Universidad de Alcalá
Escuela Politécnica Superior

Grado en Ingeniería en Sistemas de Telecomunicación

Trabajo Fin de Grado

Diseño, implementación y evaluación de una estrategia de detección de eventos acústicos

Autor: David José Gómez Ortega

Tutor: Javier Macías Guarasa

Cotutora: Leticia Monasterio Expósito

2021

UNIVERSIDAD DE ALCALÁ

ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería en Sistemas de Telecomunicación

Trabajo Fin de Grado

Diseño, implementación y evaluación de una estrategia de detección de eventos acústicos

Autor: David José Gómez Ortega

Tutor: Javier Macías Guarasa

Cotutora: Leticia Monasterio Expósito

Tribunal:

Presidente: José Manuel Villadangos Carrizo

Vocal 1º: Mª del Carmen Pérez Rubio

Vocal 2º: Javier Macías Guarasa

Fecha de depósito: 7 de octubre de 2021

A mi familia y a mis amigos, en especial a mis padres, mi hermana y mi hermano, y por supuesto, a Javier y Leticia...

"Incluso la gente que afirma que no podemos hacer nada para cambiar nuestro destino, mira antes de cruzar la calle."
Stephen Hawking

Agradecimientos

"Estar preparado es importante, saber esperar lo es aún más, pero aprovechar el momento adecuado es la clave de la vida."

Arthur Schnitzler

A mis tutores, Javier y Leticia, les agradezco por darme la oportunidad de desarrollar con ellos este proyecto, por ayudarme con cualquier problema, prestarme sus conocimientos, sus correcciones y todos esos correos que han hecho que este trabajo fuera posible.

A toda mi familia, tanto a los que están, como a los que se han ido. Y en especial a mis padres, mi hermano y mi hermana, porque nunca han dudado de mí, siempre me han apoyado y son los mejores. Hay cosas que no se pueden describir con palabras y el agradecimiento que tengo hacia ellos es una de ellas.

Resumen

Este trabajo tiene como objetivo el estudio y desarrollo de un sistema de detección de eventos acústicos. Para ello, se parte de un sistema previo sobre el que se realiza una evaluación en diferentes condiciones y sobre varias bases de datos, incluyendo modificaciones en la arquitectura y parámetros de control del mismo. Este sistema utiliza un método conocido como redes neuronales, las cuales, tras una etapa de entrenamiento, generan modelos capaces de realizar predicciones sobre unos datos de entrada. Estos datos de entrada se corresponden con archivos de audio y su etiquetado extraídos de bases de datos disponibles en la red.

Durante un periodo de búsqueda de información, se recopilaron diferentes bases de datos y sistemas disponibles públicamente orientados a la detección de eventos sonoros. De esta recopilación de sistemas se selecciona uno de ellos como sistema de referencia empleado en este trabajo. El sistema de referencia es entrenado y evaluado utilizando la base de datos que utiliza por defecto, para comprobar su correcto funcionamiento. Posteriormente este sistema fue evaluado utilizando un conjunto de datos que contiene algunas de las clases de eventos sonoros que este sistema es capaz de detectar extraídos de algunas de las bases de datos recopiladas anteriormente.

De la misma forma, el sistema de referencia se entrenó y evaluó utilizando otra base de datos que contiene clases de eventos sonoros distintas a las utilizada por el sistema previamente, apodando a esta variante como "sistema urban". Para concluir se realizan varios cambios en algunos parámetros de la red neuronal empleada en este sistema, entrenando y evaluando nuevamente el sistema de referencia y el sistema urban para cada modificación, describiendo los resultados obtenidos para cada caso.

Palabras clave: Detección y clasificación de eventos sonoros, redes neuronales, base de datos.

Abstract

The objective of this work is the study and development of an acoustic event detection system. For this purpose, we start from a previous system which is evaluated under different conditions and datasets, including modifications in its architecture and control parameters. This system uses a method known as neural networks, which, after a training stage, generate models capable of making predictions on input data. This input data corresponds to audio files and their labeling extracted from databases available on the network.

In the same way, the reference system was trained and evaluated using another database that contains classes of sound events different from those used by the system previously. This new system was dubbed as “sistema urban”. To conclude, several changes will be made in some parameters of the neural network used in this system, training and evaluating again the reference system and “sistema urban” for each modification, commenting on the results obtained for each case.

Keywords: Detection and classification of sound events, neural networks, databases.

Índice general

Índice de figuras

Índice de tablas

Lista de acrónimos

ADAGRAD	Algoritmo de Gradiente Adaptativo o *Adaptive Gradient Algorithm*.
ADAM	Estimación de Momento Adaptativo o *ADAptive Moment estimation*.
AI	Inteligencia Artificial o *Artificial Intelligence*.
ANNs	Redes Neuronales Artificiales o *Artificial Neural Networks*.
DCASE	Detección y Clasificación de Escenas y Eventos Acústicos o *Detection and Classification of Acoustic Scenes and Events*.
DCT	Transformada Discreta del Coseno o *Discrete cosine transform*.
DFT	Transformada Discreta de Fourier o *Discrete Fourier Transform*.
DOA	Dirección de Llegada o *Direction Of Arrival*.
ELU	Unidad Lineal Exponencial o *Exponential Linear Unit*.
FC	completamene conectada o *Fully Connected*.
GCC	Correlación Cruzada Generalizada o *Generalized Cross Correlation*.
GD	Descenso por Gradiente o *Gradient Descent*.
GIST	Grado en Ingeniería en Sistemas de Telecomunicación.
LReLU	Unidad Lineal Rectificada con Fugas o *Leaky Rectified Linear Unit*.
LSTM	Red de Memoria de Largo-Corto Plazo o *Long-Short Term Memory Network*.
LSTMs	Redes de Memoria de Largo-Corto Plazo o *Long-Short Term Memory Networks*.
MFCCs	Mel Frequency Cepstrum Coefficients.
MSE	Error Cuadrático Medio o *Mean Square Error*.
PHAT	Transformada de Fase o *Phase Transform*.
PReLU	Unidad Lineal Rectificada Paramétrica o *Parametric Rectified Linear Unit*.
ReLU	Unidad Lineal Rectificada o *Rectified Linear Unit*.
SED	Detección de Eventos Sonoros o *Sound Event Detection*.
SGD	Descenso por Gradiente Estocástico o *Stochastic Gradient Descent*.

ANN	Red Neuronal Articial o *Artificial Neural Network*.
CNN	Red Neuronal Convolucional o *Convolutional Neural Network*.
CRNN	Red Neuronal Convolucional Recurrente o *Convolutional Recurrent Neural Network*.
GLU	Unidad Lineal con Puerta o *Gated Linear Unit*.
GMM	Modelo de Mezclas de Gausianas o *Gaussian Mixture Model*.
GRU	Unidad Recurrente con Puerta o *Gated Recurrent Unit*.
RNN	Red Neuronal Recurrente o *Recurrent Neural Network*.
STFT	Transformada de Fourier de Corto Plazo o *Short-time Fourier transform*.
SVM	Máquina de Soporte Vectorial o *Support Vector Machine*.
Tanh	Tangente Hiperbólica.
TFG	Trabajo de Fin de Grado.

Capítulo 1

Introducción

"Esperamos que pueda suceder cualquier cosa, y nunca estamos prevenidos para nada."

Sophie Soynonov

1.1 Presentación

Este documento describe el trabajo realizado en el Trabajo de Fin de Grado (TFG) para el título correspondiente al Grado en Ingeniería en Sistemas de Telecomunicación (GIST). El trabajo tiene como objetivo el estudio y desarrollo de una estrategia de detección de eventos acústicos que permita, a partir de datos de audio de entrada correspondientes a unas clases de eventos sonoros específicos, determinar de manera automática a qué clase de eventos sonoro pertenece, así como para que instantes de tiempo se encuentra activo. Para este trabajo, estos datos de audio pueden ser extraídos de una serie de bases de datos disponibles en la red.

Para ello se recurrió a realizar una búsqueda inicial de información sobre diferentes sistemas de detección y clasificación de eventos sonoros, cuyos autores hubieran publicado su código fuente de manera pública y fuera posible su uso. A partir de dicho se planteó la selección de un sistema como sistema de referencia sobre el que abordó una evaluación del rendimiento, aplicando diferentes modificaciones. La gran mayoría de los sistemas eficientes orientados a resolver esta tarea hacen uso de redes neuronales artificiales o *Artificial Neural Networks* (ANNs), un tipo de sistema informático basado en el funcionamiento de las redes neuronales biológicas existentes en los cerebros de los animales, capaces de generar modelos que puedan realizar estas tareas de detección y clasificación, entre otras muchas, a partir de un conjunto de datos de entrenamiento.

Se planteó igualmente entrenar el sistema de referencia con otra base de datos distinta a la que utiliza de forma predeterminada, y realizar finalmente diferentes modificaciones en la arquitectura de la red neuronal empleada en este sistema, al igual que evaluar el sistema de referencia utilizando un nuevo conjunto de evaluación formado con información de otras bases de datos (conjunto que se construyó a partir de un estudio exhaustivo de las bases de datos disponibles en la comunidad científica). Posteriormente se evaluó el rendimiento para cada arquitectura de la red neuronal modificada, tanto para el sistema entrenado con su base de datos predeterminada, como para el que utiliza la nueva base de datos, comparando sus resultados con los obtenidos al evaluar el sistema sin realizar estas modificaciones.

1.2 Motivación

En el análisis de entornos humanos la detección de eventos es un objetivo fundamental en distintas áreas de la automatización de servicios (publicidad y venta, seguridad, salud, servicios sociales para sectores vulnerables de la población, etc.), a partir de datos multimodales disponibles gracias al gran despliegue de sensores intercomunicados, de bajo coste y alta velocidad y de sistemas de procesamiento cada vez más potentes.

Este trabajo se enmarca de forma general en el de análisis de escenas [8], y más específicamente en el análisis computacional de escenas y eventos acústicos, que ha recibido una importante atención en los últimos años [9–12], más enfocadas a tareas de descripción textual y abstracta de estos entornos (detección y clasificación de eventos acústicos y de las escenas que integran diferentes fuentes acústicas simultáneas) que al procesamiento de bajo nivel (separación de fuente, dereverberación, etc.) que era el foco de la literatura hace una década [13].

El trabajo se centrará en el desarrollo, implementación y evaluación de un sistema de detección de eventos utilizando únicamente información acústica, pensando para un marco final en donde se pueda perseguir la integración de la información acústica de alto nivel en sistemas de detección de eventos multimedia, como puede ser la aplicación de detección de anomalías o situaciones peligrosas, donde el audio suele usarse como fuente complementaria de información.

1.3 Objetivos y desarrollo

El objetivo global del TFG es el diseño, implementación y evaluación de un sistema que permita la detección de escenas y/o eventos partir de información acústica exclusivamente. El campo de aplicación de este trabajo es cualquier actividad en el que sea relevante la ocurrencia de dichos eventos, posiblemente para su combinación con los detectados por otras modalidades de sensado.

El trabajo implicará la utilización de técnicas de procesamiento de señal de audio, combinadas con las de aprendizaje máquina, y la aplicación de estrategias rigurosas de experimentación y validación algorítmica. El entorno de desarrollo se apoyará en una plataforma GNU/Linux, sobre el lenguajes de programación de Python.

Los objetivos específicos de este proyecto son los siguientes:

- Formación previa en las tecnologías de inteligencia artificial y aprendizaje máquina, con especial énfasis en las de aprendizaje profundo.
- Búsqueda de información relacionada, trabajos anteriores, así como adquisición de conocimientos sobre librerías necesarias de Python.
- Selección y análisis del sistema de detección de eventos acústicos, con especial atención a las soluciones basadas en aprendizaje profundo. Deberá elegirse y/o modificarse una red que, de acuerdo con la teoría, tenga una estructura lógica para llevar a cabo el objetivo deseado.
- Búsqueda, análisis y preparación de las bases de datos disponibles. Determinación del particionado de las mismas para los procesos de entrenamiento, validación y evaluación, además de indicar las clases de eventos sonoros que contiene cada una de ellas.
- Entrenamiento y validación del sistema seleccionado. Se supervisará cómo se desarrolla dicho entrenamiento, realizando los cambios pertinentes para que se realice de forma correcta hasta conseguir que la red neuronal sea capaz de llevar a cabo la detección de eventos acústicos de la forma deseada.

- Realización de pruebas experimentales. Se someterá al sistema a diversos test que comprueben el correcto funcionamiento de éste. Tras dicha validación podrán sugerirse y/o realizarse mejoras en el sistema.
- Introducción de modificaciones en la arquitectura de la red utilizada, así como en los parámetros de control de los procesos de entrenamiento y evaluación, realizando una batería final de pruebas para validación del sistema y la generación de las conclusiones oportunas.
- Documentación del trabajo realizado.

1.4 Organización de la memoria

Esta memoria se organiza en *5* grandes capítulos.

