos Lamas Álvarez
Tutor: Doroteo Torre Toledano

FECHA: Junio 2019

RECONOCIMIENTO DE EVENTOS ACÚSTICOS MEDIANTE EL USO DE *DEEP LEARNING*

AUTOR: Carlos Lamas Álvarez
DIRECTOR: Doroteo Torre Toledano

Grupo audias
Dpto. de Ingeniería de Telecomunicaciones
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Junio 2019

Resumen

Resumen

En este Trabajo de Fin de Máster se ha implementado un sistema capaz de tomar un audio cualquiera y detectar eventos en él, dando como resultado la supuesta fuente que lo origina. Para ello se han empleado las técnicas de *deep learning* conocidas como redes neuronales. Más en profundidad, se han explorado los rendimientos de los tres estilos fundamentales de estas: las DNN, las LSTM y las CNN.

La inspiración para este proyecto surge de estudios previos publicados. Algunos de ellos hechos en la Universidad Autónoma de Madrid, como: "*Exploring convolutional, recurrent, and hybrid deep neural networks for speech and music detection in a large audio dataset*", escrito por Diego de Benito Gorrón, Alicia Lozano Díez, Doroteo Torre Toledano y Joaquín Gonzalez-Rodriguez, y "*Audio event detection on Google's Audio Set database: Preliminary results using different types of DNNs*", escrito por Javier Darna Sequeiros y Doroteo Torre Toledano.

Para poder implementar todo lo necesario a nivel técnico, ha sido necesaria la utilización de los lenguajes de programación Bash y Python. De este último se han utilizado principalmente las librerías: TensorFlow, que implementa lo necesario para generar redes neuronales, y Keras, que actúa por encima de la anterior y facilita enormemente su uso. Se ha usado también Numpy para tratamiento de datos, principalmente de matrices.

El material necesario para entrenar y probar el sistema ha sido la base de datos AUDIOSET. Esta es una base de gran tamaño, con más de 2 millones de segmentos de audio. Es fácilmente accesible y es gratuita. Google la ofrece a los interesados a través de descargas directas desde la plataforma de Youtube. También es importante destacar el sistema implementado en el paper "*CNN architectures for large-scale audio classification*", el cual ha sido adaptado para servir como medio para transformar cualquier audio de entrada que se quiera en un matriz de *embeddings*, el formato de entrada del clasificador entrenado en este trabajo.

Finalmente, para la evaluación de este proyecto se ha utilizado la medida de rendimiento mAP. Esta es la utilizada para este tipo de trabajos y la que la propia Google, entre otros, usa para sus publicaciones con la base da datos AUDIOSET. Consiste en calcular el rendimiento de los aciertos por clase, de entre las que tenga el clasificador implementado, y, posteriormente, hacer la media global de esos valores; con esto se conseguirá un valor entre 0 y 1, siendo 1 el mejor resultado posible.

Palabras Clave

Deep learning, DNN, LSTM, CNN, Bash, Python, TensorFlow, Keras, Numpy, AUDIOSET, mAP.

Abstract

This project has implemented a system that can take a random audio signal as input and detect audio events in it that should mach with the source of that audio. For this task it has been considered a few deep learning tecniches, such as: DNN, LSTM and DNN; these are known as Neural Networks.

As inspiration it has been used previous published research. Some of them had been done in the Universidad Autónoma de Madrid; for example: "*Exploring convolutional, recurrent, and hybrid deep neural networks for speech and music detection in a large audio dataset*", written by Diego de Benito Gorrón, Alicia Lozano Díez, Doroteo Torre Toledano and Joaquín Gonzalez-Rodriguez, or "*Audio event detection on Google's Audio Set database: Preliminary results using different types of DNNs*", written by Javier Darna Sequeiros and Doroteo Torre Toledano.

To make this possible at a technical level, it has been necessary the use of two different programming languages: Bash and Python. The last one was chosen because there are some libraries of deep learning, exclusive of Python, that are needed: TensorFlow, that has implemented all the algorithmics, and Keras, the one used to make easier the usage of TensorFlow. It has also been used Numpy for data treatment, mainly matrices.

The data needed to make possible the training and the trials has been AUDIOSET dataset. It is a huge set that has more than 2 million audio segments for the users. It is easy to access and it is completely free. Google allows everyone to download it from Youtube. It must be lightened the system implemented in "*CNN architectures for large-scale audio classification*", it has been modified just to be part of this project because it allows the user to transform a random audio signal in a matrix of embeddings, which is the input format chosen for this project.

Finally, for the evaluation task it has been used the performance measure mean Average Precision or mAP. This measure is the one mainly used for this type of research and even Google use it in their experiments with AUDIOSET dataset. It takes the performance achieved in each class by the classifier and calculate the mean of those values; the result is a number between 0 and 1, been 1 the best result possible.

Key words

Deep learning, DNN, LSTM, CNN, Bash, Python, TensorFlow, Keras, Numpy, AUDIOSET, mAP.

Agradecimientos

LLegados a este punto se hace difícil saber que decir. Todo ha sido una montaña rusa de situaciones y emociones. Sin embargo, todo lo bueno se acaba y, con este trabajo, se cierra una etapa en mi vida. Tal vez la etapa más importante hasta el momento.

Pero se supone que esta sección es para agradecer, no para ponerse melancólico... Tendré que poner de mi parte. Lo primero, como siempre, es darle las gracias a mi familia. En los malos momentos, en los que no eres capaz de ver el final, siempre están ahí. Tal vez no aportando ideas nuevas a tu trabajo, no se dedican a eso, pero si dando consejos con los que mantener los pies en la tierra y no perderte. Muchas gracias papá, mamá y por supuesto Marta; esto jamás habría sido posible sin vosotros.

Toca ahora recordar a la familia que tienes la oportunidad de elegir, ¿o debería decir más bien a la que tienes el placer de elegir?; ellos son los mejores y por ellos ha merecido la pena toda esta aventura. Es ahora cuando me doy cuenta de que tengo que hacer esto detenidamente o Beltrán Labrador me estará recordando el resto de mi vida que: "los agradecimientos del TFM los hiciste mal, mira que no meter a éste/a..."; parece más difícil que todo el resto de la memoria, ¡menuda presión!.

Pero bueno, hagamos las cosas por orden y con buena letra. Los primeros a los que quiero recordar son los miembros de mi "Concilio de Telecos": Belt, Alex y Gator, sabéis que sois uno de los pilares de mi vida, y si no lo sabéis ya os lo digo yo, de verdad muchas gracias por todo; os quiero chicos.

De segundos pero no por ello menos importantes, ¡muchas gracias a la "Máster class". Han demostrado ser claves en el día a día y los mejores amigos que se puede tener. A los que han estado ahí desde el grado: Javi, Carlos, Sito, Merce, Pablo, Pabli (esto parece un trabalenguas) y Palomo. Y también a las nuevas adquisiciones del grupo: Esther, Bea y Sergio. Quién me iba a decir a mi que me quedaban gente tan espectacular por conocer al final de la universidad... Muchas gracias a todos por aguantarme y por querer ser mis amigos, ¡no os hacéis una idea de la suerte que he tenido!

Quiero hacer una especial mención a Víctor, el trabajador intrépido, al que quedarse en España se le hace pequeño. Mi inconsumible compañero de prácticas. Muchas gracias por estar ahí y que sepas que te voy a echar de menos; ve preparando mi habitación en Dublín.

A mis amigos del colegio: Carazo, Magro y Blas. Esos viernes de "Tierra tacos" me han dado la vida. Y a Jorge, siempre atento y siempre dispuesto a apoyar aunque no supiera como hacerlo; salvo que necesitara un ajuste, ahí siempre es el mejor.

Por último a mi tutor, Doroteo Torre. Que ha confiado en mi por segunda vez para acabar un proyecto con él y siempre ha estado ahí para ayudarme cuando lo he necesitado.

De corazón, muchas gracias a todos.

Índice general

Índice de figuras IX

Índice de tablas X

1. Introducción 1

1.1. Motivación . . . 1

1.2. Objetivos . . . 2

1.3. Organización de la memoria . . . 2

2. Estado del Arte 5

2.1. Redes neuronales . . . 5

2.1.1. *Dense Neural Network* o DNN . . . 6

2.1.2. Redes *Long Short-Term Memory* o LSTM . . . 6

2.1.3. *Convolutional Neural Network* o CNN . . . 8

2.2. *Mean Average Precision* . . . 8

2.3. *Embeddings* y antecedentes . . . 10

2.4. Modelos de atención . . . 11

2.5. DCASE *challenge* 2018 . . . 12

3. Base de datos: Google AUDIOSET 15

3.1. Estructura . . . 15

3.2. Ontología . . . 16

3.2.1. Elección de etiquetas . . . 18

3.3. División de la base de datos . . . 18

4. Diseño y desarrollo 21

4.1. Sistema . . . 21

4.2. Redes Neuronales . . . 22

4.2.1. Modelos DNN . . . 23

4.2.2. Modelos LSTM . . . 24

4.2.3. Modelos CNN . . . 24

5. Pruebas y resultados **27**

5.1. Resultados con la base de datos *balanced* 27

5.2. Resultados con la base de datos Sub–*unbalanced* 29

6. Conclusiones y trabajo futuro **31**

6.1. Conclusiones 31

6.2. Trabajo futuro 32

Glosario **33**

Bibliografía **35**

Índice de figuras

2.1. Estructura general de un modelo de *dense neural network* o DNN. Donde **n** indica el número de dimensiones de la capa de entrada, **m** el número de capas ocultas y **u** el tamaño final de la capa de salida 6

2.2. Estructura de una celda de memoria de un modelo LSTM, σ representa a la función sigmoid propia de las puertas donde está localizada, **g** y **h** son las funciones de activación tangente hiperbólica propia de la entada y la salida. 7

2.3. Modelo CNN 9

2.4. Función tipo resultado de enfrentar las medias *precision* y *recall* para una clase dada. 10

2.5. Modelo simplificado de extracción de embeddings, mediante la técnica de *bottleneck*, en una red DNN. 11

2.6. Esquema del funcionamiento de un modelo de atención 12

2.7. Clases entre las que clasificar en el *challenge* DCASE 2018 12

3.1. Comparativa entre un espectograma y una matriz de *embeddings* tipo. Se puede apreciar claramente la diferencia de tamaño de los datos; donde en el espectograma se tiene una gran cantidad de valores, tanto en el eje temporal como en el frecuencial (257x626), en el de *embeddings* tan solo se tiene una matriz de 10x128. 16

3.2. Jerarquía de la ontología de la base da datos AUDIOSET 17

3.3. Ejemplo de estructura del archivo JSON 17

4.1. Esquema del sistema implementado en el proyecto. La primera fase corresponde a la red adaptada de [2] y la segunda corresponde a al clasificador implementado en el proyecto. 21

4.2. Ejemplo de salida del sistema desarrollado en el proyecto. 22

5.1. Comparativa de coste en validación en función de la cantidad de parámetros entrenables de los modelos entrenados con la base de datos *balanced*. 29

5.2. Ampliación de la sección cercana al umbral de la figura 5.1 29

5.3. Comparativa de coste en validación en función de la cantidad de parámetros entrenables de los modelos entrenados con la base de datos Sub–*unbalanced* . . . 30

5.4. Ampliación de la sección cercana al umbral de la figura 5.3 30

Índice de tablas

3.1. Distribución de etiquetas usadas en las bases de datos 18

5.1. Resultados mAP obtenidos para los diferentes modelos estudiados, entrenados con la base de datos *balanced*. 28

5.2. Resultados mAP obtenidos para los diferentes modelos estudiados, entrenado con el subconjunto de la base de datos Sub–*unbalanced* 30

1 Introducción

1.1. Motivación

Este proyecto versa sobre la aplicación de técnicas con redes neuronales profundas en la tarea de reconocimiento de eventos de audio, con el fin de clasificar el origen que produce a los mismos. Con ello se pretende desarrollar un sistema capaz de identificar de manera satisfactoria un subconjunto de los diferentes tipos de audio de los que dispone la base de datos de Google AUDIOSET [1].