- El *capítulo 1* consiste en una introducción sobre el TFG, en donde tras una presentación general del trabajo, se plantea el objetivo del mismo, la manera de abordarlo, la metodología y plan de trabajo seguida, además de una descripción sobre la organización de la memoria empleada.
- El *capítulo 2* se trata del estudio teórico realizado, donde se pretende plasmar los conocimientos mínimos involucrados para la correcta comprensión de este trabajo. En este capítulo, se parte de una breve introducción donde se nombran algunos los métodos estadísticos de clasificación de eventos utilizados durante tradicionalmente, se introduce el concepto de "Machine learning" y se llega a hacer referencia las redes neuronales. Posteriormente, se describe el funcionamiento, fundamentos y características de las redes neuronales con cierta profundidad, detallando alguna de las arquitecturas típicas como son las redes neuronales recurrentes, las redes de memoria a corto y largo plazo, las redes de unidades recurrentes cerradas y las redes neuronales convolucionales. A continuación se describen algunos de los conceptos fundamentales de las señales de audio digitales y lo relacionado con el pre-procesado de estas, de cara a utilizarlas como datos de entrada en una red neuronal, asignando una mayor importancia al pre-procesado orientado a trabajar con redes neuronales convolucionales. Finalmente, se realizará una introducción sobre Python y se describe alguna de sus librerías importantes relacionadas con el tratamiento de datos de audio y redes neuronales, de cara a su uso durante este trabajo.
- El *capítulo 3* se encuentra relacionado a la etapa de búsqueda de información sobre base de datos y sistemas de detección y clasificación de eventos sonoros disponibles. En este capítulo se anotan los diferentes sistemas y bases de datos obtenidos durante ese periodo. Para cada base de datos se mencionará, siempre y cuando estos datos se encuentren disponibles: su distribución, las características que contienen los diferentes archivos de audio (por ejemplo, la procedencia de estos archivos, si todos los archivos de audio se encuentran estandarizados con una misma frecuencia de muestreo, etc), el número de clases de eventos sonoros que contiene (y cuáles son), la duración media de cada archivo de audio y el número de eventos disponibles para cada clase, entre otras cosas. En el caso de cada sistema anotado, se comentará su utilidad, aplicaciones, su propósito y funcionamiento, la arquitectura empleada y las bases de datos con las que ha sido probado.
- El *capítulo 4* describe el sistema de referencia escogido de entre los sistemas disponibles anotados en el capítulo 3, sus características, los archivos necesarios para su funcionamiento y las diferentes implementaciones realizadas, además del método de trabajo seguido para abordar el trabajo experimentalw, el sistema de evaluación utilizado y las métricas evaluadas. En este capítulo también se evalúa el sistema de referencia para determinar si sus resultados son similares a los mostrados

en su documentación, de cara a su validación. Del mismo modo, se evalúan las implementaciones propuestas y se muestran los resultados obtenidos para cada una de estas, con el fin de determinar si se produce una mejora en el rendimiento del sistema.

- El *capítulo 5* muestra las conclusiones obtenidas durante el desarrollo de todo el trabajo, además de incluir las líneas futuras propuestas a partir de este trabajo.

Capítulo 2

Estudio teórico

"Cada día sabemos más y entendemos menos."

Albert Einstein

2.1 Introducción

La *detección y clasificación automática de eventos sonoros* se encuentra entre uno de los campos de investigación tecnológicos más crecientes en los últimos años.

A pesar de que tradicionalmente la detección y clasificación de sonidos se asocia principalmente al reconocimiento automático del habla, lo cierto es que existe un gran número de aplicaciones para la tarea de indexación de material de audio con el fin de generar metadatos y etiquetas automáticas, como es el caso de la monitorización en la atención médica, la anotación de reuniones, el seguimiento de los niveles de ruido en un espacio determinado, la detección de anomalías sonoras como complemento a la vigilancia automática de entornos, etc.

Durante muchos años, se han utilizado varios métodos estadísticos de clasificación de eventos, como puede ser el caso del *operador de selección y contracción mínima absoluta (Lasso)*, *los modelos ocultos de Markov (HMMs)* o los modelos de mezclas de gausianas o *Gaussian Mixture Models* (GMMs), pudiendo lograr con todos ellos buenos resultados a la hora de clasificar diferentes eventos sonoros.

Sin embargo, a día de hoy y gracias a los avances tecnológicos, se han desarrollado otros métodos de mayor complejidad a través de la *inteligencia artificial o* Artificial Intelligence *(AI)* que resultan ser más eficientes para este tipo de tareas de alta dificultad.

El *aprendizaje automático* o *"Machine Learning"* es una rama de la *AI* pensada para generar modelos analíticos con la capacidad de identificar patrones utilizando una gran cantidad de datos que les permita realizar predicciones. En otras palabras, se podría decir que el Machine Learning es el campo de investigación y desarrollo que proporciona a los ordenadores la capacidad de aprender sin que se les indiquen las reglas que deben seguir para realizar una tarea, haciéndola de manera automática [14].

Existen varias estrategias de aprendizaje automático, entre las que destacan *el aprendizaje no supervisado* y *el aprendizaje supervisado*. Los métodos de *aprendizaje supervisado* son aquellos en los que el algoritmo de aprendizaje recibe un conjunto de entradas junto con una serie de etiquetas donde se indican los resultados correctos a obtener, haciendo que el algoritmo aprenda comparando sus predicciones con los resultados reales para encontrar errores y modificar en consecuencia el modelo a generar .

Mientras tanto, en los métodos de *aprendizaje no supervisado*, no incluyen estas etiquetas al realizar el proceso de entrenamiento de un modelo, y por lo tanto, el algoritmo intenta clasificar la información por sí mismo.

Para el caso específico de este trabajo, que tiene puesto el foco en la implementación de un sistema que sea capaz de detectar y clasificar diferentes eventos sonoros pertenecientes a un conjunto de clases en concreto, se ha optado por la utilización de métodos de aprendizaje supervisado. Concretamente, haciendo uso de técnicas conocidas como las redes neuronales artificiales o *Artificial Neural Networks* (ANNs).

2.2 Redes neuronales artificiales (ANNs)

Desde la creación del primer "modelo neuronal moderno" en 1943 por Warren McCulloch y Walter Pitts llamado *lógica umbral* [15], los avances en el campo de la investigación centrados en las Redes Neuronales Artificiales no han hecho más que aumentar exponencialmente hasta tal punto en el que a día de hoy, cuesta ver el techo de lo que se podrá lograr durante las próximas décadas...

Una ANN es un tipo de sistema informático basado en las redes neuronales biológicas existentes en los cerebros de los animales. Estas redes están formadas de manera general por un gran número de unidades de procesamiento interconectadas conocidas como *neuronas artificiales* o *"neuronas"* [16].

Como se describe en el modelo estándar de una neurona artificial [17] y como se puede observar en la Figura 2.1, cada neurona puede contener varias entradas y produce una única salida que se puede replicar y dirigir a varios destinos, siendo estos, otras neuronas o la salida final de la ANN. En lo que a las entradas se refiere, se pueden otorgar los valores pertenecientes a las características de una muestra de datos externos o las salidas de otras neuronas. Por otro lado, las salidas de las neuronas que transmiten la salida final de una ANN otorgan el resultado final, el cuál en nuestro caso es identificar qué tipo de sonido de una base de datos se encuentra activo para un instante determinado.

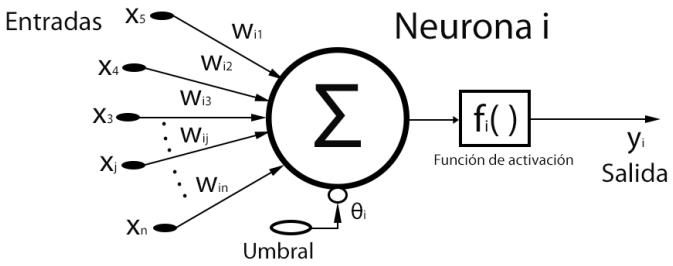


Figura 2.1: Representación del modelo de una neurona artificial estándar

La organización de las neuronas tal y como se puede apreciar en la Figura 2.2 esta basada en capas y a su vez, estas capas generalmente se pueden clasificar en tres bloques: una capa de entrada, una o varias capas ocultas y una capa de salida.

El funcionamiento de una ANN simple diseñada mediante aprendizaje supervisado es fácil de sintetizar: Los datos de entrada son transmitidos a las neuronas de la primera capa y los valores se propagan desde cada neurona hasta cada neurona de la capa siguiente. Este proceso se repite hasta llegar a las neuronas de la última capa de salida, las cuales a su vez transmiten los resultados finales de la red (como hemos comentado con anterioridad).

Cada conexión entrante de cada neurona se realiza con una ponderación determinada, de modo que a lo largo del proceso de entrenamiento de una ANN estos valores se irán ajustando con el fin de obtener

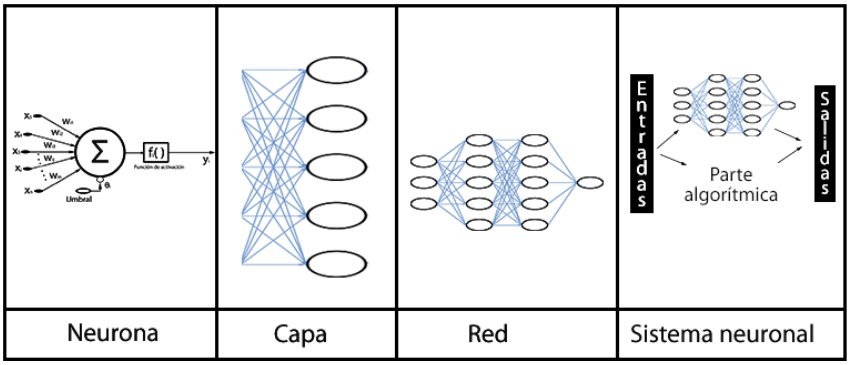


Figura 2.2: Representación del la estructura de bloques de un red neuronal

un valor óptimo que implique una mejor salida final del sistema. Del mismo modo, para cada salida de una neurona puede existir una función limitadora o umbral que altere el valor de salida o imponga un límite que no debe excederse [18]. Esta función tiene el nombre de *función de activación*, la cuál será tratada con más profundidad en la sección 2.2.1. De este modo, si nos fijamos en la Figura 2.1, la salida de una neurona se podría definir mediante la siguiente ecuación:

$$y_i = f_i\left(\sum_{j=1}^{n}(w_{ij} \cdot x_j - \theta_i)\right) \tag{2.1}$$

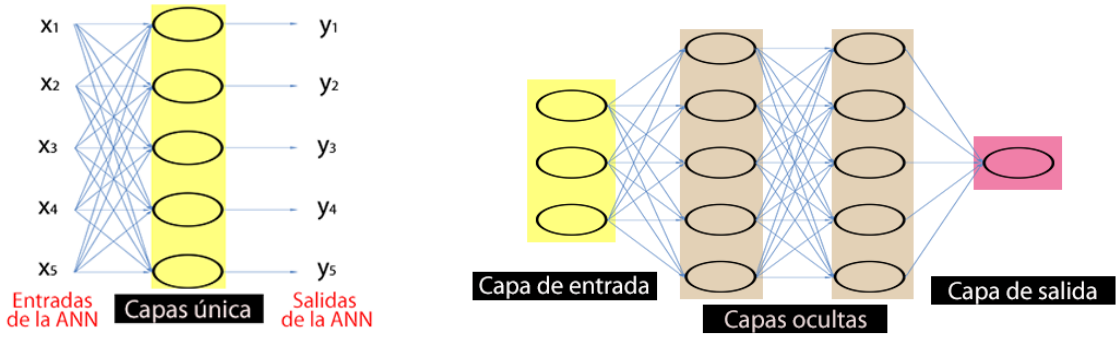
Donde:

- y_i es la salida de una neurona i.
- f_i representa la *función de activación* de la neurona i.
- w_{ij} se corresponde a los pesos asociados a cada entrada de la neurona i.
- x_j son las diferentes entradas de la neurona i.
- θ_i se define como el valor umbral de la neurona i.

La forma en la que una ANN "aprende" es a través de la generación de predicciones para cada entrada, reajustando las ponderaciones de los diferentes enlaces de las neuronas de la red cuando se detecta que la predicción realizada para una entrada es incorrecta. Este proceso se produce una y otra vez de forma recurrente hasta alcanzar los criterios preestablecidos de parada a la hora de diseñar la ANN. En un principio, todas las ponderaciones son aleatorias y por tanto, las predicciones que realiza la red son generalmente poco acertadas. Sin embargo, durante el entrenamiento, a la red se le suministra ejemplos para los que se conoce el resultado (etiquetas) para que pueda comparar las predicciones que genera con los resultados verdaderos que se deberían obtener.

La información procedente de esta comparación se propaga hacia atrás a través de la red, cambiando las ponderaciones gradualmente. A medida que progresa el entrenamiento, la red se va haciendo cada vez más precisa en la replicación de resultados conocidos. Una vez entrenada, la red se puede aplicar a casos futuros en los que se desconoce el resultado.

En cuanto a las diferentes topologías o arquitecturas que pueden presentarse en las ANNs, cabe recalcar que para que una ANN sea considerada como tal, tan solo es necesario que ésta posea como mínimo una capa formada por una o varias neuronas. Este tipo de topología son conocidas como *redes monocapa* y se suelen utilizar para tareas relacionadas con la auto-asociación, en donde se pretende



(a) Representación de una ANN monocapa

(b) Representación de una ANN multicapa

Figura 2.3: Representaciones de una ANN monocapa y una ANN multicapa

regenerar información de entrada que se presenta a la red de forma incompleta o distorsionada [19]. Por otro lado, cuando la ANN posee más de una capa, como su nombre indica, hablamos de *redes multicapa*. En la Figura 2.3 se puede observar la diferencia entre ambos tipos.