Esta base de datos es relativamente nueva. Contiene una cantidad de audios desorbitada ya que provienen del material almacenado en los servidores de YouTube, que no para de crecer día a día. Si bien no se puede hacer uso libre de todos los datos disponibles, muchos solo se pueden utilizar en el propio reproductor de Youtube, el total de los que si se puede hacer uso libre es de más de 2 millones. El venir de esta plataforma hace a la base de datos no solo muy accesible a los usuarios interesados, si no que también la hace completamente gratuita; estas dos caracterśticas permiten que destaque y de hecho es utilizada hoy en día en múltiples *challenges* científicos. Google permite a los interesados hacerse con estos datos de dos maneras diferentes: descargarlos en formato wav o en formato tfrecord; teniendo en cuenta que siempre serán archivos de 10s de duración.

Cabe destacar que este material está ya etiquetado aunque se advierte que no todas las etiquetas están colocadas con el mismo nivel de convicción. Por ello se tendrá que llevar a cabo un estudio previo de la propia base de datos para lograr tener las etiquetas óptimas para el proyecto. Las clases finales serán las que en la documentación de AUDIOSET se establecen como clases principales de la ontología, lo que no quiere decir que sean estas con las que vienen etiquetados los audios; estas clases son: *Human sounds*, *Source-ambiguous sounds*, *Animal*, *Sounds of things*, *Music*, *Natural sounds* y *Channel, evironment and background.*

La manera en la que se plantea el sistema de este trabajo es mediante el aprovechamiento de redes ya implementadas en otros estudios, como es la del caso [2]; junto con creaciones propias basadas en las ya muy extendidas *Dense neural networks* o DNN, las *Long Short–Term Memory* o LSTM [3] y las *Convolutional Neural Netwrok* o CNN.

Las primeras permiten introducir los datos al sistema en forma de vector, estos deberán ser porcesados por la red con el fin de lograr la salida desdeada. Se suelen utilizar modelos denominados *fully conected* ya que son los más sencillos de implementar y serán la base sobre la que se partirá en este proyecto.

Las segundas, basan su funcionamiento en una unidad fundamental más compleja que las anteriores, siendo una evolución de las propias DNN. Con ellas se introduce un nuevo concepto llamado memoria, que permite al sistema tener en cuenta datos anteriores a la hora de entrenar. Todo esto supone una cambio muy significativo a la hora de tratar con datos en los que la variable temporal juega un papel importante.

Finalmente, las ya mencionadas CNN. Este estilo de red neuronal permite el uso de imágenes como entrada sin necesidad de convertirlas en un vector de datos, gracias a esto se disminuye de manera muy significativa la cantidad de parámetros a entrenar, facilitando así el proceso de entrenamiento. Este tipo de redes tiene un peso fundamental en los problemas enfocados en tratamiento de imágenes; sin embargo, en este trabajo se intentarán aplicar a las matrices de datos en las que están representados los audios de AUDIOSET.

1.2. Objetivos

Con este proyecto se busca generar un sistema completo que sea capaz de tomar como entrada cualquier tipo audio; transformándolo, de ser necesario, al formato con el que están almacenados los datos de AUDIOSET. El resultado de este proceso se utilizará como entrada a los clasificadores pertinentes; con el fin de lograr detectar cuales, de las clases de las que consta la base de datos, se encuentran en el audio tratado.

Para conseguirlo se deberá adaptar el sistema implementado en [2] y desarrollar diferentes modelos de redes neuronales con el fin de lograr el mejor resultado posible, tratando de mejorar el estado del arte en este campo. Los resultados obtenidos deberán ser comparados mediante el uso de la medida de rendimiento *mean average precition* (mAP) con la que Google y buena parte de la comunidad científica trabajan en estos casos. También será importante realizar un estudio y preprocesado de la base de datos para adecuarla a las necesidades el proyecto.

1.3. Organización de la memoria

Esta memoria se ha estructurado por capítulos, quedando en un total de 6 bloques fundamentales. Con ellos se trata de ofrecer al lector, de la manera más clara posible, toda la información necesaria para entender el trabajo realizado. El documento queda estructurado de la siguiente manera:

- **Introducción**: como acaban de leer, en este capítulo se han expuesto la motivación y los objetivos más remarcables del proyecto.
- **Estado del arte**: en este capítulo se mostrarán las diferentes herramientas que se suelen utilizan a la hora de resolver problemas de clasificación de audio. Se hará especial hincapié en las redes neuronales, explicándolas de manera concisa ya que son la herramienta fundamental de este proyecto, y se analizarán los sitemas ya publicados que obtengan buenos resultados en comparación a los de Google.
- **Base de datos**: a lo largo de este capítulo se realizará un estudio de la estructura que sigue la base de datos AUDIOSET y de sus características principales. Se dará mayor atención

a la ontología de la base de datos, destacando su jerarquía de etiquetas, su manera de representarlas y la distribución de las mismas dentro de la base de datos.

- **Diseño y desarrollo del trabajo**: este bloque expondrá el sistema total implementado, desde que se toma un audio hasta la calsificación final. Se dará especial atención a todos los pasos que se han tenido que llevar a cabo para poder diseñar y entrenar las redes neuronales utilizadas.
- **Pruebas y resultados**: esta sección expondrá los mejores resultados obtenidos para todos los modelos implementados comparándolos con los obtenidos por otros estudios así como entre si mismos.
- **Conclusiones y trabajo a futuro**: se enuncian las conclusiones finales del trabajo de fin de máster, a su vez se presentan las posibles líneas de trabajo futuro que harán más completo el proyecto.

2

Estado del Arte

2.1. Redes neuronales

Las redes neuronales son, hoy en día, el máximo exponente en técnicas de *machine learning*. Se basan en una interpretación muy simplificada del comportamiento del cerebro humano y han demostrado ser capaces, bajo las condiciones adecuadas, de cumplir tareas que hasta su puesta en marcha eran impensables.

El fin fundamental de esta técinca es la identificación de patrones ocultos en los datos. Estos patrones hacen que la red sea capaz de "tomar decisiones" en base a ellos y como consecuencia extraer conclusiones en forma de resultados. Para poder hacer todo esto las redes cuentan con diferentes tipos de arquitectura que tienen diferentes estilos y profundidades, de las que se hablará más adelante, y de una serie de parámetros que puedan cambiar de valor y de forma, en función del tipo de red que se use, conforme procesa los datos; a esos valores se les llama pesos y al proceso de cambio se le llama entrenamiento.

El entrenamiento de una red basa su funcionamiento en el algoritmo *backpropagation* [4] el cual utiliza los resultados de la red y los compara con lo que el usuario busca, el *groundtruth*, para modificar los parámetros que se mencionaron antes. Esta comparación se lleva a cabo con la función de coste, una operación matemática que devuelve un valor que deberemos minimizar. En un entrenamiento ideal los resultados de esta función de coste tendrán forma de exponencial negativa según se avance en el entrnamiento; cuando alcanza su mínimo, si se ha dado un entrenamiento correcto, se dice que la red ha convergido y será ahí cuando concluya el entrenamiento. Un problema que puede surgir es que la red no converja en el mínimo absoluto de la regresión de la función de coste, en estos casos no se habrá alcanzado la mejor optimización del entrenamiento y la se debrán ajustar mejor sus parámetros.

Es importante destacar un último parámetro común a todos los estilos de red neurnal, la función de activación. Por lo general son operaciones no lineales que aportan una transformación a los valores internos de la red e influyen notablemente en los resultados del entrenamiento. Esas operaciones no lineales son las que permiten a las redes modelar relaciones no lineales, lo que las diferencian de modelos lineales convencionales.

2.1.1. *Dense Neural Network* o DNN

Este tipo de red constituye el estilo de red neuronal más simple de todos. Basa su arquitectura en la relación de sus unidades fundamentales, las neuronas, y su organización en capas (figura 2.1). Cada neurona de cada capa está conectada a todas las neuronas de las capas siguientes mediante una operación lineal, los pesos, y las funciones de activación mencionadas anteriormente. Los primeros forman una estructura en forma de matriz, una por cada intersección entre capas, y modifican su valor durante el entrenamiento definiendo cuanta importancia aporta el valor de las neuronas de una capa a las neuronas de la capa siguiente, para ello multiplican su valor al de la neurona. Las segundas se aplican al resultado de sumar todos los resultados de las multiplicaciones de la capa anterior con sus respectivos pesos.

Por otro lado, existe otro parámetro denominado bias. Es una constante, generalmente de valor 1, que se aplica a cada capa oculta y sobre la que también intervienen los pesos vistos antes, por lo que también puede considerarse como un valor entrenable de la red. Tiene como función actuar como regulador del umbral que rige a las funciones de activación. Esto quiere decir que si una función de activación tiene su umbral de actuación en 0 y se le añade el bias con valor 1, este umbral pasará a estar en 1, modificando sustancialmente el valor final de la neurona.

2.1.2. Redes *Long Short-Term Memory* o LSTM

Uno de los principales problemas que se presentan con el uso de DNN es que no son capaces de tener en cuenta el factor temporal de los datos a la hora de entrenar. Por eso, con el tiempo, surgieron nuevos modelos que trataban de incluir este parámetro en la ecuación; uno de los

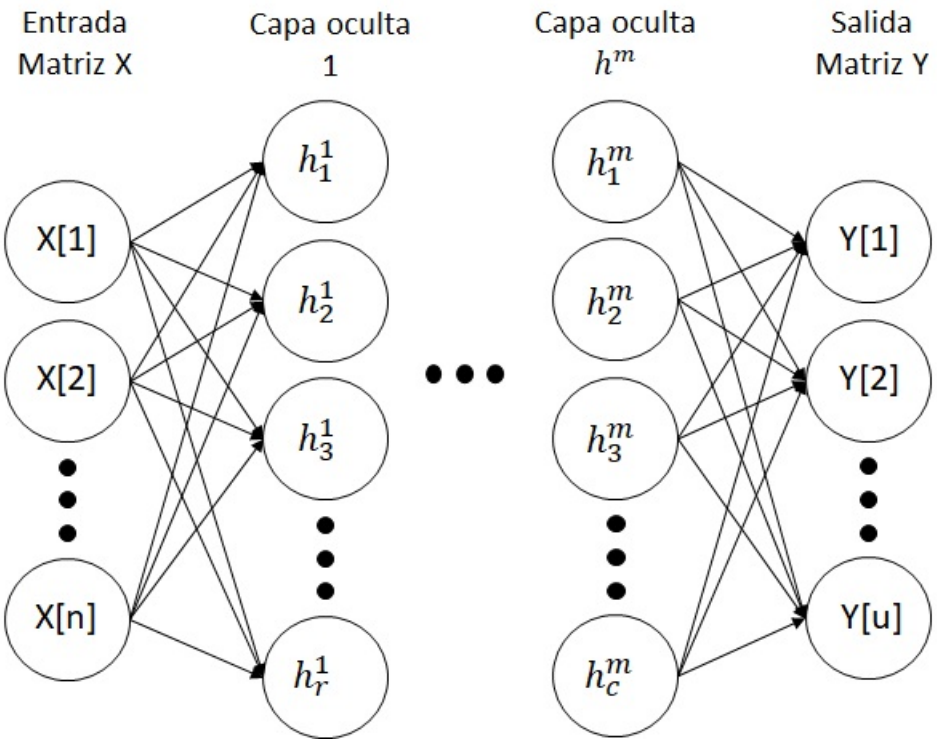


Figura 2.1: Estructura general de un modelo de *dense neural network* o DNN. Donde **n** indica el número de dimensiones de la capa de entrada, **m** el número de capas ocultas y **u** el tamaño final de la capa de salida

primeros en ser reconocido fue el modelo *Reccurrent Neural Network* o RNN [5]. Por desgracia, estas reportaron un nuevo inconveniente denominado desvanecimiento de gradiente. Este problema consistía en que con el paso del entrenamiento las funciones de activación de las puertas iban tendiendo a dar resultados cercanos a 0, evitando un aprendizaje productivo ya que el gradiente de la red se hacía demasiado pequeño para entrenar. Sin embargo, en el que nos centraremos es el que supo aprovechar al máximo las cualidades temporales de los datos, los modelos LSTM, que supieron suplir el problema.