En el caso de las *redes multicapas* existen dos posibles modos de propagar la información:

- *Feedforward*, donde las neuronas reciben las señales de entrada procedentes de la capa anterior (la cual se encuentra más próxima a la entrada de la red) o de la entrada de la red (en el caso de la capa de entrada), y por tanto la información se transmite "hacia delante" en el sentido entrada → salida.
- *Feedback*, que se puede dar en ocasiones en las redes que utilizan el modo *feedforward*, en el que se conecta la salida de las neuronas pertenecientes a las capas posteriores a la entrada de las neuronas de las capas anteriores. Para este modo se considera que la propagación de la información es "hacia atrás" en sentido salida → entrada.

Con estos dos modos de propagación podemos diferenciar dos tipos de *redes multicapa*: Las *redes feedforward* o *redes unidireccionales* (que son aquellas que propagan información únicamente hacia adelante) y las *redes feedforward/feedback*, también llamadas *redes recurrentes o realimentadas* (que son aquellas que propagan información hacia delante y hacia detrás). En la sección 2.3 se mostrará algunos de los modelos de arquitectura típicos de ANNs, pero antes de entrar en detalles es necesario definir una serie de parámetros o funciones que hay que tener en cuenta a la hora de trabajar con una ANN.

2.2.1 Funciones de activación

Tal y como hemos mencionado anteriormente, cada neurona puede tener una *función de activación* encargada de limitar, alterar o establecer un umbral para los valores de salida de la neurona.

Debido a que las ANNs son capaces de resolver problemas cada vez más complejos, las *funciones de activación* se seleccionan para que los modelos generados sean no lineales, pudiendo hacer frente a problemas más complejos.

Aunque se ha comentado que la aplicación de la *función de activación* (cuando la hay) es el último paso antes de que las neuronas proporcionen una salida, esto no quiere decir que cada neurona sea independiente a la hora de aplicar una u otra función de activación. Para una ANN generalmente se suele aplicar la misma función de activación para todas las neuronas que pertenecen a una misma capa. Dependiendo de la *función de activación* que se utilice en las diferentes partes de una red el resultado final puede ser bastante variable y afectar significativamente el rendimiento de la red.

Tal y como se menciona en [20,21], la mayoría de las ANNs suelen entrenarse utilizando el algoritmo de retropropagación de error o "*backpropagation*" para poder ajustar los pesos del modelo. Para ello requieren de la derivada del error de predicción, lo que implica que las *funciones de activación* deben ser diferenciables.

Existe una gran variedad de *funciones de activación*, por lo que en este apartado vamos a describir las más comunes.

2.2.1.1 Sigmoide

La función *sigmoide* es una de las funciones más utilizadas en el diseño de ANN. La función es diferenciable y es muy útil gracias a que permite adaptar y reducir valores extremos o anómalos convirtiéndolos en muestras válidas sin tener que eliminarlos. Su funcionamiento consiste en transformar los valores de muestras independientes a valores comprendidos entre *0* a *1* [14].

Esta función se define en la ecuación 2.2.

$$f(x) = \left(\frac{1}{1+e^{-x}}\right) \tag{2.2}$$

En cuanto a su forma, tal y como se puede ver en la Figura 2.4, se encuentra centrada en *0.5* y posee un rango de *0* a *1* [21].

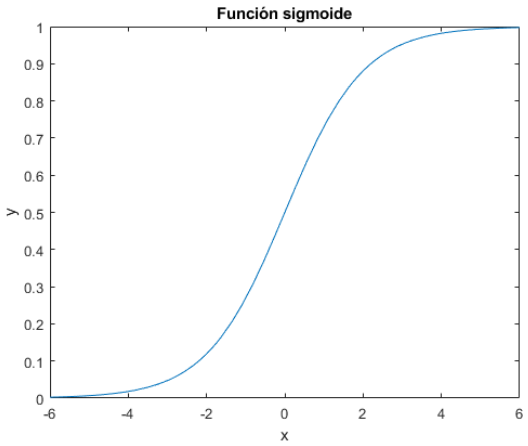


Figura 2.4: Representación de la función sigmoide

El principal inconvenientes de esta función es el *"Gradiente de fuga"*, que consiste en que la función tiende a saturarse en 0 o 1 cuando se trabaja con entradas muy pequeñas o muy grandes, tal y como se puede apreciar en la Figura 2.4 [21].

2.2.1.2 Tangente hiperbólica (Tanh)

La tangente hiperbólica (Tanh) es una función de activación muy utilizada y popular que se asemeja a la función *sigmoide* descrita anteriormente con la ventaja de que puede tratar mejor los números con valores negativos. La tangente hiperbólica (Tanh) representa la relación entre el seno hiperbólico y el coseno hiperbólico. Al igual que la función *sigmoide* también se trata de una función diferenciable [14] [21].

Esta función se define según la ecuación 2.3.

$$f(x) = \left(\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}\right) \tag{2.3}$$

En la Figura 2.5 se puede observar que su representación es muy similar a la función *sigmoide* siendo también una curva con forma de S pero esta vez, la función se encuentra centrada en 0 y posee un rango de -1 a 1 [21].

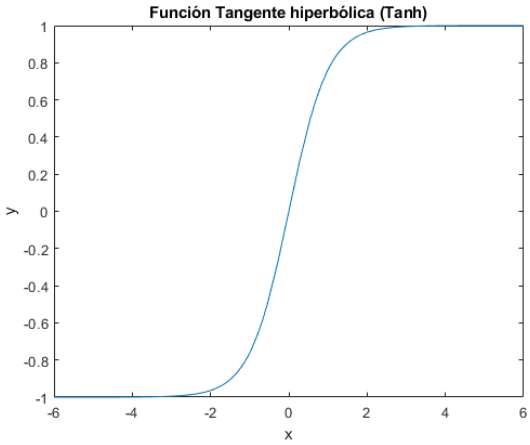


Figura 2.5: Representación de la función Tanh

Al igual que la función *sigmoide*, el principal inconvenientes es el *"Gradiente de fuga"* cuando se trabaja con entradas muy pequeñas o muy grandes, tal y como se puede apreciar en la Figura 2.5.

2.2.1.3 Unidad lineal rectificada (ReLU)

La función de activación unidad lineal rectificada o *Rectified Linear Unit* (ReLU) es probablemente la más utilizada para las capas ocultas de una ANN (principalmente en las *Redes Neuronales Convolucionales* que mencionaremos más adelante) debido a que puede gestionar mejor algunas limitaciones presentes en las funciones *Sigmoide* y Tanh para modelos profundos relacionados con la desaparición de gradientes durante el entrenamiento debido al *"Gradiente de fuga"*. Sin embargo, esta función de activación también posee otros defectos como posibles problemas de saturación o un problema conocido como "ReLU moribunda" (*dying ReLU*), el cual consiste en que algunas neuronas solo producen valores de salida de *0* debido a que la suma ponderada de sus entradas es negativa en todo momento durante el entrenamiento [21].

La función se puede definir según la ecuación 2.4.

$$f(x) = \begin{cases} 0 & si \quad x \leq 0 \\ \\ x & si \quad x > 0 \end{cases} \tag{2.4}$$

Tanto en su definición mediante la ecuación 2.4 como en su representación en la Figura 2.6, se puede observar que si el valor de entrada es positivo, se devuelve el mismo valor sin alterar; pero en el caso de que sea negativo, se devuelve un *0*.

2.2.1.4 Unidad lineal rectificada con fugas (LReLU)

La función de activación unidad lineal rectificada con fugas o *Leaky Rectified Linear Unit* (LReLU), también conocida como *Leaky ReLU* es una modificación de la función ReLU pensada para solucionar el problema conocido de la "ReLU moribunda". Para ello, si el valor de entrada de esta *función de activación* es negativo, en vez de convertirlo a la salida estrictamente en un *0*, este valor se multiplica por un hiperparámetro conocido como "α" que define el grado de "fuga" de la función.Los valores típicos

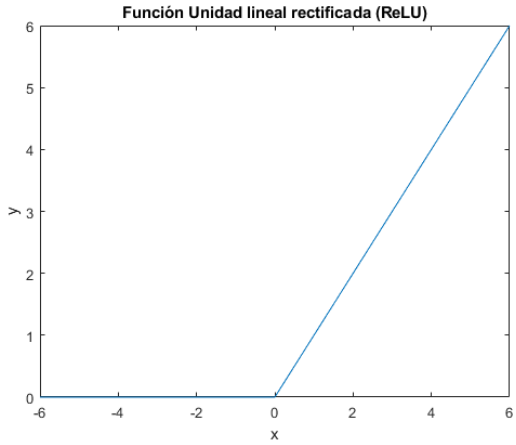
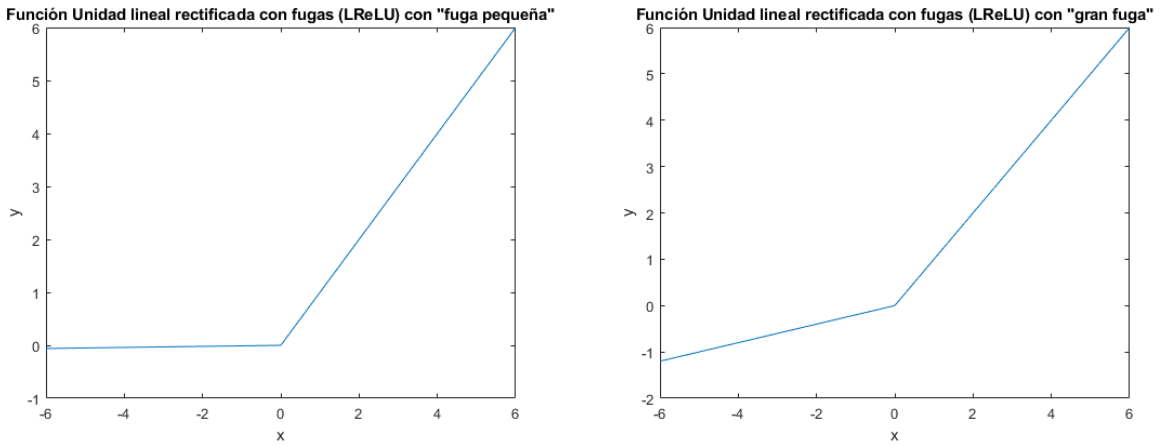


Figura 2.6: Representación de la función unidad lineal rectificada o *Rectified Linear Unit* (ReLU)



(a) Función LReLU con $\alpha = 0{,}01$

(b) Función LReLU con $\alpha = 0{,}2$

Figura 2.7: Representaciones de la función LReLU para diferentes valores de α.

para el parámetro α suelen rondar desde *0.01* (para una "fuga pequeña") hasta *0.2* (para una "gran fuga"). [22]

La función se puede definir según la ecuación 2.5.

$$f(x) = \begin{cases} x \cdot \alpha & si \quad x \leq 0 \\ \\ x & si \quad x > 0 \end{cases} \tag{2.5}$$

En la Figura 2.7 se puede apreciar dos representaciones de la función unidad lineal rectificada con fugas o *Leaky Rectified Linear Unit* (LReLU) para dos valores diferentes de α.

2.2.1.5 Unidad lineal rectificada paramétrica con fugas (PReLU)

La función de activación unidad lineal rectificada paramétrica o *Parametric Rectified Linear Unit* (PReLU) es una variante de la función LReLU, con la diferencia de que el valor del parámetro α no es un valor fijo introducido por el usuario, sino que es un parámetro que la red "aprende" durante su entrenamiento para conseguir un valor óptimo que mejore el modelo. Al igual que la LReLU, también consigue aumentar la velocidad de aprendizaje al salvar algunas neuronas de caer en el efecto de la "ReLU moribunda" [23].

A la hora de describir esta función se puede definir de la misma manera que LReLU según la ecuación 2.5. Por esta razón, ocurre lo mismo a la hora de representar la función unidad lineal rectificada paramétrica o *Parametric Rectified Linear Unit* (PReLU) y LReLU para los mismos valores de α.

2.2.1.6 Unidad lineal exponencial (ELU)

La función de activación unidad lineal exponencial o *Exponential Linear Unit* (ELU) es una función que al igual que las funciones ReLU, LReLU o PReLU, conserva las entradas positivas a la salida de la función y modifica las entradas negativas. En este caso, al igual que en la función LReLU, unidad lineal exponencial o *Exponential Linear Unit* (ELU) contiene un hiperparámetro α predefinido y establecido generalmente con el valor de *1*. El objetivo de este hiperparámetro es controlar el valor de "saturación" de la función ELU, que se puede definir como la variación de la información que se propaga hacia la siguiente capa [24].

La función se puede definir según la ecuación 2.6.

$$f(x) = \begin{cases} \alpha^{e^x - 1} & si \quad x \leq 0 \\ \\ x & si \quad x > 0 \end{cases} \tag{2.6}$$

Por su parte, la Figura 2.8 muestra una representación de una función ELU para $\alpha = 1$.

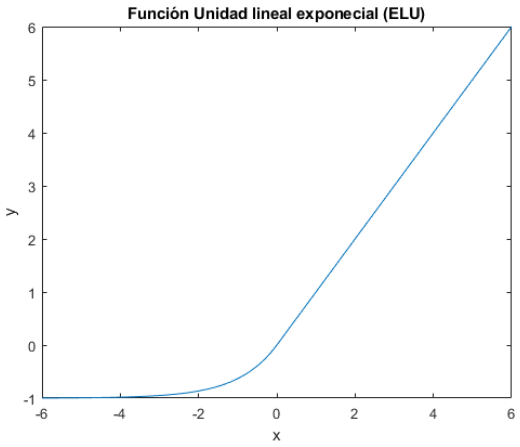


Figura 2.8: Representación de la función ELU con $\alpha = 1$

Las principales desventajas de esta *función de activación* es que la velocidad de cálculo es más lenta que la ReLU y sus variantes debido al uso de una función exponencial, aunque durante el proceso de entrenamiento puede compensarse con una tasa de convergencia más rápida [22].

2.2.1.7 Softmax

La función de activación *softmax* devuelve una distribución de probabilidad para cada una de las clases reconocidas en una ANN. Está diseñada para trabajar en sistemas de clasificación multiclase utilizando datos continuos, siendo uno de los principales recursos utilizados en las capas de salida de un clasificador. Para este caso, la función calcula las probabilidades de cada muestra sobre todas las clases del sistema para facilitar posteriormente la determinación final de a qué clase pertenece [25]. Esta función se puede definir según la ecuación 2.7.

$$\sigma(x)_i = \frac{e^{x_i}}{\sum_{j=0}^{k} e^{x_j}} i = 0, 2, ...k \tag{2.7}$$

Donde:

- $\sigma(x)_i$ es la salida de la *función de activación softmax* para un valor i del vector de entrada x.
- x_i representa un valor i del vector de entrada x.
- k se corresponde al número de muestras totales en el vector de entrada x.

Como podemos ver en la ecuación 2.7, la fórmula calcula la exponencial del valor de entrada dado y lo normaliza dividiéndolo entre la suma de los valores exponenciales de todos los valores en las entradas. La principal ventaja de usar softmax es el rango de probabilidades de salida, siendo éste de *0* a *1*, y la suma de todas las probabilidades igual a *1* [26].

2.2.2 Tamaño de lote (*Batch Size*)

Normalmente, a la hora de entrenar una ANN con una cantidad considerable de datos, se suelen utilizar particiones que se suministran a la red denominados "*lotes*" o "*minilotes*" dependiendo del tamaño de éstas. Estas particiones definen la cantidad de muestras que se propagan por la ANN al mismo tiempo durante cada entrenamiento [27].

De este modo, si por ejemplo tenemos *1000* muestras como conjunto de entrenamiento y desea configurar un *tamaño de lote* de *100* muestras, la ANN toma las primeras 100 muestras del conjunto de datos de entrenamiento y las usa para entrenar la red. Después, toma las siguientes *100* muestras y entrena la red nuevamente, repitiendo este proceso hasta que se utilicen todas las muestras del conjunto de entrenamiento [28].

Las ventajas que puede tener usar un *tamaño de lote* respecto al número de todas las muestras para una red con una gran cantidad de datos principalmente son que requiere menos memoria y que por lo general, las redes entrenan más rápido con minilotes. Eso es porque actualizamos los pesos después de cada propagación [28]. Una cosa a tener en cuenta para conseguir que esta práctica resulte útil consiste en que si se trabaja con *tamaños de lote* excesivamente pequeños se puede provocar que la ANN pierda precisión, por lo que la elección correcta del *tamaño de lote* se corresponde con encontrar el mejor equilibrio entre la eficiencia y la capacidad de la memoria disponible [27].

Si se entrena una ANN haciendo uso de minilotes, para estabilizar el proceso de aprendizaje y reducir drásticamente el número de épocas de entrenamiento necesarias para entrenar se suele emplear una técnica llamada *normalización por lotes* (*Batch Normalization*). La *normalización por lotes* se encarga de estandarizar las entradas a una capa de la ANN para cada minilote. Esta estandarización de entradas se puede aplicar en cada entrada de las capas ocultas, o justo antes o después de aplicar una función de activación en las capas ocultas, pudiendo resultar beneficioso realizar la estandarización antes de aplicar la función de activación si los valores de las entradas a la función de activación no se corresponden con una distribución gausiana. En la práctica, es común permitir que la capa aprenda dos nuevos parámetros, una nueva media y desviación estándar conocidos como β y γ respectivamente, que permiten el escalado y desplazamiento automático de las entradas de la capa estandarizada. El modelo ajusta estos parámetros como parte del proceso de entrenamiento [29].

2.2.3 Optimizadores

Durante el entrenamiento de las ANNs, el objetivo es minimizar los errores entre las predicciones y los "resultados reales" encontrando los pesos adecuados que permitan una buena generalización de las redes.

Para que esto ocurra es necesario hacer uso de un *optimizador*, el cual es el encargado de ajustar los pesos para conseguir mejores resultados en la ANN. La efectividad de estos pesos se calcula a través de una *función de pérdida* (*loss function*) descrita en la sección 2.2.4) [30].

2.2.3.1 Descenso de gradiente

Uno de los *optimizadores* más populares es el "descenso por Gradiente o *Gradient Descent*". Puesto que "gradiente", en términos sencillos, significa pendiente o inclinación de una superficie, descender por el gradiente significa literalmente descender una pendiente para alcanzar el punto más bajo de esa superficie [31]. Para entender de una manera sencilla el funcionamiento del descenso por Gradiente o *Gradient Descent* (GD), se va a hacer uso de un ejemplo citado en [31]:

Imaginemos una gráfica bidimensional como una parábola representada en la Figura 2.9.

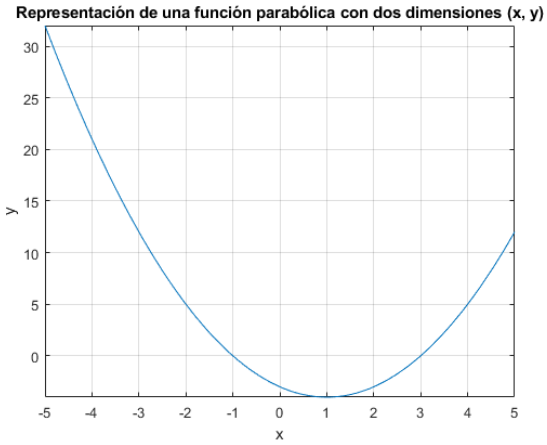


Figura 2.9: Representación de la función parabólica: $x^2 - 2x - 3$

El mínimo valor de la parábola en la dimensión Y se produce en x = 1. El objetivo de un algoritmo de GD es encontrar el valor de X que se corresponda con ese valor mínimo de Y. En este caso *"Y"* sería denominada como función objetivo sobre la que opera el algoritmo de GD para descender al punto más bajo.

Con este ejemplo se puede llegar a intuir que el funcionamiento de un optimizador de GD se basa en calcular el gradiente (derivada parcial) de la *función de costes* para cada peso de la red. Como el objetivo es minimizar el error, se modificará cada peso en la dirección del gradiente (dirección negativa) [32]. Al final, el GD es un algoritmo iterativo que comienza a operar desde un punto aleatorio en una función y se desplaza por su pendiente mediante varios pasos hasta alcanzar un mínimo local [31]. El descenso por Gradiente o *Gradient Descent* se puede definir mediante la ecuación 2.8.

$$x_{n+1} = x_n - \alpha \cdot \nabla f(x_n) \tag{2.8}$$

Donde:

- x_n es el valor de la conjetura n en la función de pérdidas *f(x)*.
- x_{n+1} representa el valor de la siguiente conjetura.
- $\nabla f(x_n)$ se corresponde a la derivada parcial de la función de pérdidas *f(x)* para la conjetura n.

- α se trata del hiperparámetro conocido como *tasa de aprendizaje*.

A la hora de utilizar el optimizador GD, uno de los hiperparámetros más importante de una ANN es la *tasa de aprendizaje* (*"learning rate"*). Ajustar este hiperparámetro a un valor óptimo es crucial ya que un valor de *learning rate* muy pequeño puede hacer que el aprendizaje de la red se demore bastante, mientras que un valor de *learning rate* muy grande confiere un comportamiento erróneo al aprendizaje de la red [30].

Habiendo explicado su funcionamiento y a pesar de que el optimizador de GD se utilice bastante, su algoritmo presenta las limitaciones de que solo funciona cuando nuestra función es diferenciable en todas partes y que solo encuentra mínimos locales (en lugar del mínimo global) [31].

2.2.3.2 Descenso de gradiente estocástico

El optimizador de descenso por Gradiente estocástico o *Stochastic Gradient Descent* (SGD) consiste en una aproximación estocástica del GD, puesto que reemplaza el gradiente real calculado sobre el conjunto de datos completo por una estimación del mismo calculado a partir de un subconjunto de datos seleccionado al azar con el objetivo de reducir la carga computacional y lograr iteraciones más rápidas [30]. De manera similar que en el GD, *la tasa de aprendizaje* es crucial para este algoritmo. Para el GD describimos que la tasa de aprendizaje se manifestaba como un hiperparámetro fijo sin embargo, para el caso del SGD, *la tasa de aprendizaje* consiste en un parámetro cuyo valor se va ajustado disminuyéndose gradualmente en cada iteración a medida que la ANN se va entrenando [30].

2.2.3.3 Adagrad

El algoritmo de Gradiente adaptativo o *Adaptive Gradient Algorithm* (ADAGRAD) es una modificación del SGD en el que *la tasa de aprendizaje*, en vez de considerarse un valor uniforme para todos los parámetros del sistema, se adapta individualmente para cada parámetro teniendo en cuenta la frecuencia con la que se actualiza durante el entrenamiento. Así, el valor de *la tasa de aprendizaje* aumenta para los parámetros más dispersos y disminuye para los que son menos dispersos [33]. Como aspecto negativo, si se trabaja con parámetros muy dispersos, esto puede hacer que la tasa de aprendizaje sea cada vez más pequeña hasta un punto que sea imposible actualizar el parámetro de manera efectiva [33].

2.2.3.4 Adadelta

El optimizador *Adadelta* es una modificación del algoritmo de optimización ADAGRAD. Este optimizador fue diseñado para corregir el problema presente a la hora de trabajar con parámetros demasiado dispersos, donde al hacerse tan pequeño el valor de la *tasa de aprendizaje* podría provocaba que la optimización para estos parámetros no fuera del todo buena al ser imposible actualizar los parámetros en cuestión. La solución que se propone es restringir una ventana de tamaño fijo de los últimos gradientes en vez de tener en cuenta cada gradiente desde el principio del entrenamiento. Así pues, *Adadelta* continúa aprendiendo incluso cuando se han hecho muchas actualizaciones [34].

2.2.3.5 RMSProp

El optimizador *Root Mean Square Propagation* conocido popularmente como *RMSProp* o *Propagación cuadrática media*, utiliza una *tasa de aprendizaje* adaptativa para cada parámetro, al igual que los optimizadores ADAGRAD y *Adadelta*), en lugar de utilizar una tasa de aprendizaje fija como un hiperparámetro.

Para adaptar la *tasa de aprendizaje* de cada parámetro se normaliza su gradiente utilizando un promedio móvil de los gradientes de las iteraciones recientes al cuadrado. Al igual que *Adadelta*, *RMSProp* se encarga de resolver el problema de trabajar con parámetros demasiado dispersos en *Adagrad* [35].

2.2.3.6 Adam

El optimizador estimación de momento adaptativo o *ADAptive Moment estimation* (ADAM) combina las ventajas de ADAGRAD y *RMSProp*. Por parte de *AdaGrad* adopta la capacidad para lidiar con gradientes dispersos y por parte de *RMSProp*, la capacidad para lidiar con objetivos no estacionarios. También, al igual que estos dos optimizadores, ADAM mantiene una *tasa de entrenamiento* individual para cada parámetro pero se ajusta en función de las estimaciones del primer momento (la media) y segundo momento (la varianza) de los gradientes (derivadas parciales) de los pesos de la red [36].

2.2.4 Funciones de pérdida

Como se indica en [37], las *funciones de pérdida* o *loss functions* son funciones que evalúan la desviación entre las predicciones realizadas por la red neuronal y los valores reales durante el aprendizaje. Una red neuronal se considera más eficiente cuanto menor es el resultado de la *función de pérdida* que se aplique. La reducir al mínimo de la desviación entre los valores predichos y los valores reales se da cuando los pesos de la ANN están ajustados al máximo [14]. A continuación abordaremos las funciones de pérdida más significativas.

2.2.4.1 Error cuadrático medio

El error cuadrático medio o *Mean Square Error* (MSE) es una función muy popular que se encarga de calcular, de forma global, el porcentaje de error cometido por nuestra ANN, midiendo la diferencia entre el estimador y la predicción de la ANNs. El empleo del error cuadrático medio o *Mean Square Error* (MSE) es útil cuando se tratan problemas de regresión donde se analiza la relación entre varias variables [14]

De esta forma, podemos definir el MSE según la ecuación 2.9.

$$MSE = \frac{1}{n} \cdot \sum_{i=1}^{n} (Y_i - Pred_i)^2 \tag{2.9}$$

Donde:

- Y consiste en el vector de valores reales.
- $Pred$ representa el vector de predicciones.
- n se corresponde al número de muestras totales.