La principal diferencia entre este nuevo modelo y las DNN es la estructura de su unidad fundamental. Donde antes teníamos una neurona con un valor adquirido de la combinación de valores de la capa anterior, ahora se tiene una estructura llamada *memory cell* o celda de memoria (figura 2.2). Es una estructura mucho más completa que consta de varias entradas caracterizadas por utilizar su propia salida en combinación con los nuevos datos de entrada para calcular nuevos resultados, es decir, tiene memoria.

Cómo se ve en la figura 2.2 la celda tiene 4 "entradas": la entrada propiamente dicha (abajo) y las 3 *gates* o puertas de la celda (laterales). Todas ellas toman como entrada a los nuevos datos y los antiguos; sin embargo, no todas se comportan de igual manera y cada una aporta al resultado final lo que la propia celda considera mediante un peso, similar en funcionamiento a los que aparecen en las DNN.

Más en profundidad, hay tres tipos de *gates*, cada una con una función diferenciada que actúa sobre el estado de la neurona. Este estado es conocido como la memoria de las LSTM y es la que mantiene o desecha la información a lo largo del proceso de entrenamiento en función de la actuación de las puertas antes mencionadas, que son:

- **Puerta de entrada**: es la encargada de actualizar el estado de la celda. Mediante la función *sigmoid* que la representa, toma un valor entre 0, que será clasificado como poco

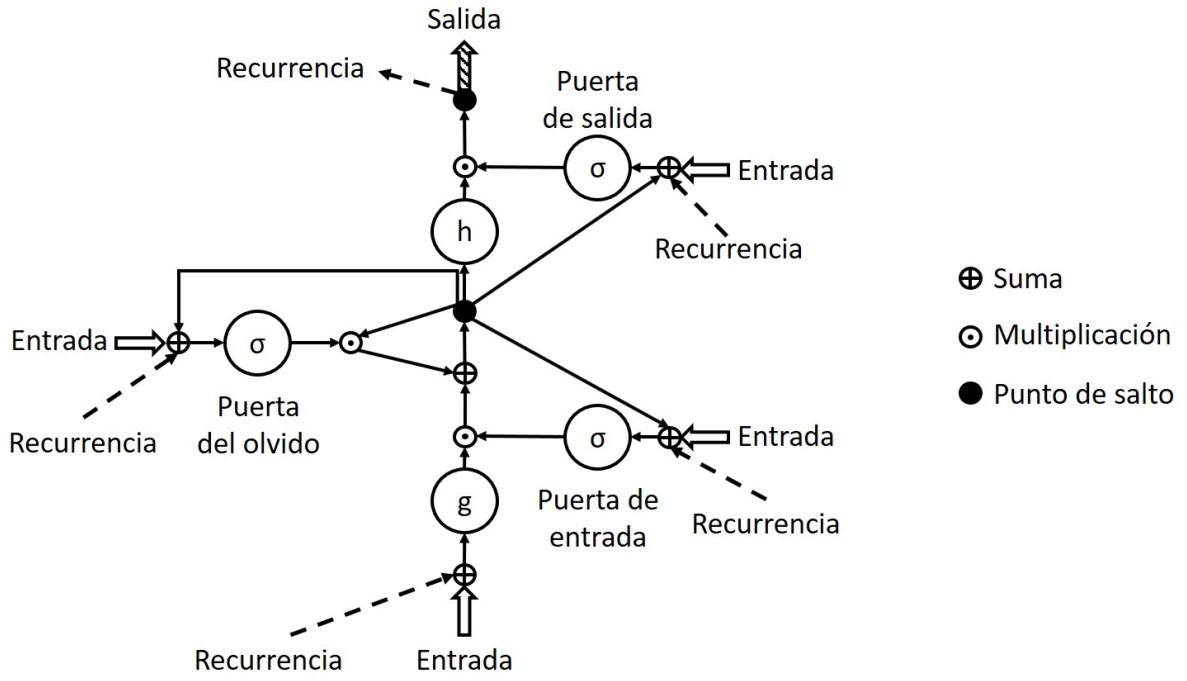


Figura 2.2: Estructura de una celda de memoria de un modelo LSTM, σ representa a la función sigmoid propia de las puertas donde está localizada, **g** y **h** son las funciones de activación tangente hiperbólica propia de la entada y la salida.

imporante, y 1, que será muy importante. Este valor se multiplicará por la salida de la función tangente hiperbólica o *tanh* de la entrada decidiendo de esta manera que información es importante o cual no.

- **Puerta del olvido**: esta es la encargada de decidir que información debe ser mantenida o desechada dentro de la neurona. Para ello toma la información aportada por la entrada (información actual) y por la salida (información pasada) y las pasa por su función *sigmoid* que devolverá un valor entre 0 y 1; siendo 0 olvidar y 1 mantener.
- **Puerta de salida**: esta última puerta es la que decide como será el próximo estado de la neurona. Para ello recoge el valor actual del estado devuelto por la *tanh* de salida y lo multiplica por el resultado de su porpia *sigmoid* siendo este resultado el nuevo estado de la neurona.

2.1.3. *Convolutional Neural Network* o CNN

Al igual que en el caso de las redes LSTM, este nuevo modelo de redes surge de la falta de rendimiento que tienen las DNN; en este caso para el procesamiento de imágenes. Con este procedimiento se puede introducir a una red una imagen, o una matriz de datos, directamente sin necesidad de "estirar" los valores en un vector unidimensional, como si ocurría en los casos anteriores. Al igual que en el modelo anterior, la principal diferencia existente entre este nuevo modelo y las DNN convencionales son la estructura de su unidad fundamental y, en este caso, la estructura de pesos.

Ahora las neuronas (figura 2.3a) son un conjunto de valores que forman una estructura bidimensional, es decir, una matriz. Esto permite a la red ir extrayendo características diferentes según sea la complejidad del modelo. Por poner un ejemplo, es bastante común que la primera capa extraiga bordes, la segunda estructuras, etc.

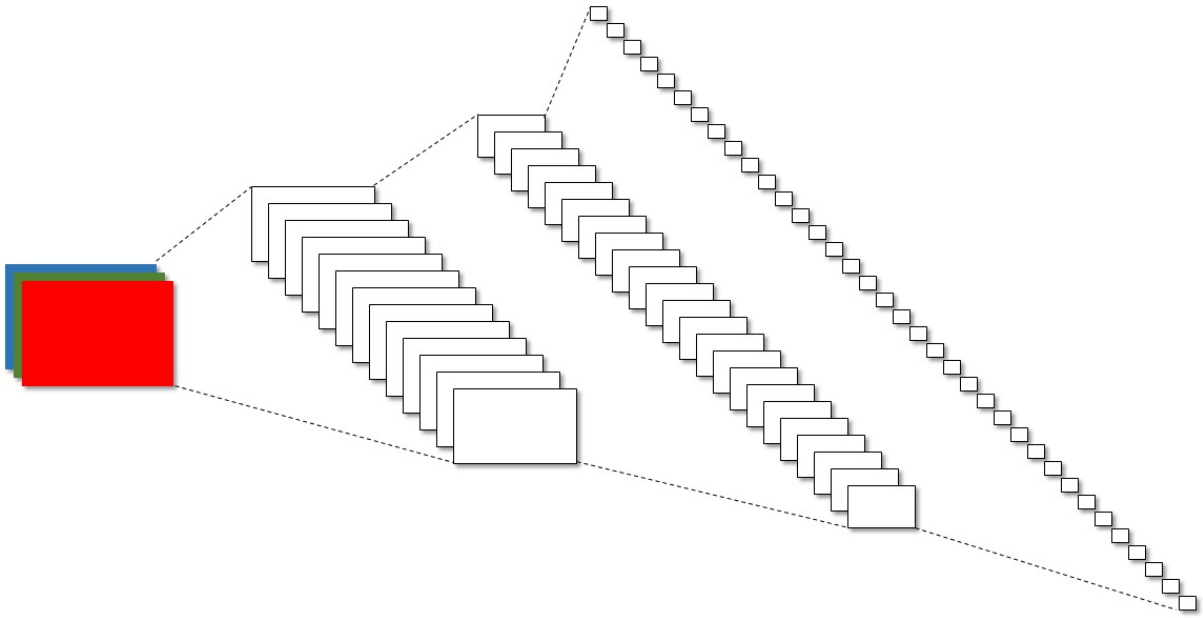
Por otro lado, los pesos también tienen forma de matriz aunque de un tamaño muy inferior a las neuronas, normalmente 3x3 o 5x5 y se le suele llamar *kernel*. Se pueden definir como filtros más que pesos ya que su función es recorrer las matrices de datos resultando en otra matriz ya tratada por el mismo. Existen diferentes aproximaciones a la hora de aplicar estos filtros en función del número de dimensiones de los datos, pero la más habitual (figura 2.3b) es utilizar el mismo filtro para todas las capas y elegir el número de filtros diferentes por capa.

Es importante destacar que entre las capas convolucionales se suele utilizar una técnica llamada *MaxPooling*. Esta consiste en disminuir el tamaño de las matrices de un capa por un factor, típicamente 2x2. Con esto se pretende reducir la dimensionalidad de la red sin perder información, concentrándola, haciendo en definitiva más fácil el entrenamiento al reducir significativamente el número de parámetros a entrenar antes de llegar a las capas DNN.

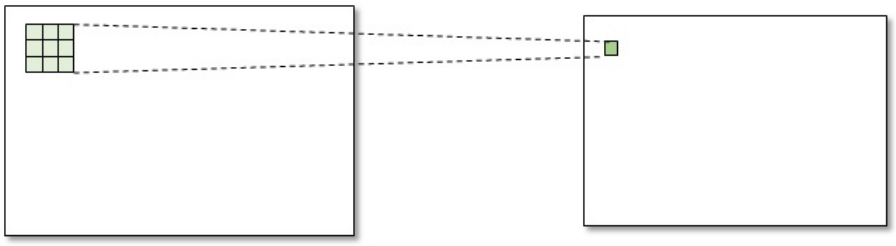
2.2. *Mean Average Precision*

En sistemas enfocados en clasificaión o detencción de eventos acústicos se suele utilizar la medida de rendimiento *mean average precition* o mAP. Este tipo de medida, como es habitual en problemas de clasificación, se pondera entre valores de 0 a 1 y cuanto más alto sea el resultado mejor será el rendimiento del sitema.

Existen varias maneras de calcular este rendimiento, todas ellas tienen el mismo objetivo, pero no dan los mismos resultados. Por lo general el método a utilizar se elige en función del que se utiliza en los estudios contra los que se quieren comparar los resultados, de esta manera el usuario sabe que son equivalentes.



(a) Modelo general de una red CNN



(b) Funcionamiento habitual de un filtro en una capa convolucional, al aplicar un *kernel* a la matriz de datos se calcula un único valor de la matriz de datos de la capa siguiente.