Una de las ventajas de MSE es que se puede aplicar a cualquier ANN que contenga funciones de activación diferenciables. También, esta función es muy útil si se trabaja con valores muy altos o bajos respecto a la media de las muestras [38].

2.2.4.2 Entropía cruzada

La función de pérdida de *entropía cruzada* se puede utilizar como función de pérdida en redes neuronales de aprendizaje automático. Esta función se suele emplear en modelos de ANNs cuyas salidas representan

una probabilidad [39]. Se puede definir mediante la ecuación 2.10.

$$C = -\frac{1}{n} \cdot \sum_{i=1}^{n} [Y_i \cdot \ln Pred_i + (1 - Y_i) \cdot \ln(1 - Pred_i)] \tag{2.10}$$

Donde:

- C se trata de la función de pérdida de entropía cruzada.
- Y consiste en el vector de valores reales.
- $Pred$ representa el vector de valores de las predicciones.
- n se corresponde al número de muestras totales.

Existen diferentes variantes de esta función que se pueden aplicar en las ANNs dependiendo de la forma de clasificar las salidas, que describimos a continuación.

2.2.4.2.1 Entropía cruzada categórica La función de pérdida de *entropía cruzada categórica*, también llamada *pérdida logarítmica*, resulta útil cuando se utiliza en ANNs que asignen a su salida probabilidades para múltiples eventos asignando una sola etiqueta de salida. Suele estar precedida de una función de activación *Softmax*, descrita en la sección 2.2.1.7, en la última capa [39]. Cada probabilidad de clase predicha se compara con la salida *0* o *1* deseada de la clase real y se calcula un valor que penaliza la probabilidad en función de qué tan lejos está respecto al valor real. Esta penalización es de naturaleza logarítmica, lo que genera valores muy grandes para las probabilidades cercanas a *1* y valores muy pequeños para las probabilidades que tienden a *0* [39]. La *entropía cruzada categórica* se define según la ecuación 2.11.

$$LCE = -\frac{1}{n} \cdot \sum_{i=1}^{n} \sum_{j=1}^{m} [Y_{ij} \cdot \log(Pred_{ij})] \tag{2.11}$$

Donde:

- LCE en este caso se trata de la función de pérdida de *entropía cruzada categórica*.
- Y consiste en el vector de valores reales.
- $Pred$ representa el vector de predicciones.
- m se corresponde al número de muestras totales.
- n se corresponde al número de clases de eventos a clasificar.

2.2.4.2.2 Entropía cruzada binaria La función de pérdida de *entropía cruzada binaria* resulta útil cuando se utiliza en ANNs que se encargan de realizar una clasificación binaria, aunque también se puede utilizar para resolver problemas de clasificación múltiple. En general, esta función se suele utilizar en sistemas que contengan una función de activación *"sigmoide"* en la última capa [39]. La *entropía cruzada binaria* se define como se muestra en la ecuación 2.12 [40].

$$LB = -\frac{1}{n} \cdot \sum_{i=1}^{n} [Y_i \cdot \log(Pred_{ij} + (1 - Y_i) \cdot \log(1 - Pred_i))] \tag{2.12}$$

Donde:

- LB en este caso se trata de la función de pérdida de *entropía cruzada binaria*.
- Y consiste en el vector de valores reales binario
- $Pred$ representa el vector de valores de las predicciones binario
- n se corresponde al número de muestras totales.

2.2.4.2.3 Entropía cruzada categórica dispersa La función de pérdida de *entropía cruzada categórica dispersa* es una variante de la función de pérdida de *entropía cruzada categórica* definida anteriormente (2.2.4.2). La *entropía cruzada categórica dispersa* se puede definir de la misma forma que la *entropía cruzada categórica* mediante la ecuación 2.11. La única diferencia que tienen es en cómo se definen las etiquetas que se suministran a la ANN. Esta diferencia se puede entender de una manera muy simple con el ejemplo descrito en [41]:

- La entropía cruzada categórica se usa cuando las etiquetas verdaderas están codificadas en un solo uso, por ejemplo, para el problema de clasificación de 3 clases diferentes se tendrían los siguientes valores: *[1,0,0]*, *[0,1,0]* y *[0,0,1]*.
- En el caso de la entropía cruzada categórica dispersa, las etiquetas verdaderas están codificadas en números enteros, por ejemplo, la codificación para el problema de clasificación de 3 clases se realizaría de la siguiente manera: *[1]*, *[2]* y *[3]*.

2.2.5 Regularización

Para el caso del aprendizaje supervisado, un modelo que es capaz de realizar predicciones correctamente de datos que no han sido utilizados para el proceso de entrenamiento se considera un modelo que está bien ajustado. Ajustar un modelo no es una tarea fácil, hasta tal punto que el *sobreajuste* (*overfitting*) y el *desajuste* (*underfitting*) de los modelos pueden llegar a ser uno de los problemas más comunes a la hora de generar modelos robustos [42].

Se denomina *sobreajuste* al hecho de que un modelo esté tan ajustado a los datos de entrenamiento que haga que no generalice bien el resto de datos "no conocidos por el sistema". El *sobreajuste* se produce cuando un sistema de aprendizaje automático se entrena demasiado o con datos anómalos, lo que provoca que el algoritmo "aprenda" patrones que no son generales. De este modo, si un modelo tiene como concepto clave el de entrenarse para identificar patrones generales que sean extrapolables a nuevos datos pero se encuentra *sobreajustado*, lo más normal es que se vuelva ineficiente [43]. Por otro lado, si el modelo de aprendizaje no puede capturar la naturaleza inherente de los datos, provocando que el modelo no se ajuste bien, se puede decir que el modelo esta *desajustado*.

Para las ANNs, la regulación es el proceso que se encarga de evitar que un modelo de aprendizaje sufra un sobreajuste sobre los datos de entrenamiento, con la finalidad de pretender que el modelo se ajuste correctamente. Esta regulación implica un mecanismo para reducir los errores de generalización del modelo de aprendizaje [42].

Es importante recalcar que en muchas ocasiones, generar el mejor modelo que sea capaz de identificar y distinguir los patrones de las distintas clases de eventos para los que se está entrenando el sistema es realmente un problema muy complejo. Por este motivo, y en la práctica, el algoritmo de aprendizaje no tiene porqué ser capaz de generarlo, sino que se centra en generar un modelo que reduzca de manera significativa el error de entrenamiento. Esto quiere decir que para un mismo sistema se pueden generar varios modelos óptimos diferentes que se consideren *ajustados*.

Para que se pueda entender de una manera más gráfica, en la Figura 2.10 se pueden observar tres representaciones en donde el modelo se encuentra desajustado, ajustado y sobreajustado respectivamente [42].

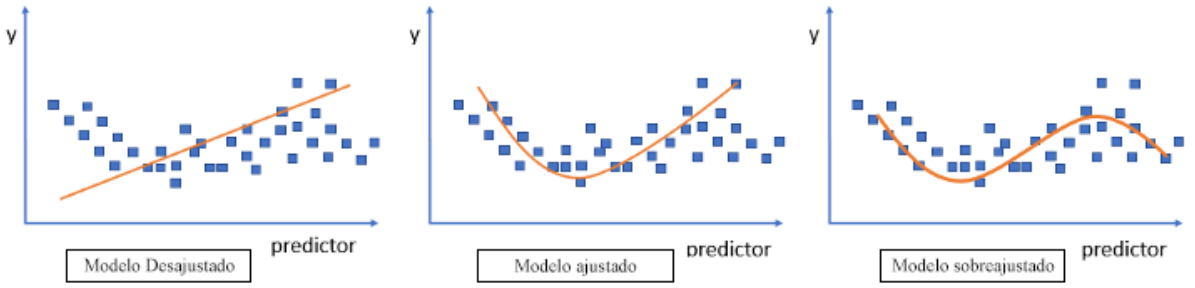


Figura 2.10: Ejemplos de ajustes de curvas en el modelo de aprendizaje.

El *error de generalización* es una escala para medir la capacidad del modelo de aprendizaje para predecir correctamente la respuesta a datos no vistos. Puede minimizarse evitando sobreajustar los datos de entrenamiento. Además del aumento de datos a utilizar durante el entrenamiento, la *"regularización L1"*, la *"regularización L2"*, la *"parada temprana"* y el "dropout" son técnicas de regularización importantes para ayudar a mejorar la generalización de un modelo de aprendizaje [42].

2.2.5.1 Regularización L1

Un modelo que aplique la *regularización L1* o *regularización Lasso* modifica la *función de pérdida* agregando un nuevo término de regularización para evitar el sobreajuste del modelo [42]. Este nuevo término se conoce como *L1* y se define según la ecuación 2.13.

$$L1 = \lambda \cdot \underbrace{\frac{1}{m} \cdot \sum_{j=1}^{m} |w_j|}_{\Omega} \tag{2.13}$$

Donde:

- λ es un hiperparámetro que regula la contribución de Ω relativa a la función. Para $\lambda = 0$ se anula la regularización mientras que para valores más altos la regularización es mayor.
- Ω consiste en la media del valor absoluto de los parámetros del sistema.
- w se corresponde con el valor de un parámetro del sistema.
- wm representa el número de parámetros totales del sistema.

Para distintos valores de λ y Ω se pueden calcular soluciones más adecuadas para la función de pérdida original de cara a conseguir un ajuste adecuado del modelo. Aplicando *L1* a la "nueva función de pérdida", J en la ecuación 2.14, quedaría de la siguiente forma [42]:

$$J = \textit{Función de pérdida} + \underbrace{\lambda \cdot \frac{1}{m} \cdot \sum_{j=1}^{m} |w_j|}_{L1} \tag{2.14}$$

Si utilizamos la función de pérdida de error cuadrático medio, descrito en 2.2.4.1, la nueva función de pérdida se quedaría como:

$$J = \frac{1}{n} \cdot \sum_{i=1}^{n} (Y_i - Pred_i)^2 + \lambda \cdot \frac{1}{m} \cdot \sum_{j=1}^{m} |w_j| \tag{2.15}$$

Como se puede apreciar, en caso de utilizar la regularización L1, al tomar el valor absoluto de la pendiente en la ecuación 2.13, los parámetros del sistema más pequeños eventualmente desaparecerán y se convertirán en *0*, por lo que la *regularización L1* ayuda a seleccionar características que son importantes y convertir el resto en ceros [44].

2.2.5.2 Regularización L2

Un modelo que aplique la *regularización L2* o *regularización Ridge*, al igual que la *regularización L1* también modifica la *función de pérdida* agregando un nuevo término de regularización para evitar el sobreajuste del modelo [42]. Esta vez, este término es conocido como *L2* y se define según la ecuación 2.16.

$$L2 = \lambda \cdot \underbrace{\frac{1}{2 \cdot m} \cdot \sum_{j=1}^{m} (w_j)^2}_{\Omega} \tag{2.16}$$

Donde:

- λ es un hiperparámetro que regula la contribución de Ω relativa a la función. Para $\lambda = 0$ se anula la regularización mientras que para valores más altos la regularización es mayor.
- Ω consiste en la media del cuadrado de los parámetros del sistema.
- w se corresponde con el valor de un parámetro del sistema.
- m representa el número de parámetros totales del sistema.

Aplicando *L2* a la "nueva función de pérdida", J en la ecuación 2.17, quedaría como:

$$J = \textit{Función de pérdida} + \underbrace{\lambda \cdot \frac{1}{m} \cdot \sum_{j=1}^{m} (w_j)^2}_{L1} \tag{2.17}$$

La *regularización L2* funciona mejor cuando la mayoría de los parámetros del sistema son relevantes, haciendo que cuando varios de los atributos de entrada se encuentran correlados entre ellos, esta regularización se encargue de que los parámetros acaben tomando valores más pequeños. Consiguiendo con esta disminución de los parámetros minimizar el efecto de la correlación entre los atributos de entrada y una mejor generalización el modelo [44].

2.2.5.3 Parada anticipada *Early Stopping*

Un gran número de iteraciones o épocas de entrenamiento puede conducir a un *sobreajuste* del modelo de aprendizaje sobre un conjunto de datos mientras que un número menor de épocas resulta en un *desajuste* del modelo de aprendizaje. La *parada anticipada* o *parada temprana* (*early stopping*), es un tipo de regularización muy popular que se implementa para evitar el *sobreajuste* de un modelo de aprendizaje [26].

Durante un entrenamiento, una ANN se entrena con *datos de entrenamiento* y simultáneamente se valida (poniendo a prueba la red) con *datos de prueba* (también conocidos como *datos de validación*).