Figura 2.3: Modelo CNN

Como el propio nombre indica, esta medida de rendimiento consiste en calcular el valor AP por cada clase de estudio y posteriormente hacer la media de esos resultados. Este valor se define en [1] como el área bajo la curva que se obtiene al enfrentar las medidas *precition* y *recall*. El cálculo del AP se muestra en la fórmula 2.1 donde f(r) es la función formada (figura 2.4) por esas medidas antes mencionadas.

$$AP = \int_0^1 f(r)dr \tag{2.1}$$

La medida *precision* se define (fŕomula 2.2 izquierda) como el número de datos que se han clasificado correctamente (*TruePositive* o TP), según la clase que se tome como positiva, dividido por el número total de datos que se han clasificado como positivos, lo sean o no (TP en conjunto con los *FalsePositive* o FP). Por otro lado el *recall* se define (fórmula 2.2 derecha) como el número de datos clasificados correctamente como positivos (TP) entre el número total de datos de la clase a la que se toma como positiva (TP en conjunto con los *FalseNegative* o FN).

$$p = \frac{TP}{TP + FP} \qquad\qquad r = \frac{TP}{TP + FN} \tag{2.2}$$

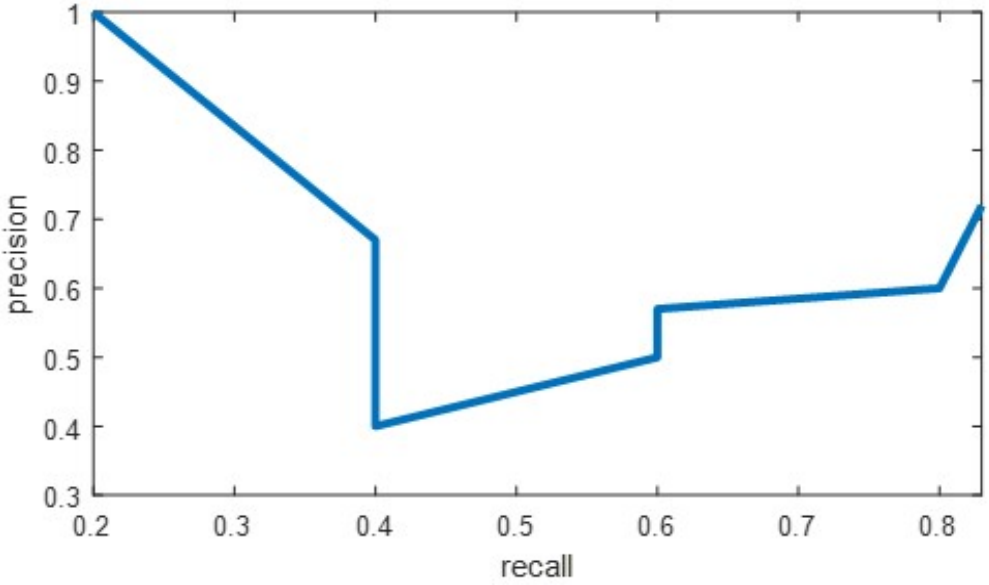


Figura 2.4: Función tipo resultado de enfrentar las medias *precision* y *recall* para una clase dada.

2.3. *Embeddings* y antecedentes

Tradicionalmente en el *machine learning* aplicado a audio, se ha utilizado una estructura denominada *i-vector*. Esta era la reprensentación condensada de un audio de tal manera que se conseguía resumir la información de un audio sin perder prácticamente información útil. El problema con ellos es que son generados mediante un modelo basado en máquinas GMM-UBM, que son entrenadas con un aprendizaje supervisado generativo. Este tipo de entrenamiento lo que hacen es modelar las distribuciones de probabilidad de las clases, de las que consta el problema que se trata, por separado sin optimizar la separabilidad entre las mismas. [6]

Con la intención de mejorar los resultados obtenidos con esas máquinas GMM-UBM, surge la estructura de información denominada *embeddings*. Al contrario que los anteriores vectores, esta nueva representación de los datos se consigue tras extraer los valores que contiene una capa DNN.

En la figura 2.5 se puede ver una respresentación muy simplificada del funcionamiento de estos sistemas. Por lo general se introduce como entrada la cantidad de datos que se quieren "resumir" y se elige un capa intermedia de la red, a la que se llamará *bottleneck*, para reducir su dimensionalidad significativamente con respecto a sus capas adyacentes y extraer de ella sus valores finales.

En [2] se implementan una serie de redes que buscan aprovechar las virtudes de las CNN para reconocimiento de imágenes. Sin embargo, también hacen un estudio del tema del que trata este proyecto, extrayendo los *emeddings* de la base AUDIOSET; para ello extraen los vectores de datos de la mejor versión de la red ResNet-50 [7] que entrenan. Para lograrlo modificaron el modelo original en varios puntos:

- Se modificó el desplazamiento de filtros de 2 a 1 de la primera capa CNN con filtro 7x7.
- Se modificó el estilo de pooling utilizado, utilizando el llamado *Average Pool* en un intento de reflejar el cambio de valor producido en las activaciones.

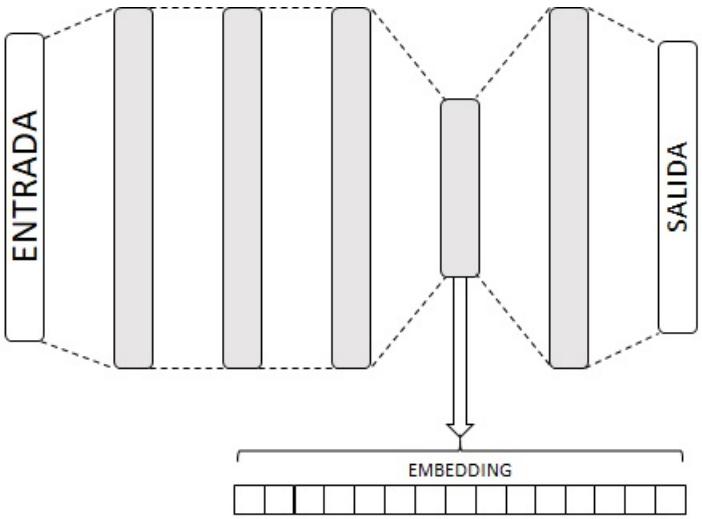


Figura 2.5: Modelo simplificado de extracción de embeddings, mediante la técnica de *bottleneck*, en una red DNN.

La principal consecuencia de todo lo anterior es que la nueva red contiene un total de 30 millones de parámetros entrenados, en comparación con los 26 originales. Aplicando esto sobre la base de datos AUDIOSET [1] y destacando que operan sobre las 527 clases de la jerarquía más baja de la base de datos, la cual se explicará más adelante, consiguen un resultado de mAP 0.314.

2.4. Modelos de atención

Como se ha visto hasta ahora, el fin de este proyecto es la clasificación de audios o, dicho de otra manera, la detección de los eventos acústicos que ocurren durante el segmento de audio que se trata. Esto es relevante porque recientemente han surgido nuevas aproximaciones a este problema que, además de extraer las clasificaciones pertinentes, tratan de localizar en que momento del audio ocurren los eventos detectados; los llamados modelos de atención. Esta herramienta no ha sido usada durante el trabajo dado que requiere de una *softmax* como función de activación de salida, esto haría que las etiquetas de salida del sistema tuvieran que estar formadas por todas las combinaciones posibles entre las elegidas para este proyecto. Esto, si bien es posible, dificultaría enormemente la tarea. Sin embargo, aunque no se use en el proyecto, resulta interesante dar unas nociones de la técnica para entender más en profundidad en que punto del problema se encuentra la comunidad investigadora.

En [8] se define como una herramienta que tiene como fin aprender a focalizarse en las partes más relevantes de una secuencia de datos de entrada para cada salida del sistema. Para ello se aprovecha de modelos que estén dispuestos en una estructura llamada *encoder*; esto quiere decir que la red irá disminuyendo el tamaño de sus capas paulatinamente hasta lograr llegar a una en la que los datos queden muy resumidos, el *bottleneck*, llamado codificador. Una vez se llega al codificador (figura 2.6), se asigna un peso a cada una de las unidades ocultas que lo componen. Posteriormente el mecanismo de atención "alimenta" con un vector de contexto c_t, extraído del codificador, al decodificador para hacer una predicción. El vector de contexto se deberá calcular de la siguiente manera:

$$c_t = \alpha_t.h \tag{2.3}$$

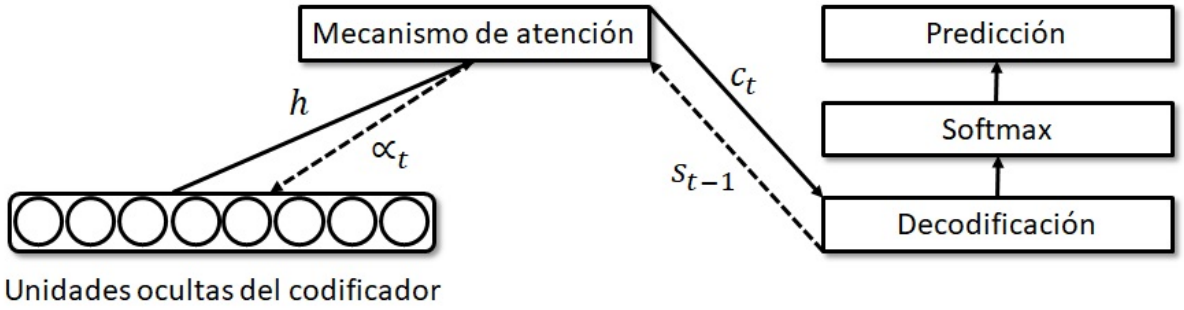


Figura 2.6: Esquema del funcionamiento de un modelo de atención

donde α_t es el vector de atención calculado en el instante t y h es el vector que representa los valores de las unidades ocultas del codificador. Por otro lado ese vector de atención será calculado de la siguiente manera:

$$\alpha_t = softmax(score(s_{t-1}, h)) \tag{2.4}$$

donde s_{t-1} corresponde al estado en el que se encuentra el decodificador. La función *score* puede tomar diferentes formas, algunas de las más extendidas son [9]: *multi-layer perceptron* o MLP, *multi-head attention*, etc.

Sin embargo, recientemente han sido publicados estudios que hacen uso de esta herramienta para problemas de clasificación, un claro ejemplo se da en [10]. En este estudio logran producir una red DNN basada en modelos de atención que obtiene un mAP de 0.327. Con esto logran desbancar los resultados obtenidos por Google para el mismo problema, clasificar eventos acústicos diferenciando entre las 527 clases que ofrece AUDIOSET, que lograron un 0.314 de mAP.

2.5. DCASE *challenge* 2018

En esta competición se propuso a los participantes el reto de crear un sistema capaz de reconocer eventos acústicos haciéndo uso de un subconjunto de la base de datos AUDIOSET. Concretamente, se debía tratar con 41 clases diferentes que no siguen una jerarquía clara dentro de la ontología de la base de datos (figura 2.7).

- Tearing
- Bus
- Shatter
- Gunshot, gunfire
- Fireworks
- Writing
- Computer keyboard
- Scissors
- Microwave oven
- Keys jangling
- Drawer open or close
- Squeak
- Knock
- Telephone
- Saxophone
- Oboe
- Flute
- Clarinet
- Acoustic guitar
- Tambourine
- Glockenspiel
- Gong
- Snare drum
- Bass drum
- Hi-hat
- Electric piano
- Harmonica
- Trumpet
- Violin, fiddle
- Double bass
- Cello
- Chime
- Cough
- Laughter
- Applause
- Finger snapping
- Fart
- Burping, eructation
- Cowbell
- Bark
- Meow

Figura 2.7: Clases entre las que clasificar en el *challenge* DCASE 2018

Como parte del *challenge* se proporciona a los usuarios la base de datos con la que realizar los experimentos. Esta consiste de dos partes diferenciasdas, el *train set* y el *test set*. El primero consta de 9500 muestras con una distribución desbalanceada de categorías mientras que el segundo está compuesto por 1600 muestras con una distribución similar a la base de entrenamiento.

Para evaluar los resultados se utiliza el mAP@3, esta variante del mAP se calcula considerando buena una decisión en la que se tienen en cuenta las tres mejores opciones dadas por el detector de eventos. Con esta medida en el estudio obtienen muy buenos resultados siendo el mejor de 0.9538 [11], con lo que supera con creces el *baseline* de 0.6943 con el que se comparaba. Sin embargo, esta medida de rendimiento no es la misma que se utiliza en este proyecto con lo que los resultados no son comparables.

3

Base de datos: Google AUDIOSET

Esta base de datos fue creada en 2017 por el equipo de Google Sound Understanding. Está principalmente generada para problemas de clasificación y/o detección del origen de un audio. Es completamente gratuita y la compañía permite hacer uso de ella desde su página web [1].

La propia Google ha puesto a disposición de los usuarios dos maneras de hacerse con los datos: un script que descarga directamente de Youtube los segmentos de audio o unos archivos comprimidos que contienen la base de datos en un formato resumido llamado *embeddings* (vistos en el capítulo 2.3), hechos por la propia compañía.

3.1. Estructura

La base de datos está constituída por más de 2 millones de segmentos de audio de 10s extraídos de vídeos de Youtube. Estos audios están muestreados a 16KHz y es algo que habrá que tener en cuenta en la implementación final del proyecto.