Al utilizar el método de regularización de *parada anticipada*, esto se puede utilizar para determinar el número de iteraciones necesarias para conseguir una convergencia mejor del modelo de aprendizaje antes de que su error de validación (error que se comete al evaluar la red con el conjunto de datos de validación) comience a aumentar [26]. Tal y como refleja la Figura 2.11, un modelo de aprendizaje se mejora a medida

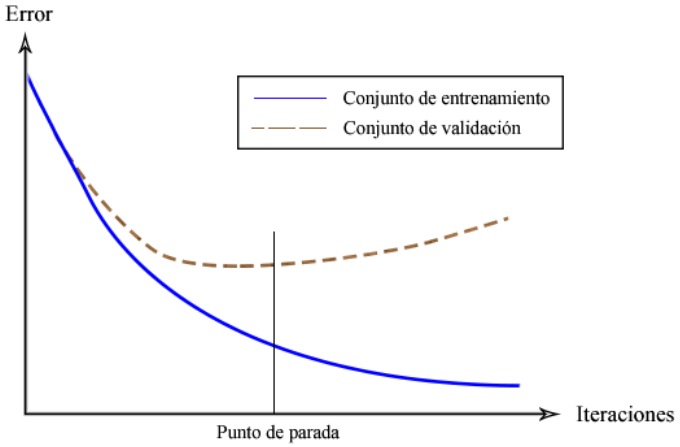


Figura 2.11: Representación del *error de entrenamiento* y *error de validación* durante varias interacciones de un entrenamiento

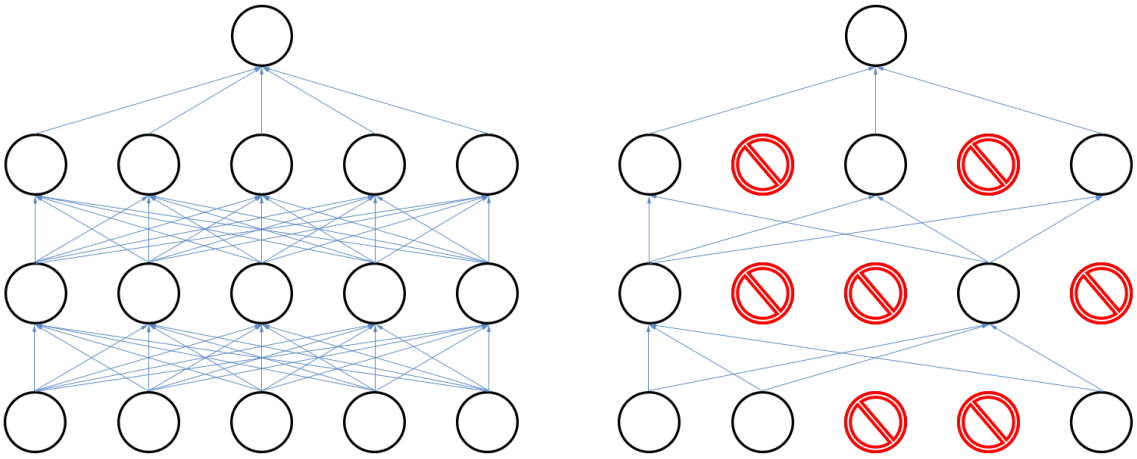
que aumenta el número de iteraciones de entrenamiento hasta el punto en que puede sobreajustarse a los datos de entrenamiento, en caso de que no se detenga en una iteración adecuada. El rendimiento del modelo frente a los datos de entrenamiento se puede mejorar aún más con un mayor número de iteraciones de entrenamiento, pero esto causa un mayor error de validación. Con la parada temprana, una ANN obtiene pesos óptimos para una mejor convergencia del aprendizaje con un error de generalización menor [26].

2.2.5.4 Dropout

El *Dropout* o *abandono* es un método de regularización utilizado en las redes neuronales para evitar que las neuronas memoricen una parte de la entrada provocando un *sobreajuste* del modelo entrenado. Al utilizar *dropout* se utiliza un algoritmo basado en "minilotes", descrito en la sección 2.2.2, como cuando se aplica el *gradiente estocástico*, sección 2.2.3.2. El *dropout* provoca que para cada vez que se cargue un "minilote" durante la fase de entrenamiento de una ANN, se desactiva aleatoriamente un porcentaje de "conexiones" hacia algunas neuronas en cada capa oculta. Esta desactivación se produce acorde a una probabilidad de descarte previamente definida establecida entre *0* y *1*. Cuando los valores de esta probabilidad son más próximos a *0*, el *dropout* desactivará cada vez menos neuronas, mientras que si los valores son cercanos a *1*, ocurrirá el caso contrario. Esta probabilidad puede ser igual para toda la red o distinta para cada capa [45].

La Figura 2.12 muestra un esquema de conexiones de las neuronas de una red neuronal donde se puede apreciar la diferencia entre utilizar y no utilizar *dropout*.

Una vez tengamos el modelo listo para realizar predicciones sobre nuevas muestras, hay que compensar de algún modo el hecho de que no todas las neuronas permanecieran activas durante el entrenamiento. Para ello, una de las soluciones podría ser multiplicar todos los parámetros de la red por la probabilidad de no descarte [45].



(a) Red neuronal sin utilizar *dropout*

(b) Red neuronal utilizando *dropout*

Figura 2.12: Representación de las conexiones entre las neuronas de una ANN

2.3 Tipos de arquitecturas de redes neuronales artificiales

Como se ha comentado anteriormente al introducir las ANNs y al explicar sus conceptos básicos, sección 2.2, una ANN esta formada por una o más capas, distinguiendo entre *redes monocapa* y *redes multicapa*. A la hora de resolver problemas que requieran una gran complejidad computacional lo más habitual es tratar con *redes multicapa* ya que las *redes monocapa* poseen una gran limitación al procesar la información de entrada para reconocer patrones.

Dentro de las *redes multicapa* existe una gran variedad de tipos de arquitecturas especificas reconocidas. En este apartado se va a describir algunas de las arquitecturas típicas más utilizadas durante el diseño de una ANN.

2.3.1 Redes neuronales recurrentes (RNNs)

Una *red neuronal recurrente* o *RNN* es un tipo de arquitectura de una ANN basada en la generalización de una red *feedforward*, que a su vez realiza otra serie de conexiones para otorgar una "memoria" a la red [14].

La característica principal de una RNN genérica es que además de transmitir la salida de la neurona hacia la siguiente capa, también se transmite la salida hacia la entrada de la propia neurona creando una especie de "realimentación". Por lo que para cada instante de tiempo, una neurona recibe dos entradas: el conjunto de entradas procedentes de la capa anterior y su salida correspondiente al instante de tiempo anterior. Su estructura se puede ver en detalle en la Figura 2.13 [14].

Puesto que la salida de las neuronas de las RNNs en un instante de tiempo determinado contienen "información" de instantes de tiempo anteriores, se puede decir que una RNN tiene una cierta "memoria". Esta "memoria" es lo que hace que este tipo de redes sean muy adecuadas para resolver problemas de aprendizaje automático que involucran datos secuenciales como puede ser el reconocimiento del habla y el lenguaje; ya que hace posible "recordar" información que puede resultar relevante sobre la entrada

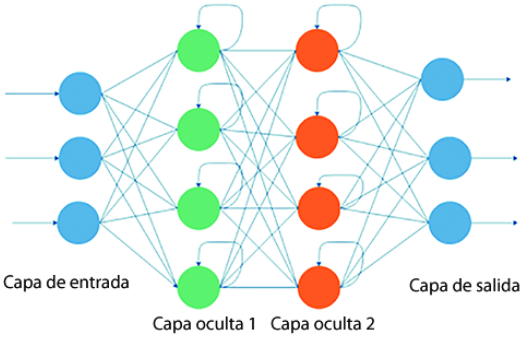


Figura 2.13: Representación de un ejemplo de una arquitectura básica de una RNN. Reproducido de [1]

que recibieron. Este hecho les ayuda a ser más precisas en las predicciones posteriores gracias a poseer una "información de contexto" a diferencia de otros tipos de ANNs Por otro lado, se pueden destacar como aspectos negativos de las RNNs: su gran complejidad a la hora de implementarse y las grandes capacidades de cálculo que requirieren durante el entrenamiento de la red [14].

2.3.2 Redes de memoria a corto y largo plazo (LSTMs)

Las redes de memoria de largo-corto plazo o *Long-Short Term Memory Networks* (LSTMs), son una extensión de las RNNs que amplían su memoria para retener valores importantes que se han procesado hace una larga secuencia de instantes de tiempo, permitiendo recordar sus entradas durante un largo período de tiempo. Esto se debe a que las LSTMs contienen su información en la "memoria", donde una neurona puede leer, escribir y borrar información de su "memoria" en función de la "relevancia" que asigna a la información que está recibiendo. La asignación de esta "relevancia" se decide a través de unos pesos que se ajustan durante el entrenamiento [14].

Una neurona de una red de memoria de largo-corto plazo o *Long-Short Term Memory Network* (LSTM) posee tres puertas que se encargan de modificar la información que retiene la "memoria" de la neurona [14]:

- Puerta de entrada (input gate): La puerta de entrada determina la cantidad de información sobre el estado actual de la red que debe almacenarse en "memoria".
- Puerta de olvido (forget gate): Determina la cantidad de información que se encuentra en la "memoria" que debe descartarse.
- Puerta de salida (output gate): Determina la cantidad de información de la entrada y la memoria del estado actual que se utilizará para decidir la salida.

En la Figura 2.14 se puede observar la estructura de una neurona para un instante de tiempo t perteneciente a una red *LSTM* [46] donde:

- C_t se trata de la información de la "memoria" en el instante de tiempo t que se transmite al siguiente instante de tiempo $t+1$ de la misma neurona. Al igual que C_{t-1} es la información de la "memoria" que se transmite desde el instante de tiempo t-1 a el instante de tiempo t.
- I_t se trata de la entrada de la neurona procedente de la capa anterior para el instante de tiempo t.

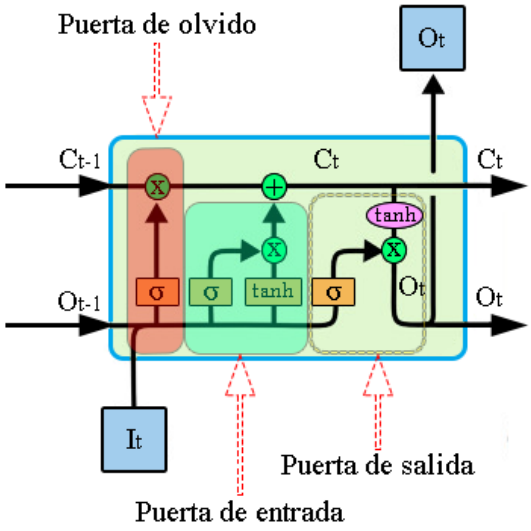


Figura 2.14: Representación de una neurona de una Red *LSTM* que se encuentra en el instante de tiempo *t*.

- O_t se corresponde con la salida de la neurona correspondiente para el instante de tiempo *t*, la cual se transmitirá a la siguiente capa y a la entrada de esta misma neurona para el instante de tiempo *t+1*. Del mismo modo O_{t-1} se corresponde a la salida de la neurona correspondiente para el instante de tiempo *t-1*.
- Los distintos símbolos σ representan la aplicación de una *sigmoidea* cada uno, que se encarga de generar números entre *0* y *1*, describiendo la cantidad de información que se deja pasar en cada una de las tres puertas [46].
- *tanh* función *tangente hiperbólica* que convierte los valores en un rango entre *-1* y *1* [46].

2.3.3 Redes de unidades recurrentes con puerta (GRUs)

Las redes que utilizan unidades recurrentes con puertas o *Gated Recurrent Units (GRUs)* o simplemente *redes* GRU, son una variante de las redes LSTM. La principal diferencia que hay entre estas dos arquitecturas consiste en que las redes GRUs no tienen que usar una "unidad de memoria" para controlar el flujo de información con el objetivo de evitar el problema de la dependencia a largo plazo [47].

Las *redes GRUs* tienen menos parámetros por lo que su proceso de entrenamiento es más rápido y suele hacer falta una menor cantidad de datos para encontrar generalizaciones sin embargo, una *red LSTM* se ha demostrado que normalmente es más efectiva a la hora de trabajar con una gran cantidad de datos, ya que puede llegar a proporcionar mejores resultados ante contextos complejos [48].

Para este caso, una neurona de una *red GRU* posee "únicamente" *dos* "puertas" a diferencia de las *tres* "puertas" que tenía una neurona de una *red LSTM* [48]:

- Puerta de reajuste (*reset gate*): Determina la cantidad de información de la entrada que hay que incorporar.
- Puerta de actualización (*update gate*): Determina la cantidad de información que hay que mantener de los instantes de tiempo anteriores.

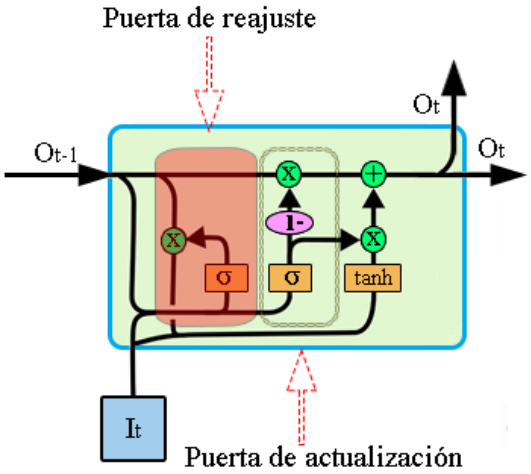


Figura 2.15: Representación de una neurona de una Red *GRU* que se encuentra en el instante de tiempo *t*.