En un inicio se pensó en introducir los datos al sistema en su formato espectograma, pero esto a la larga quedó demostrado que no era muy conveniente dada la enorme cantidad de parámetros de entrada que se tendrían. Por ello, en el proyecto se han utilizado los datos en su variante *embeddigns*; esto se hizo así gracias a que Google subió a internet no solo los datos en este formato si no también los documentos que permitían hacer esta transformación; con ello ofrecen a los usuarios la posibilidad de desarrollar sus propias bases de datos en el mismo formato y poder hacer pruebas consistentes.

En el caso de esta base de datos, estos *embeddings* consisten en resumir cada segmento de audio en un total de 10 vectores, uno por cada segundo de datos, de 128 valores cada uno. Con esto se disminuye significativamente el número de muestras por audio; donde antes había 160882 muestras, con la matriz de espectograma, ahora hay tan solo 1280 (figura 3.1).

Cabe destacar que los datos están almacenados en el formato .tfrecord, exclusivo de TensorFlow [12]. Con este formato se consigue almacenar todo de una manera muy comrpimida; sin embargo, resulta muy incómodo a la hora de trabajr con ellos. Por esta razón se utilizó parte del código desarrollado previamente por Javier Darna en [13] en el que transformaba los datos de este formato a matrices de numpy [14], una por cada segmento de audio.

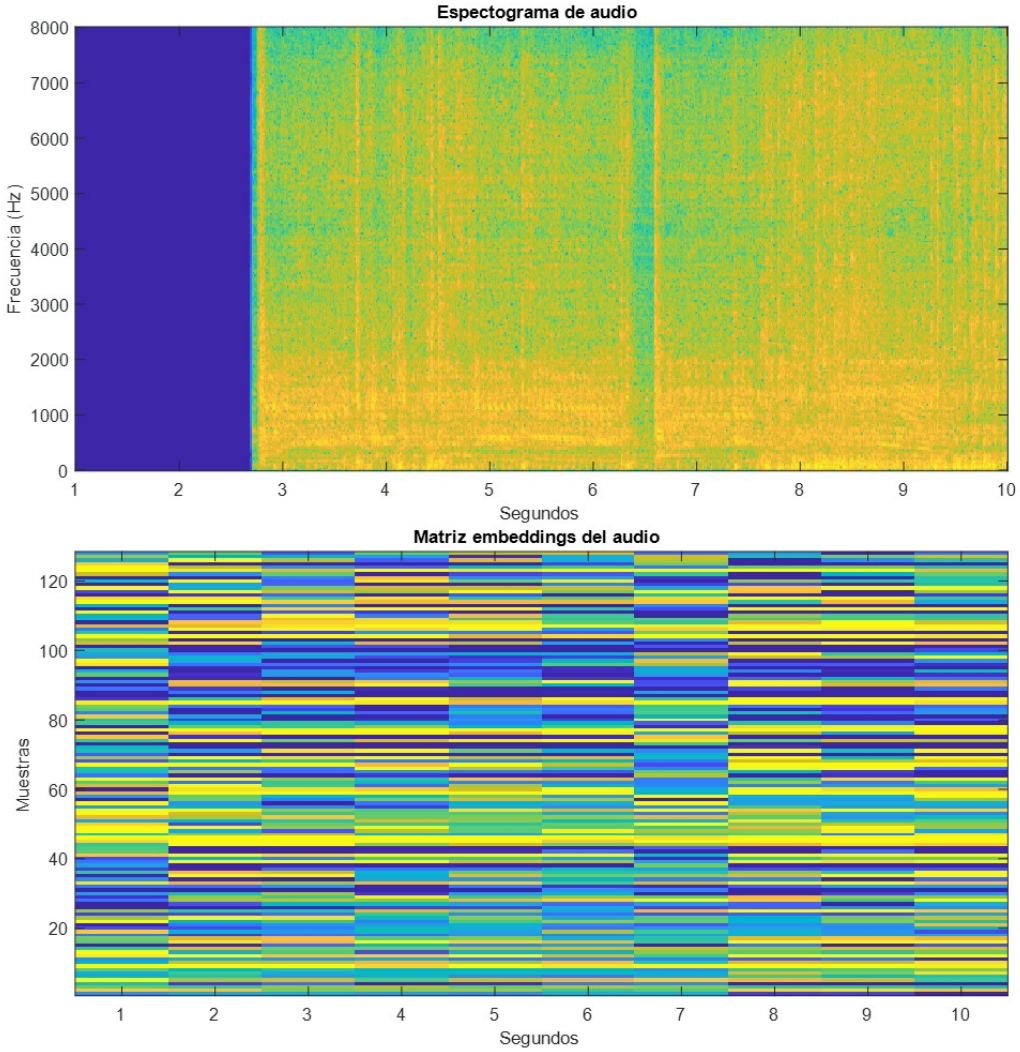


Figura 3.1: Comparativa entre un espectograma y una matriz de *embeddings* tipo. Se puede apreciar claramente la diferencia de tamaño de los datos; donde en el espectograma se tiene una gran cantidad de valores, tanto en el eje temporal como en el frecuencial (257x626), en el de *embeddings* tan solo se tiene una matriz de 10x128.

3.2. Ontología

Cada uno de los segmentos de audio está etiquetado siguiendo una estricta jerarquía, ver figura 3.2. Hay siete clases principales y cada una de ellas tiene varios "hijos" que a su vez tienen más etiquetas asociadas. En su versión más "desplegada" consta de 527 clases diferentes; todas ellas desarrolladas en un archivo csv llamado *class_ labels_ indices*, disponible en la propia página web. En él se relacionan todas estas etiquetas con un código que sirve para buscar la asociación con su etiqueta padre.

Para poder hacer esa relación es importante hablar del documento json que lo hace posible, el archivo *ontology*. Un ejemplo de como está tratado este documento se puede ver en la figura 3.3 y está estructurado de la siguiente manera:

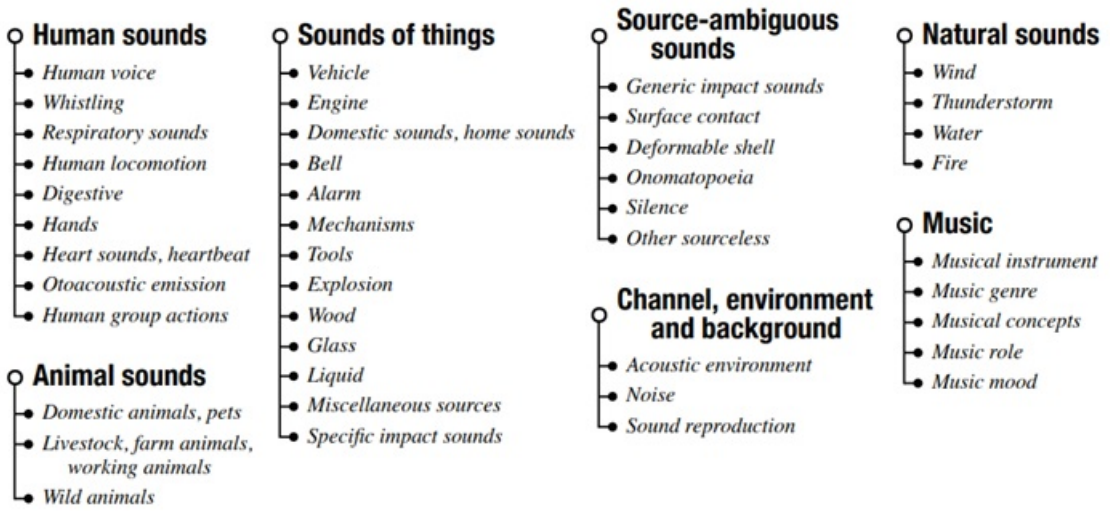


Figura 3.2: Jerarquía de la ontología de la base da datos AUDIOSET

- ***id***: código identificativo de la hace referencia a una etiqueta de las posibles en la base de datos. Está seleccionado con el uso de un *Knowledge Graph Machine ID* (MID) exclusivo para cada etiqueta. Un posible código sería: /m/0gy1t2s.
- ***name***: nombre que recibe la etiqueta. Por ejemplo , la etiqueta "*Bicycle bell*".
- ***description***: descripción más extensa de la etiqueta que por lo general ha sido extraída de webs como Wikipedia.
- ***citation_ uri***: enlace URL que envía al usuario a la web de la que se extrajo la descripción anterior.
- ***positive_ examples***: enlaces URL que envían al usuario a segmentos de vídeos de youtube donde se puede comprobar a que tipo de audio corresponde la etiqueta.
- ***childs_ ids***: lista de códigos MID de las etiquetas que están justo por debajo en la jerarquía de la ontología.
- ***restrictions***: de todas las categorías posibles hay algunas que entran en los subconjuntos "lista negra" o "abstractos". Estos casos son simplemente intermediarios en la jerarquía de la ontología pero no están pensados para ser usados como etiquetas.


```
{
    "id": "/m/0gy1t2s",
    "name": "Bicycle bell",
    "description": "The sound of a percussive signaling instrument mounted on a bicycle for warning pedestrians and other cyclists.",
    "citation_uri": "http://en.wikipedia.org/wiki/Bicycle_bell",
    "positive_examples": ["youtu.be/LK8j0Q4UU4g?start=10&end=20", youtu.be/s0F2kzW8quU?start=160&end=170","youtu.be/k7oGk-
            ozhKI?start=30&end=40","youtu.be/kTkG_z_GWFs?start=18&end=28","youtu.be/OY-jfOrL-_A?start=20&end=30",
            "youtu.be/vSPaTX3nq5A?start=40&end=50"],
    "child_ids": [],
    "restrictions": []
}
```


Figura 3.3: Ejemplo de estructura del archivo JSON

3.2.1. Elección de etiquetas

Como se ha dicho antes, las clases elegidas para este proyecto han sido las siete clases principales de la ontología: *Human sounds*, *Source-ambiguous sounds*, *Animal*, *Sounds of things*, *Music*, *Natural sounds* y *Channel, evironment and background*. Sin embargo, las clases no vienen asociadas a los datos de esta manera, sino que vienen definidas con los códigos MID mencionados anteriormente.

Con el fin de lograr las etiquetas adecuadas a las necesidades del trabajo, se creó un *script* que toma las clases asociadas a los audios y las devuelve en el formato de 7 clases requerido. Para ello hace uso del archivo JSON antes mencionado y, mediante una serie de bucles y búsquedas de cadenas, se recorre de manera inversa el árbol jerárquico de *ontology*, buscando a que etiquetas "padre" hacen referencia los códigos MID iniciales. Tras todo el proceso el resultado es un vector de 7 valores donde estarán a 1 las etiquetas que contiene el audio y a 0 las que no.

3.3. División de la base de datos

La Base de datos está separada en tres grupos fundamentales:

- ***Evaluation***: conjunto de 20383 audios pensados como conjunto de test.
- ***Balanced train***: conjunto de 22160 audios pensados para entrenar de manera balanceada con respecto a la jerarquía que ejercen las etiquetas "hijas" de las 7 principales.
- ***Unbalanced train***: último conjunto de datos, constituido por 2042985 ejemplos. Como indica el nombre los datos no están balanceados lo cual presta a que este conjunto sea preprocesado a la hora de crear una base de entrenamiento más apta.

En este proyecto se han utilizado, para entrenar, tanto el conjunto *balanced train* como un subconjunto del grupo *unbalanced train* de la base de datos, generado especificamente para este proyecto y que se ha llamado Sub–*Unbalanced train*. Del primer grupo se han empleado la totalidad de audios disponibles, separando un total de 2000 audios aleatorios para ser utilizados en validación. Es importante señalar que pese a aparecer en su nombe que estén balanceados solo lo están para las 527 clases de la jerarquía más baja de la base de datos, para el caso de este proyecto, en la tabla 3.1 se ve más claramente su distribución.

Por otro lado, del segundo conjunto se han cogido de manera aleatoria 192417 audios. Se ha tratado de conseguir que el subconjunto final sea lo más balanceado posible, pero sin desperdiciar ninguna muestra de las clases menos respresentadas. Por ello no se ha coseguido balancear la base

	Balanced train	*Evaluation*	*Unbalanced train*	Sub–*Unbalanced train*
Human Sounds	35,91 %	36,63 %	53,47 %	46,64 %
Animal Sounds	10,89 %	11,69 %	2 %	18,24 %
Sounds of Things	11,22 %	11,79 %	4,7 %	18,87 %
Ambiguous Sounds	35,89 %	38,07 %	13,38 %	25,11 %
Background	39,30 %	38,23 %	51,81 %	31,88 %
Natural sounds	4,41 %	4,85 %	1,72 %	18,23 %
Music	10,41 %	10,08 %	9,54 %	21,63 %

Tabla 3.1: Distribución de etiquetas usadas en las bases de datos

de datos correctamente, pero como se puede ver en la tabla 3.1 la distribución se mejora mucho con respecto a la original. Una solución habría sido introducir tan solo audios que tuvieran una etiqueta única; sin embargo, esto no se tuvo en consideración ya que en un ambiente realista siempre se corre el riesgo de tener varias clases simultáneas.

Al igual que en el caso del conjunto *balanced* se han separado aleatoriamente audios, del subconjunto creado, para la validación; en este caso se cogieron 21078 audios.

Por último, para test, se ha utilizado el conjunto de *evaluation*.

4

Diseño y desarrollo

A lo largo de este capítulo se expondrá de manera extensa el trabajo realizado. Primero se comentará la organización general del sistema generado; después se hará especial hincapié en las diferentes redes neuronales implementadas, explicando en detalle los diferentes modelos probados y así como sus características.

4.1. Sistema

Como se ha ido comentando a lo largo de la memoria, el fin de este proyecto es construir un sistema de clasificiación de clases de audios basado en redes neuronales. Para hacer más completo el trabajo, se decidió hacer uso de las herramientas publicadas en [2], esto permitió añadir una capa extra de complejidad al proyecto haciéndolo más completo. A raíz de esto se ha conseguido un programa que actúa conforme a dos redes neuronales diferentes, aprovechando las virtudes de ambas, para lograr el objetivo con audios que no estén en la base de datos.

En la figura 4.1 se plantea de forma esquemática la estructura principal que sigue el proyecto. Se parte de la toma o grabación de un audio cualquiera, este será el archivo de que partirá todo

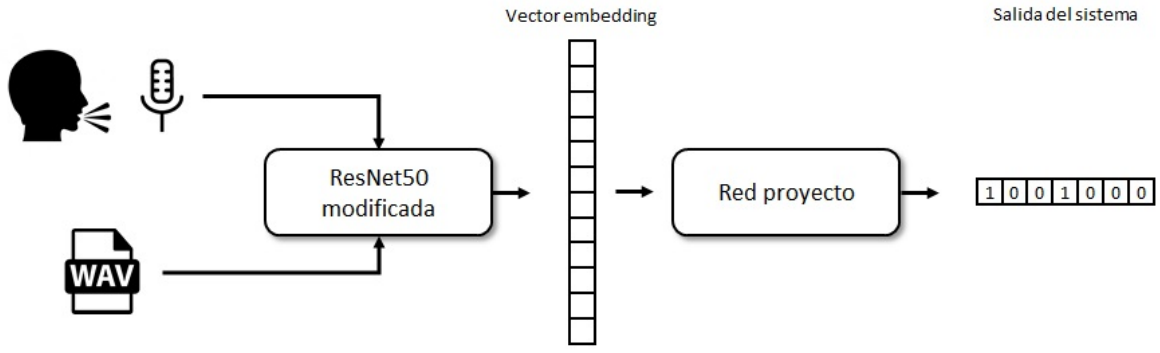


Figura 4.1: Esquema del sistema implementado en el proyecto. La primera fase corresponde a la red adaptada de [2] y la segunda corresponde a al clasificador implementado en el proyecto.

el proceso. El audio deberá estar grabado a una frecuencia de muestreo de 16KHz, para que cumpla los requisitos que tiene la base de datos AUDIOSET [1], de no ser así se modifica su frecuencia de muestreo mediante el uso de la herramienta Sox [17].

A lo anterior le sigue la aplicación de la red implementada en [2]; tomará el audio de entrada y lo devolverá en forma de matriz de *embeddings*. Como se comentó anteriormente, en el proyecto se trabaja con matrices de tamaño 10x128; sin embargo, esta red devuelve un vector de 1x128 por cada segundo de audio que aportemos. Para solucionar eso se recogen todos los vectores de salida de la red modificada y se aúnan en una gran matriz de Sx128, siendo S el número de segundos de los que conste el audio que se trata. Posteriormente el sistema está implementado de tal manera que vaya recorriendo la matriz de vectores *embeddings* en grupos de 10, para satisfacer el tamaño de entrada, simulando un enventanado y desplazándose en saltos de 1 para aproximar el resultado a una clasificación por segundo.

Ahora lo que queda es utilizar la red propia de este proyecto. Esta coge la matriz de 10x128 que se haya elegido del proceso anterior y devolverá un vector de 7 dimensiones, una por cada clase, con la salida estimada. La salida comprenderá valores entre 0 y 1 y serán independientes entre ellos, es decir, no tiene que sumar 1 entre todos ya que puede darse el caso que un mismo segmento de audio contenga más de una clase. Este proceso y arquitectura se explica en el apartado 4.2 con más detalle. En la figura 4.2 se puede ver un ejemplo de salida del sistema, donde se exponen los resultados de un segmento de audio cualquiera.

4.2. Redes Neuronales

Durante la realización de este proyecto se han entrenado una gran cantidad de redes neuronales con el fin de lograr los mejores resultados posibles. Para poder conseguirlo se ha optado por un entrenamiento iterativo basado en el cambio de los parámetros de la propia estructura de la red,dicho de otra manera, de la modificación de hiperparámetros. A este tipo de entre-

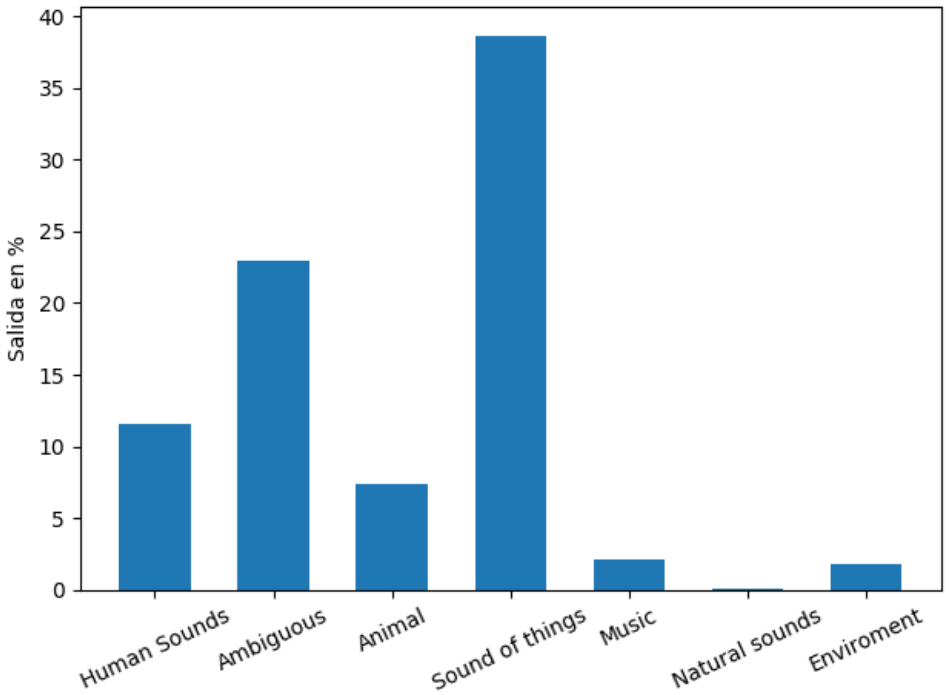


Figura 4.2: Ejemplo de salida del sistema desarrollado en el proyecto.

namiento se le denomina *grid search* y permite entrenar de seguido muchos modelos si perder tiempo modificándolos a mano.

El lenguaje de programación elegido ha sido Python ya que es en el cual está desarrollada las dos librerías principales del trabajo: TensorFlow, Keras [18] y Numpy. La primera es una librería de *deep learning* creada por Google, está específicamente diseñada para desarrollar códigos de redes neuronales y es de las más utilizadas hoy en día. Sin embargo, es compleja de usar por si sola y de ahí surge la necesidad de la segunda librería. Keras es una herramienta de apoyo que corre por encima de TensorFlow y lo gestiona de una manera sencilla, haciendo muy fácil y rápida la implementación de redes complejas sin que el código que las representan lo sea. La última es una librería que implementa funciones matemáticas de alto nivel para operar con vectores o matrices.

4.2.1. Modelos DNN

Para este tipo de modelos se ha decidido utilizar una arquitectura diferenciada en dos partes fundamentales; una parte de ella es invariante y la otra es la parametrizada. La parte invariante está constituida por la dimensionalidad de la entrada, la última capa oculta y la salida; así como hiperparámetros fundamentales para el entrenamiento. Estos casos invariantes se resumen de la siguiente manera:

- **Capa de entrada**: consta de una dimensionalidad constante de 1280, este valor viene del estiramiento de la matriz de embeddings de 10x128.
- **Última capa oculta**: consta de una dimensionalidad constante de 128 neuronas.
- **Capa de salida**: consta de una dimensionalidad constante de 7, este número viene dado por la cantidad de clases que queremos tratar (las 7 principales de la ontología de AUDIOSET).
- **Función de activación**: siempre será ReLU [15] excepto en la capa de salida que será *sigmoid*.
- **Optimizador**: siempre será Adam.
- **Función de coste**: siempre será *binary crossentropy*.

Por otro lado los parámetros de la red que si que se ven modificados son los siguientes:

- **Cantidad de capas ocultas intermedias**: se modifica su número de 2 hasta 6, lo que al combinarlo con la parte invariante se resume en redes de una profundidad mínima de 4 y máxima de 8.
- **Número de neuronas en las capas intermedias**: se modifica su cantidad eligiendo entre los valores [1024 512 256 128].
- ***Learning rate***: puede oscilar entre los siguientes valores [0.1 0.01 0.001 0.0001 0.00001].
- **Dropout** [16]: al igual que los casos anteriores se escoge entre los siguientes valores [0 0.2 0.3 0.4].

Tras todas estas pruebas se escogió la que más convenía a nuestras necesidades y se eligió para servir de *baseline* del resto del proyecto. Dado que es el estilo de red más sencillo se espera que modelos más complejos sean capaces de adaptarse mejor al problema.

4.2.2. Modelos LSTM

Con este estilo de red neuronal se ha decidido seguir un planteamiento similar al de los casos del apartado anterior. La estructura de la red estará diferenciada en 2 partes fundamentales: la parte invariante y la parametrizada. Cabe destacar que este tipo de redes, al acabar en una capa de clasificación, deben utilizar en las capas finales el modelo DNN. También conviene aclarar que se ha utilizado en todas las capas LSTM, excepto la justamente anterior a la capa DNN final, la secuencia temporal completa; habilitando la opción de keras *return_sequences*. Con todo esto aclarado, pasamos a ver los parámetros consantes elegidos para estos modelos de redes:

- **Capa de entrada**: consta de una dimensionalidad constante de 10x128, esto viene dado por el tamaño definido para las matrices de datos: 10 vectores embeddings con 128 valores cada uno, por audio.
- **Última capa oculta**: consta de una dimensionalidad constante de 128 neuronas, al igual que los casos DNN anteriores.
- **Capa de salida**: al igual que en el apartado anterior, consta de una dimensionalidad constante de 7 por la cantidad de clases que queremos tratar.
- **Función de activación**: en las capas LSTM siempre serán las utilizadas por defecto en estos casos que ya se explicó en el apartado 2.1.2: función sigmoid para las puertas y función tangente hiperbólica para la entrada y salida. A su vez y al igual que en el modelo anterior, las capas DNN intermedias tendrán ReLU y la capa de salida tendrá la función *sigmoid*.
- **Optimizador**: siempre será Adam.
- **Función de coste**: siempre será *binary crossentropy*.