En la Figura 2.15 se puede observar la estructura de una neurona para un instante de tiempo t perteneciente a una red *GRU* [48], donde:

- I_t se trata de la entrada de la neurona procedente de la capa anterior para el instante de tiempo t.
- O_t se corresponde con la salida de la neurona correspondiente para el instante de tiempo t, la cual se transmitirá a la siguiente capa y a la entrada de esta misma neurona para el instante de tiempo $t+1$. Del mismo modo O_{t-1} se corresponde a la salida de la neurona correspondiente para el instante de tiempo $t-1$.
- Los distintos símbolos σ representan la aplicación de una *sigmoidea* cada uno, que se encarga de generar números entre *0* y *1*, que describen la cantidad de información que se deja pasar en cada una de las puertas [46].
- *tanh* función *tangente hiperbólica* que convierte los valores en un rango entre *-1* y *1* [46].
- El bloque "1−" se interpreta como: "*Salida del bloque* = (1 − *Entrada del bloque*)" [48].

2.3.4 Redes neuronales convolucionales (CNNs)

Una CNN o *ConvNet*, es un tipo de ANN que hace uso de la operación de convolución en lugar de la multiplicación general de pesos por entradas en al menos una de sus capas [49].

La convolución es una operación matemática que hace la integral del producto de 2 señales o funciones, con una de las señales invertida y desplazada [50].

Una CNN recibe como información de entrada un conjunto de valores típicamente codificados en forma de matrices y procesa esta información a través de sus capas imitando al cortex visual del ojo humano e identificar así distintas características en las entradas. Para esta tarea, una *CNN* posee varias capas ocultas especializadas en la identificación de patrones con una jerarquía determinada donde por ejemplo, las primeras capas pueden detectar lineas o curvas y las capas más profundas son capaces de detectar patrones más específicos [51].

A diferencia de las *ANNs* convencionales, las capas de una CNN tienen neuronas dispuestas en 3 dimensiones: ancho, alto y profundidad. La palabra "profundidad" en este caso se refiere a la tercera dimensión de un volumen de activación [52]. De este modo, si tenemos una imagen con apenas 28x28 píxeles de alto y ancho en escala de grises, esto equivale a tener 784 neuronas, una por píxel, para la capa que reciba dicha imagen como entrada. Ahora bien, si esta imagen fuera en color RGB (tres colores), el número de neuronas de entrada requeridas sería de 2352 (28x28x3) [51].

Como ya se ha mencionado, las *CNNs* contienen al menos una capa que hace uso de la operación de convolución, estas capas son conocidas como *capas convolucionales*. Su función es realizar la convolución de dos matrices, donde una de ellas es una matriz que se le asigna un tamaño constante llamada *kernel* o matriz "K", y la otra se trata de una matriz "M" variable correspondiente a una "porción" de la imagen de entrada de la capa de convolución con el mismo tamaño que "K". Esta operación se repite un número de veces determinado que depende del valor que se le asigne a una constante llamada "*stride*", que se define como el número de píxeles que avanza la ventana "M" en cada cálculo. Para un valor de $stride = 1$, la operación de convolución se repite para cada píxel de la imagen donde se pueda extraer una ventana del mismo tamaño de "K" [53].

En el caso de no querer reducir el tamaño de la imagen de entrada de una *capa de convolución*, a la imagen resultante de las operaciones de convolución se le puede añadir una serie de píxeles conocidos como "*padding*" para conservar el tamaño de la imagen de entrada. Estos píxeles pueden tomar diferentes valores, como pueden ser: *0*, *1* o incluso repetir los valores de sus píxeles vecinos [53].

La fórmula que define el tamaño de la imagen de salida de una *capa convolucional* aplicable para los valores de la dimensión de "anchura" o "altura", es la desarrolla en 2.18 [53].

$$N_s = \left(\frac{N_e + 2 \cdot P - K}{S} \right) + 1 \tag{2.18}$$

Donde:

- N_e consiste en el tamaño de la imagen de entrada para la dimensión de "anchura"(eje x) o "altura"(eje y).
- N_s es el tamaño de la imagen de salida para la misma dimensión que represente N_e. Por ejemplo: Si N_e representa el número de píxeles de la altura de la imagen de entrada, N_s sería el número de píxeles de la altura de la imagen de salida.
- P hace referencia al valor de *padding* asignado correspondiente a la dimensión que represente N_e.
- S hace alusión al valor de *stride* escogido para la dimensión que represente N_e.
- K se corresponde con el valor asignado para la dimensión que represente N_e de la matriz *kernel*.

En la Figura 2.16 se muestra de una manera más visual como quedaría una imagen de entrada de dimensiones "$N_e = (6, 6)$" tras aplicar esta serie de convoluciones con una *matriz kernel* de dimensiones "$K = (3, 3)$", *stride* de *1* en las dimensiones de "anchura" y "altura" ($S = (1, 1)$) y sin aplicar *padding* (P):

Normalmente, y de manera similar a una ANN normal, se suele aplicar un mismo tipo de función de activación para cada una de las salidas de las neuronas pertenecientes a una misma capa. En el caso de las CNNs, una función de activación se aplica para cada valor perteneciente a cada píxel de la imagen procedente de la realización de sucesivas convoluciones sobre una imagen de entrada en una capa de convolución. Existe una gran variedad de *funciones de activación*, algunas de estas se han mencionado

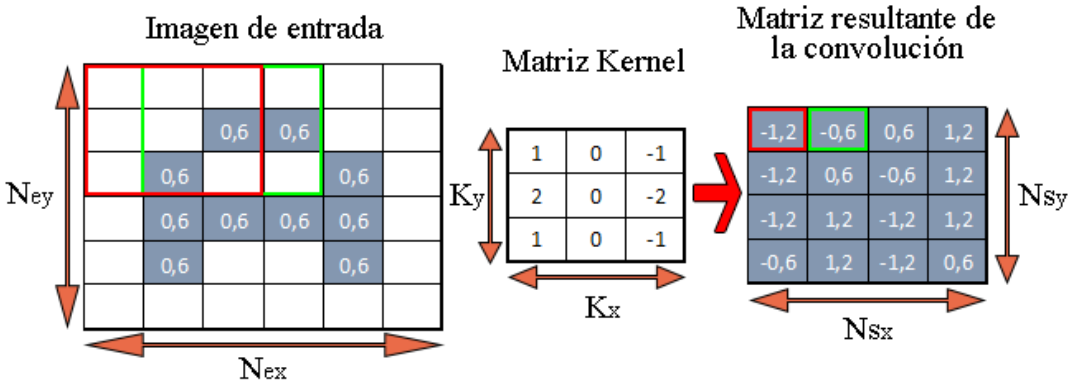


Figura 2.16: Resultado de convolucionar una *matriz Kernel* con la matriz de la imagen de entrada de una capa de convolución.

en el apartado 2.2.1, aunque la *función de activación* más utilizada dentro de las *capas convolucionales* es la función *PReLU* descrita en la sección 2.2.1.5).

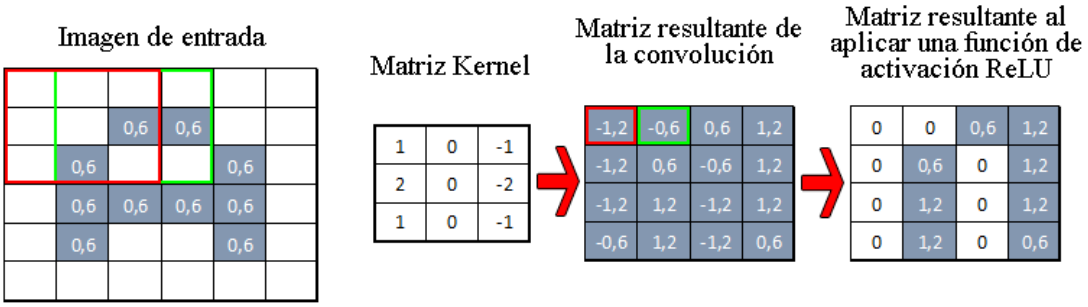


Figura 2.17: Resultado de aplicar una función de activación ReLU en una capa convolucional.

En la Figura 2.17 se puede apreciar el uso de una *función de activación* en una capa convolucional, donde se aplica una función de activación ReLU a la imagen generada tras realizar las convoluciones entre la *matriz de Kernel* y la matriz de la imagen de entrada.

A veces, con el objetivo de reducir el número de parámetros y por ende, el costo computacional de la CNN, se tiende a aplicar unos filtros que reducen el tamaño de la imagen procedente de las convoluciones realizadas a la imagen de entrada de la capa convolucional después de aplicarle una función de activación, si se da el caso. Para este contexto, esta técnica es conocida como *subsampling* y existe una variedad de filtros diferentes que se pueden aplicar con el objetivo de disminuir el tamaño de una matriz. El filtro más usado dentro de las CNNs es el denominado *Max-Pooling*, que se encarga de recorrer la matriz de entrada agrupando un conjunto de valores mediante bloques de un tamaño determinado para seleccionar y conservar los valores más altos dentro de cada bloque, tal y como se puede ver en la Figura 2.18 [51].

Al igual que con la *matriz de kernel*, en la operación de *Max-Pooling* también hay que asignarle un valor de *stride* para indicar cuantos saltos se corresponden con cada desplazamiento En el caso de la Figura 2.18 se le asigna una ventana de *2x2* y un valor de *stride* de *2x2*, por lo que la matriz resultante tendrá sus dimensiones reducidas a la mitad [53].

El conjunto de operaciones que se realizan a partir de una matriz de entrada pueden ir desde convolucionar la imagen con una *matriz kernel*, aplicar una *función de activación*, realizar un *subsampling* o

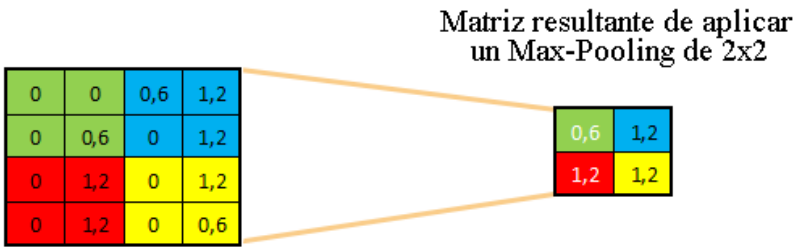


Figura 2.18: Resultado de aplicar un Max-Pooling de *2x2* con un *stride* de *2x2*.

realizar una operación de *Batch Normalization*, dando lugar a lo que es conocido como un *bloque convolucional* dentro de una CNN. En una CNN puede existir más de un bloque convolucional y de hecho, lo normal es trabajar con varios *bloques convolucionales* concatenados. De forma general, la arquitectura de una CNN consta de una capa de entrada seguida de uno o varios *bloques convolucionales*. Posteriormente se suelen utilizar una o varias *capas densas* y una capa de salida, tal y como se puede apreciar en la Figura2.19 [52].

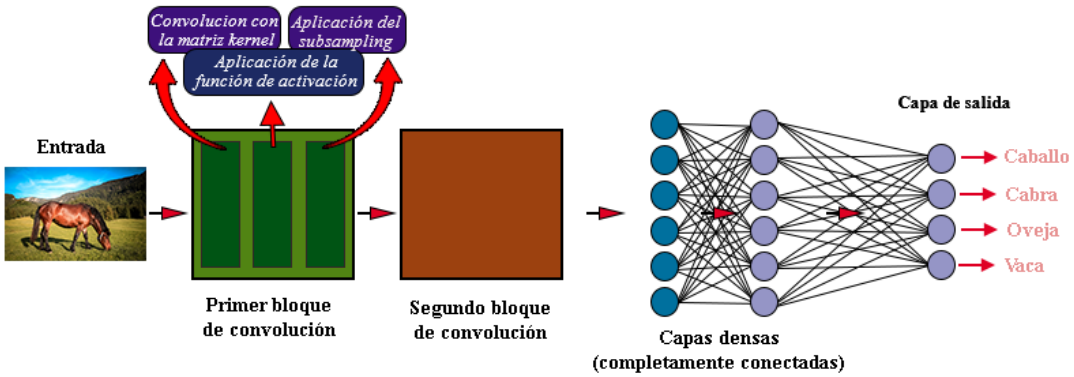


Figura 2.19: Representación de un ejemplo de una arquitectura básica de una CNN.

Una *capa densa*, también conocida como capa completamene conectada o *Fully Connected* (FC) consiste en una capa regular de neuronas como las vistas en una ANN multicapa tradicional, en donde cada neurona de una capa intermedia se conecta a todas las neuronas de las capas anterior y posterior, recibiendo información de todas las neuronas de la capa anterior y transmitiendo información a cada neurona de la capa siguiente. Por este hecho se dice que la capa está densamente conectada [54].

Como se puede apreciar, existe un sin fin de variaciones posibles a la hora de diseñar una CNN, que se pueden incrementar aun más si combinamos las capas pertenecientes a las CNNs con capas de otras arquitecturas de redes neuronales como pueden ser las capas recurrentes de las RNNs, dando lugar a las conocidas redes neuronales convolucionales recurrentes o *Convolutional Recurrent Neural Networks* (CRNNs), utilizadas comúnmente para tareas de Detección de eventos sonoros o *Sound Event Detection* (SED).

Siguiendo con este ejemplo de red "híbrida", una CRNN parte de la misma arquitectura de una CNN pero incorporando una o más capas recurrentes justo después de los bloques convolucionales. Para ello

hay que realizar un redimensionamiento antes de transmitir datos hacia las capas recurrentes ya que las capas convolucionales se desarrollan en vectores de características tridimensionales, mientras que las redes neuronales recurrentes se desarrollan en vectores de características bidimensionales [55].