La parte de los modelos parametrizada es la siguiente:

- **Cantidad de capas ocultas intermedias**: se modifica solo el número de capas LSTM, variando su número de 2 hasta 3, lo que al combinarlo con la parte invariante se resume en redes de una profundidad mínima de 4 y máxima de 5.
- **Número de neuronas en las capas intermedias**: se modifica su cantidad eligiendo entre los valores [512 256 128].
- ***Learning rate***: se escoge el mismo rango de valores que para el caso DNN [0.1 0.01 0.001 0.0001 0.00001].
- **Dropout**: al igual que el parámetro anterior, se eligen los mismos datos que en el caso de los modelos DNN [0 0.2 0.3 0.4].

4.2.3. Modelos CNN

Al igual que en los modelos anteriores, para este estilo de redes se ha decidido separar la arquitectura en una parte invariante y otra parametrizada. De nuevo, nos encontramos ante un modelo que para lograr clasificar entre 7 clases necesita terminar en al menos una capa DNN. A continuación se enumeran los elementos que no modifican su estructura a lo largo del entrenamiento:

- **Capa de entrada**: consta de una dimensionalidad constante de 10x128x1, esta última dirección es un requerimiento de la librería Keras a la hora de implementar capas CNN de entrada.
- **Última capa oculta**: de nuevo siempre será de 128 neuronas.
- **Capa de salida**: consta de una dimensionalidad de 7 al igual que en los modelos anteriores.
- **Función de activación**: siempre será ReLU en las capas intermedias y *sigmoid* en la capa de salida, como en el caso de los modelos DNN.
- **Optimizador**: siempre será Adam.
- **Función de coste**: siempre será *binary crossentropy*.

La parte de los modelos parametrizada es la siguiente:

- **Cantidad de capas ocultas intermedias**: se modifica solo el número de capas CNN, varía entre las profundidades 1 hasta 3, lo que al combinarlo con la parte invariante se resume en redes de una profundidad mínima de 3 y máxima de 5.
- **Número de neuronas en las capas intermedias**: se modifica su cantidad eligiendo entre los valores [32 16 8 4].
- ***Learning rate***: de nuevo se elige el mismo rango de valores [0.1 0.01 0.001 0.0001 0.00001].
- **Dropout**: toma los valores [0 0.2 0.3 0.4].
- **Filtros**: se modifican los tamaños entre 2x10 y 2x20. Cabe destacar que siempre se usa el mismo tamaño en todas las capas CNN que se establenzcan. Este tipo de filtros no son de un tamaño habitual y se han utilizado por el interés de estudiar si existen relaciones locales entre las dimensiones de un vector *embedding*. También se han realizado pruebas con filtros de 2x2 con el fin de ver el comportamiento al usar elementos más habituales en redes CNN.

5
Pruebas y resultados

A lo largo de este capítulo se van a exponer los mejores resultados obtenidos a lo largo del proyecto. Como se ha introducido en el capítulo 4.2 se han entrenado múltiples redes neuronales para buscar el mejor candidato a resolver el problema del proyecto; sin embargo, en este capítulo solo se destacaran los mejores resultados para cada tipo de modelo.

Para poder entrenar diferentes modelos de una manera cómoda, se utilizó una tarjeta gráfica *Nvidia Geaforce* GTX 1070 y un entrenamiento iterativo que fuera rotando entre las diferentes configuraciones, explicadas en el apartado 4.2, según fueran acabando los modelos previos. Por defecto siempre se ha eligido un tamaño de *batch* de 100 y 100 *epochs* como máximo número de iteraciones de entrenamiento. Sin embargo, entrenar hasta el final todos los modelos probaba ser algo tedioso e innecesario, pues en múltiples ocasiones el mejor resultado en validación durante el entrenamiento no se daba al término de las 100 *epochs* si no en una momento intermedio, por ello se decidió utilizar una función, que implementa Keras por defecto, llamada *EarlyStopping* [19].

El *EarlyStopping* es un procedimiento que consiste en finalizar un entramiendo, antes de que se llegue al número de *epochs* preestablecido, si se cumplen unas condiciones especificadas por el usuario. En el caso de este trabajo el entrenamiento concluía si a lo largo de un periodo de como máximo 10 *epochs*, la función de coste apliacada a la base de datos de validación no percibía una mejora. Con esto se consiguieron entrenar más de 2000 modelos diferentes sin perder mucho tiempo en entrenamiento innecesario.

Una vez se tienen todos los modelos entrenados, se prueban con la base de datos de validación para elegir el modelo más conveniente, eligiendo aquel que tenga el mAP más alto. Una vez se tiene el modelo definitivo, se utiliza sobre la base de Test, *evaluation*, para extraer los resultados finales.

5.1. Resultados con la base de datos *balanced*

Como indica el título del capítulo, en este bloque se expondrán los resultados para las diferentes redes neuronales entrenadas con la base de datos *balanced*. En la tabla 5.1, los resultados se muestran destacando la configuración final de hiperparámetros con la que se han obtenido el mejor rendimiento.

De manera aclaratoria, el código de hiperparámetros es:

- **C**: se referirá al número de capas del modelo al que hace refenrecia (DNN, LSTM o CNN), sin tener en cuenta las capas invariantes explicadas en el apartado 4.2.
- **N**: referente al número de unidades fundamentales de cada capa C.
- **LR**: hará referencia al *learning rate* con el que se ha entrenado el modelo.
- **DR**: expondrá el *dropout* que ha habido en las capas intermedias durante ese entrenamiento.

Los mejores resultados se obtienen con la red LSTM. El modelo está formado, en su versión final, de 2 capas LSTM con 512 unidades en cada una de ellas; esto da un total de 3478535 parámetros entrenados. El resultado final en validación es de 0.5922 y en test de 0.5984 de mAP, el cual no se diferencia mucho de los obtenidos para los demás modelos probados.

Cabe destacar que si que es significativa la mejora que hay con respecto a los resultados obtenidos en el estado del arte: 0.314 para Google y 0.327 para [10]. Sin embargo, esta no es una comparación del todo justa ya que, para el caso del estado del arte, la red trata de clasificar entre las 527 clases de la jerarquía más baja de AUDIOSET, y no las 7 que se han elegido para este proyecto.

Por otro lado, este hecho puede indicar que la aproximación seguida en este trabajo es más inteligente ya que, al clasificar los audios en la jerarquía más alta posible, se podría introducir un nuevo clasificador que siga a este y que se aproveche de los resultados. Con las clases más altas ya detectadas se deberían entrenar nuevas redes que se centren exclusivamente en las clases hijas de las 7 principales.

Además de lo anterior, en la figura 5.1 se puede ver una representación de los modelos en función de su complejidad, es decir, del número de elementos entrenables de los que consta la red. Esto abre una nueva vertiente de aproximación dado que al ser tan parecidos los resultados, en un escenario real puede que prime el tamaño de los datos de la red entrenada por encima del valor mAP final obtenido en test.

A modo de ejemplo se ha sugerido en el proyecto suponer un umbral, marcado en negro en la figura 5.1, que no difiera mucho del mejor resultado en validación pero que si sea un poco menor, en este caso 0.588. Si este fuera el caso, la red con mejores prestaciones sería la red CNN con la siguiente configuración: filtros de 2x2, 2 capas de tipo CNN (**C**), 16 neuronas en cada capa (**N**), 1e-03 de *learning rate* (**LR**) y 0.4 de *dropout* (**DR**). Cabe destacar que en las figuras 5.1 y 5.2 este modelo queda marcado con una "X", ene negro, junto con el mejor modelo del total de pruebas (la re LSTM), marcado con una "+", en cyan.

Modelo	C	N	LR	DR	Validación	Test
DNN	3	1024	1e-05	0.3	0.5799	0.5804
LSTM	2	512	1e-05	0.2	0.5922	0.5984
CNN_2x10	2	8	1e-04	0.4	0.5881	0.5898
CNN_2x20	2	16	1e-04	0.4	0.5894	0.5971
CNN_2x2	1	8	1e-04	0.4	0.5889	0.5838

Tabla 5.1: Resultados mAP obtenidos para los diferentes modelos estudiados, entrenados con la base de datos *balanced*.

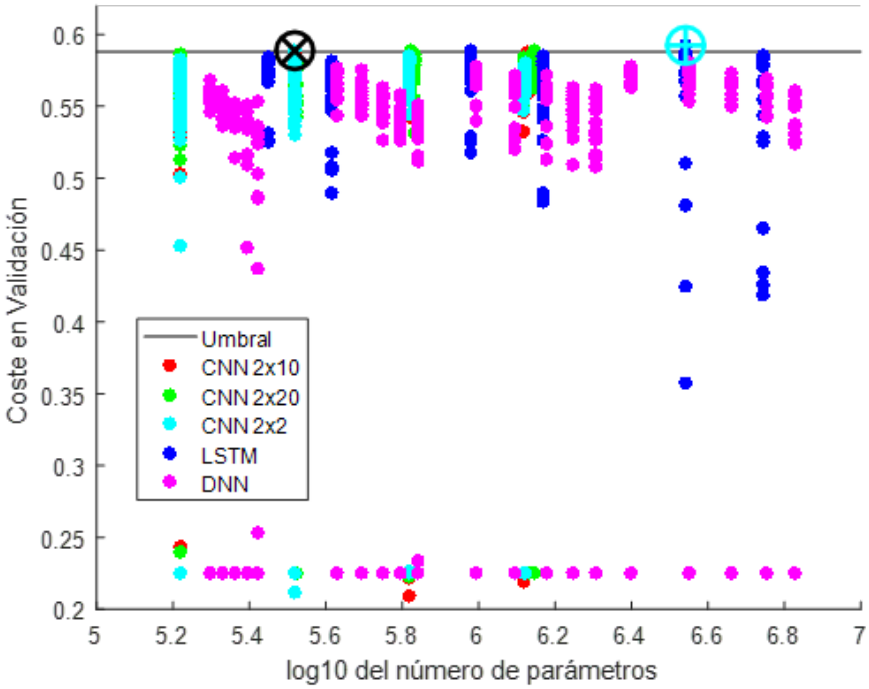


Figura 5.1: Comparativa de coste en validación en función de la cantidad de parámetros entrenables de los modelos entrenados con la base de datos *balanced*.

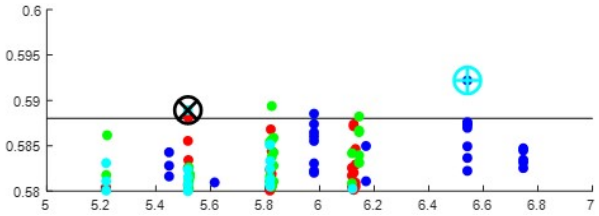


Figura 5.2: Ampliación de la sección cercana al umbral de la figura 5.1

5.2. Resultados con la base de datos Sub–*unbalanced*

En este capítulo se van a exponer los resultados obtenidos con los diferentes modelos de redes neuronales que sea han probado a entrenar con la base de datos propia de este proyecto, la Sub–*Unbalanced*. Cabe destacar que el código de símbolos de la tabla 5.2 es el mismo que el del apartado anterior.

En la tabla 5.2 se pueden ver los dos candidatos que han sido elegidos como los mejores en las pruebas. La base de datos elegida para esta tarea es muy grande, como se ha visto en el capítulo 3.3, por ello el entrenamiento necesario para cada prueba se podía extender a más de una semana. Por ello se han elegido solo estos dos estilos, porque se quería comparar un modelo DNN, de nuevo tomándolo como *baseline*, con el mejor estilo de redes del apartado anterior, es decir, la LSTM.

El mejor resultado se ha conseguido para la red LSTM de 2 capas ocultas, con 1024 neuronas cada una, un *learning rate* de 1e-04 y nada de *dropout*, lo que da una red de 21640199 parámetros entrenables; alcanzando un mAP en validación de 0.57839 y en test de 0.5884; estos resultados se quedan cerca del 0.5984 en test del mejor modelo del apartado anterior. Este nuevo entrenamiento con una base de datos mucho más compleja no ha conseguido mejorar los resultados conseguidos

Modelo	C	N	LR	DR	Validación	Test
DNN	2	1024	1e-05	0.0	0.5661	0.3977
LSTM	3	1024	1e-04	0.0	0.5739	0.5884

Tabla 5.2: Resultados mAP obtenidos para los diferentes modelos estudiados, entrenado con el subconjunto de la base de datos Sub–*unbalanced*

para la anterior. Sin embargo, sorprende mucho el desempeño obtenido por la red DNN. En validación se ha conseguido un mAP de 0.5661, si bien no logra alcanzar los resultados de la tabla 5.1 logra mantener un resultado similar, pero el resultado en test solo consigue alcanzar un 0.3977 lo que la hace el peor modelo final entrenado.

Por otro lado y al igual que en el apartado anterior, se ha añadido una gráfica en la figura 5.3 que expone la distribución de los modelos comparando sus valores de mAP en validación con el número de parámetros a entrenar. De nuevo, si se eligiera un umbral, por ejemplo 0.55, para tomar un modelo en función no solo de su rendimiento, si no también de su tamaño; habría que escoger la red LSTM de: 3 capas ocultas, 512 neuronas por capa LSTM, *learning rate* de 1-e04 y ningún *dropout*; marcada con una "X" en la figura 5.3 y 5.4 junto con el mejor modelo marcado con una "+".

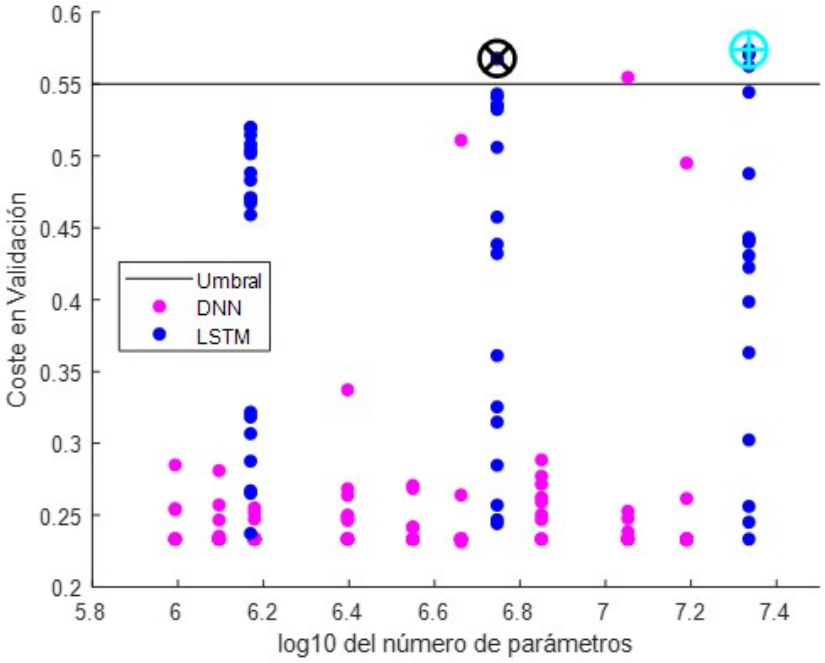


Figura 5.3: Comparativa de coste en validación en función de la cantidad de parámetros entrenables de los modelos entrenados con la base de datos Sub–*unbalanced*

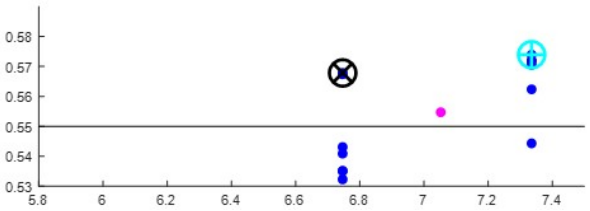


Figura 5.4: Ampliación de la sección cercana al umbral de la figura 5.3

6

Conclusiones y trabajo futuro

6.1. Conclusiones

De manera general se puede concluir el proyecto con la afirmación de que se ha conseguido implementar un sistema que detecta de manera más o menos satisfactoria eventos acústicos relacionados con el origen que produce un audio dado. Sin embargo, si se toma el trabajo como la suma de diferentes partes, se pueden destacar diferentes objetivos cumplidos:

- Se ha llevado a cabo un estudio de diferentes herramientas presentes en el lenguaje de programación Python, tales como: TensorFlow y Keras para todo lo relacionado con la implementación de redes neuronales y Numpy para toda la gestión de matrices de datos a la hora de procesar la base de datos. Con esto y con el lenguaje de programación bash se ha generado la totalidad del código del trabajo.

- Se ha llevado a cabo un preprocesado exhaustivo de la base de datos utilizada; consiguiendo modificar la misma para que los datos obtuvieran las etiquetas que se requerían para el proyecto, siempre siguiendo la ontología especificada en los propios datos. Gracias a esto, ahora se tiene la posibilidad de, con esa misma base de datos, optar a resolver problemas diferentes ya que el *groundtruth* puede ser modificado según se necesite.

- Se ha adaptado un sistema ajeno al proyecto para que funcione según la necesidades del trabajo; consiguiendo así poder calcular nuevas matrices de *embeddings*, ajenas a la base de datos, con las que realizar nuevas pruebas.

  Por otro lado, se ha conseguido adapatar también, de manera satisfactoria, otro sistema que permite llevar a cabo las medidas de rendimiento mAP correctas con las que poder comparar los resultados del trabajo con los de otros estudios publicados.

- Se ha implementado código que permite entrenar diferentes modelos y configuraciones de las redes neuronales explicadas anterioremente de una manera sencilla. Además, con él,

se puede realizar un entrenamiento iterativo que deja procesar muchas arquitecutas diferentes de redes de manera automática, salvado mucho tiempo en configuraciones al usuario.

- Tras aproximadamente un mes entrenando de manera ininterrumpida, se han coseguido finalizar satisfactoriamente más de 2000 modelos diferentes de redes neuronales. En el caso de este TFM se han dado como mejores opciones 5 redes con arquitecturas distintas entrenadas con la base de datos *balanced* y 2 con la base Sub–*unbalanced*.

  Con lo anterior se ha logrado un sistema final que, si bien no es comparable con los resultados generales de otros estudios, si se pueden comparar los AP de alguna de las clases por separado; esto es así ya que coinciden con las clases de algunos de esos trabajos. Siendo este el caso, la categoría "Music" puede ser comparada con otros estudios; en el caso de este proyecto se ha conseguido un AP de 0.9313 mientras que en estudios como [1] o [21] consiguieron unos AP de 0.896 y 0.898 respectivamente.

6.2. Trabajo futuro

Una posible vía futura de investigación para continuar este proyecto sería la de probar nuevas arquitecturas de redes, con diferentes configuraciones, llegando al punto de mezclar arquitecturas básicas como las CNN y las LSTM.

Otra posible vía de desarrollo sería aquella que implementara de manera satisfactoria los modelos de antención explicados en el aparatado 2.4. Para ello se deberán adecuar las etiquetas de la base de datos para que tomen todas las combinaciones posibles entre ellas y poder aplicar la función de activación *softmax* de salida que este tipo de herramienta necesita.

Por otro lado y para finalizar, se podría extender el objetivo del proyecto y tratar de ampliarlo para lograr reconocer eventos en una jerarquía más baja de la base de datos, como dijo previamente en el capítulo 5. Por ello una posible vía de desarrollo sería entrenar nuevas redes neuronales, con entrenamientos exclusivos de la clase predicha por los clasificadores de este proyecto.

Glosario

- **DNN**: Deep Neural Network
- **CNN**: Convolutional Neural Network
- **LSTM**: Long Short-Time Network Recurrent Neural Netowrk
- **RNN**: Recurrent Neural Network
- **mAP**: mean Average Precision
- **ReLU**: Rectified Linear Unit

Bibliografía

[1] Gemmeke, J.F., Ellis, D.P.W., Freedman, D., Jansen, A., Lawrence, W., Channing Moore, R., Plakal, M. y Ritter, M., *AUDIO SET: an ontology and human–labeled dataset for audio events*, IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 776–780, 2017. Disponible en https://research.google.com/audioset/index.html.

[2] Hershey, S., Chaudhuri, S., Ellis, D., P., W., Gemmeke, J., F., Jansen, A., Channing Moore, R., Plakal, M., Platt, D., Saurous, R., A., Seybold, B., Slaney, M., Weiss, R., J. y Wilson, K., *CNN architectures for large-scale audio classification*, IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 131–135, 2017.

[3] Hochreiter, S. y Schmidhuber, J., *Long Short-Term Memory*, Neural Computation, vol. 9, no 8, pp. 1735–1780, Nov. 1997.

[4] Rumelhart, D. E., Hinton, G. E. y Williams, R. J., *Parallel distributed processing*, Explorations in the microstructure of cognition, vol. 2, pp. 216–271, 1986.

[5] Elman, J.L., *Finding structure in time*, COGNITIVE SCIENCE, vol. 14, no. 2, pp. 179–211, 1990.

[6] Lozano Díez, A., *Bottleneck and embedding representation of speech for dnn-based language and speaker recognition*, 2018. Doctoral dissertation, Universidad Autónoma de Madrid.

[7] He, K., Zhang, X., Ren, S. y Sun, J., *Deep residual learning for image recognition*, arXiv preprint arXiv:1512.03385, 2015.

[8] Tang, G., Sennrich, R. y Nivre, J., *An Analysis of Attention Mechanisms: The Case of Word Sense Disambiguation in Neural Machine Translation*, arXiv:1810.07595v1, 17 Oct 2018.

[9] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Homez, A.N y Kaiser, L, *Attention is all you need*, Proc. NIPS, 2017.

[10] Kong, Q., Xu, Y., Wang, W., y Plumbley, M.D., *AudioSet Classification with Attention Model: A Probabilistic Perspective*, arXiv:1711.00927v1, 2 Nov 2017.

[11] Jeong, I. y Lim, H., *Audio tagging system for DCASE 2018: focusing on label noise, data augmentation and its efficient learning*, DCASE, 2018

[12] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irvingm G., Isard, M., Jozefowicz, R., Jia, Y., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Schuster, M., Monga, R., Moore, S., Murray, D., Olah, C., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke,

V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., y Zheng, X., *TensorFlow: Large-scale machine learning on heterogeneous systems*, 2015. Software disponible en tensorflow.org. Last accessed on 17/Jun/2019.

[13] Darna, J.S. y Toledano, D.T., *Audio event detection on Google's Audio Set database: Preliminary results using different types of DNNs*, in Proc. of IBERSpeech, 2018.

[14] Oliphant, T., *NumPy: a guide to Numpy*, USA: Trelgol Publishing, 2006. Software disponible en http://www.numpy.org/. Last accessed on 19/Jun/2019.

[15] Nair, V. y Hinton, G.E., *Rectified linear units improve restricted boltzmann machines*, Proceedings of the 27th International Conference on Machine Learning, IEEE, pp. 807–814, 2010.

[16] Srivastava, N., Hinton, G.E., Krizhevsky, A., Sutskever, I. y Salakhutdinov, R., *Dropout: a simple way to prevent neural networks from overfitting*, Journal of Machine Learning Research. IEEE, pp. 1929–1958, 2014.

[17] Software disponible en *http://sox.sourceforge.net/*

[18] Chollet, F. y otros, *Keras*, 2015. Software disponible en https://keras.io. Last accessed on 19/Jun/2019.

[19] Prechelt, L., *Early Stopping - But When?*, Neural Networks: Tricks of the Trade, pp. 55–67, 2012.

[20] Zazo Candil, R., *Exploiting temporal context in speech technologies using lstm recurrent neural networks*, 2018. Doctoral dissertation, Universidad Autónoma de Madrid.

[21] Benito-Gorrón, D., Lozano-Díez, A., Toledano, D.T. y , Gonzalez-Rodríguez, J., *Exploring Convolutional, Recurrent and Hybrid Deep Neural Networks for Speech and Music Detection in a Large Audio Dataset*, EURASIP Journal on Audio, Music and Speech Processing, 18 Jun 2019. https://doi.org/10.1186/s13636-019-0152-1. Last accessed on 19/Jun/2019.