2.4 Conceptos fundamentales sobre las señales de audio digitales

Una *señal de audio digital* es la codificación digital de una señal eléctrica que representa una onda sonora, tal y como se puede ver en la Figura 2.20. Esta codificación se realiza mediante dos procesos: "*el muestreo*" y "*la cuantificación digital de la señal eléctrica*". Donde

- El *muestreo* de una señal consiste en la selección de ciertos valores de una señal analógica continua para obtener una discreta [56].
- El proceso de *cuantificación digital* se realiza posteriormente al muestreo en el que se toman un conjunto de valores de amplitud de una determinada señal analógica. El objetivo de este proceso es cuantificar con bits estos valores mediante la asignación de niveles, decidiendo si el valor de la muestra está dentro del margen de niveles previamente fijados y se le asigna un valor preestablecido según el código utilizado en la codificación [57]. Cada valor asignado a cada muestra es lo que se conoce como *amplitud* de esa muestra, siendo estos valores para todo conjunto de muestras, la amplitud de la señal.

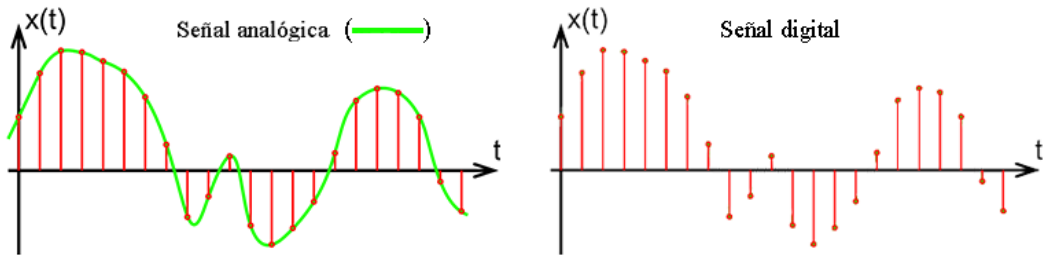


Figura 2.20: Representación de una señal de audio digital.

Las señales de audio digital se pueden almacenar, editar y copiar utilizando computadoras, máquinas de reproducción de audio u otras herramientas, pudiendo manipularlas de diversas maneras. Para comprender como se puede dar la composición de una señal de audio a la hora de registrarla de forma digital es necesario mencionar algunos aspectos fundamentales:

- *Frecuencia*: Es el número de vibraciones por segundo que da origen al sonido analógico. El espectro de un sonido se caracteriza por su rango de frecuencias. Ésta se mide en *Hertzios (Hz)*. El oído humano normalmente capta sólo aquellos sonidos comprendidos en el rango de frecuencias aproximado de entre *20* Hz y *20000* Hz, teniendo en cuenta que la percepción auditiva es relativa y cambia con la persona y la edad [58].

- *Tasa de muestreo (sample rate)*: *La tasa de muestreo*, *frecuencia de muestreo* o *sample rate* se define como la cantidad de valores que se toman de una señal analógica durante un instante de tiempo para generar una señal de audio digital. Esta tasa se mide en Hertzios (Hz). Por ejemplo: Para una señal con una frecuencia de muestreo de *44100* Hz nos indica que cada segundo contiene *44100* muestras [58].

- *Resolución (bit resolution)*: Es el número de bits utilizados para almacenar cada muestra procedente de la señal analógica. Una resolución de *8-bits* proporciona *256* (2^8) niveles de amplitud, mientras que una resolución de *16-bits* alcanza *65536* (2^{16}). Un audio digital tendrá más calidad cuanto mayor sea su resolución [58].

- *Decibelio del nivel de presión sonora*: El *decibelio*, *decibel* o *dB* se utiliza para medir la intensidad del sonido y otras magnitudes físicas. *El decibelio del nivel de presión sonora (dB SPL)* toma como referencia el menor nivel de presión sonora que el oído humano medio puede detectar. En la práctica, a *dB* a menudo se le da el significado de dB SPL. La escala de decibelios es logarítmica, por lo que un aumento de tres decibelios en el nivel de sonido ya representa una duplicación de la intensidad del ruido [59]. El silencio o ausencia de sonido se corresponde a *0* dB (dB SPL) y el umbral del dolor para el oído humano se sitúa en torno a los *130-140* dB (dB SPL) [58].

- *Canales de audio*: Es el número de pistas que componen un archivo de audio. Si consta de un solo canal decimos que es *mono*, mientras que si son dos canales se llama *estéreo*. Si el archivo de audio posee dos o más canales también se denomina *multicanal* [60].

- *Velocidad de transmisión (bitrate)*: El *bitrate* define la cantidad de espacio físico (en bits) que ocupa un segundo de duración de un archivo de audio. Un archivo de audio tendrá más calidad cuanto mayor sea su bitrate pero como consecuencia, también ocupará más espacio (en bits) en la unidad de almacenamiento [58].

- *CBR/VBR*: Son las siglas de "*Constant/Variable Bitrate*". *CBR* indica que el audio ha sido codificado manteniendo el bitrate constante a lo largo del clip de audio mientras que *VBR* varía entre un rango máximo y mínimo en función de una tasa de transferencia [58].

- *Códec*: Acrónimo de "codificación/decodificación". Un *códec* es un algoritmo especial que reduce el número de bytes que ocupa un archivo de audio. Los archivos codificados con un codec específico requieren el mismo códec para ser decodificados y reproducidos. Uno de los más utilizados es el MP3 [58].

2.4.1 Pre-procesamiento de archivos de audio

A la hora de diseñar un modelo basado en la detección de eventos acústicos existen varios procesos que se tienen llevar a cabo para cumplir este propósito. Uno de estos procesos consiste en el pre-procesamiento de los archivos de audio que se van a utilizar donde se organiza y procesa este conjunto de archivos, transformando los datos de las señales de audio para adaptarlos a la entrada de un sistema tal y como muestra la Figura 2.21 en el caso de trabajar con una ANN.

Si queremos trabajar con una ANN, dentro del pre-procesamiento se realiza la tarea de adaptar una base de datos en *subconjuntos de entrenamiento, validación (o prueba) y evaluación* si la base de datos no contiene esta distribución, creando estos subconjuntos con una distribución adecuada de datos pudiendo ser por ejemplo, de *60 %* de los datos de cada clase de evento sonoro destinados al *subconjunto de entrenamiento* y del *20 %* para los *subconjuntos de validación y evaluación*. La agrupación del subconjunto de entrenamiento y validación a menudo se denomina *subconjunto de desarrollo*.

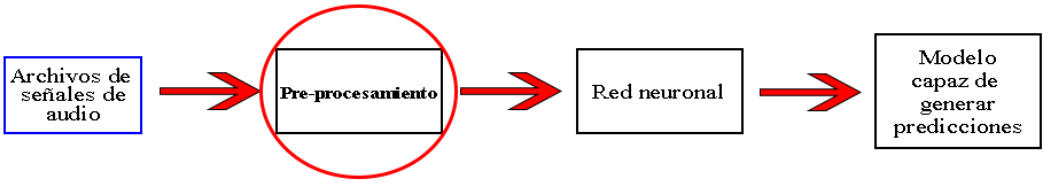


Figura 2.21: Representación de los bloques de trabajo para generar un modelo orientado a la detección de eventos acústicos utilizando una ANN.

El subconjunto de entrenamiento se compone de los datos que utiliza la ANN para entrenarse. Una vez que se hayan fijado los parámetros de la red y se haya generado un modelo durante una interacción del entrenamiento, se utilizarán los datos de validación para compararlos con las predicciones que realiza la red sobre estos y evaluar así cómo de acertado se encuentra el modelo, con el objetivo de mejorar el ajuste de parámetros en la siguiente interacción de entrenamiento y generar otro modelo (repitiendo este ciclo hasta que se detenga el entrenamiento).

Cuando el entrenamiento finaliza, el sistema selecciona y exporta el modelo más "óptimo" que ha generado durante el entrenamiento, y es en este punto donde se utilizan los datos de evaluación para determinar "la calidad final" de este modelo al realizar predicciones de datos que no ha utilizado antes.

En muchas bases de datos públicas orientadas para el entrenamiento de una ANN se pueden dar estas distribuciones, mientras que en otras se dan un conjunto de archivos y hay que realizar esta distribución manualmente.

Para poder trabajar correctamente con los archivos de una base de datos es importante que se presente una coherencia entre el procesamiento de cada archivo. Para una base de datos de archivos de audio, esto quiere decir que cada archivo se encuentre procesado mediante el mismo códec, que posean la misma tasa de muestreo, resolución (bit resolution), el mismo número de canales y se encuentren normalizados en amplitud entre otras cosas. Esto es muy importante para que un sistema se entrene correctamente.

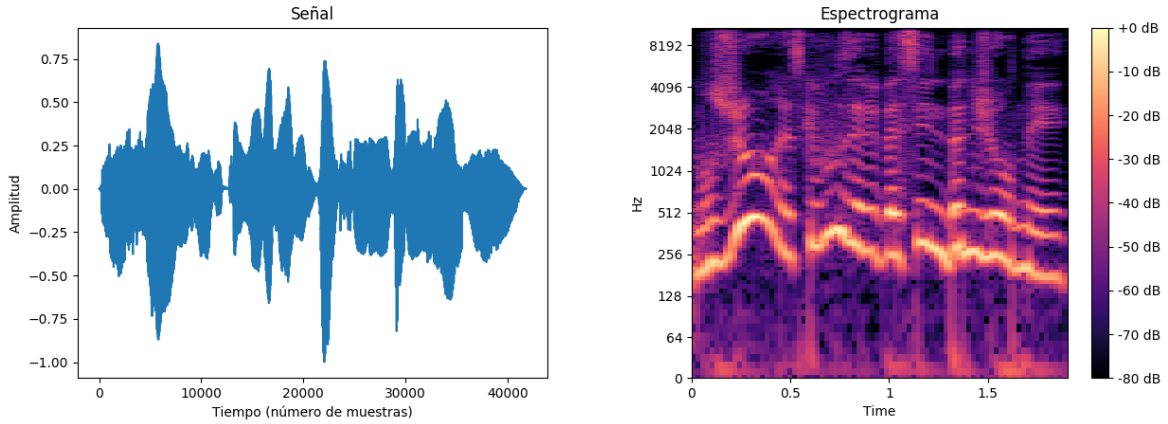
Por lo tanto, es necesario realizar un pre-procesado de cada archivo de audio para dar coherencia y adaptar cada archivo a la entrada de un sistema determinado con el objetivo de extraer la información más relevante que permita resolver el problema en cuestión, en este caso, clasificar un conjunto de señales de audio en un determinado número de clases. Por ello, la extracción de características es una parte fundamental dentro de cualquier sistema de reconocimiento de patrones [61].

En este apartado nos centraremos más en específico en el caso del pre-procesamiento de señales de audio para aplicarlas a la entrada de una CNN o CRNN, ya que se ha demostrado que son las redes más eficientes a la hora de tratar la detección de eventos sonoros.

2.4.2 Pre-procesamiento de señales de audio para adaptarlas a una red neuronal convolucional (CNN)

La finalidad de una CNN que se encargue de una tarea de detección y clasificación no es otra que la de una ANN convencional, pero su estructura de capas puede llegar a ser relativamente diferente. En este apartado centraremos nuestra atención en el tipo de datos que hay suministrar como entrada a una CNN en un problema de detección de eventos acústicos.

El pre-procesamiento de señales de audio para adaptarlas a una CNN consiste en toda la serie de procesos que se realizan sobre dichas señales para transformar en matrices de datos que contengan características que permitan extraer patrones. En este apartado vamos a describir tres tipos de extracciones de



(a) Representación en el dominio del tiempo de una señal de audio

(b) Representación de un espectrograma de una señal de audio

Figura 2.22: Representación de una señal de audio que contiene un grupo de personas hablando

características: *"Espectrogramas, espectrogramas de coeficientes cepstrales de frecuencia Mel (MFCCs) y los espectrogramas de Mel"*.

2.4.2.1 Espectrograma

Un espectrograma consiste en la representación gráfica del espectro de frecuencias de una señal [62]. Se realiza seleccionando un determinado número de muestras de una señal de audio por medio de una ventana temporal de un tamaño concreto. Esta ventana permite realizar una () de las muestras [63].

Seguidamente se desplaza la ventana a lo largo del tiempo de la señal, para seleccionar otro número de muestras diferentes. Se vuelve a calcular la STFT. Esta operación se repite sucesivamente a lo largo de la señal. La suma de la representación de las STFTs de las ventanas consecutivas es la que aporta la información en el dominio frecuencial de la señal, la variación de la energía y la frecuencia en función del tiempo. Normalmente, tras este proceso se aplica una operación logarítmica para apreciar mejor las variaciones de energía producidas [63].

Normalmente, un espectrograma representa el tiempo sobre el eje horizontal, la frecuencia sobre el eje vertical y la representación de la intensidad de las magnitudes de las STFTs mediante una escala de grises o de colores tal y como se puede observar en la Figura [62].

En la Figura 2.22 se puede apreciar un ejemplo de como se podría visualizar una señal de audio sobre un grupo de personas hablando en el dominio del tiempo (Figura2.22a) y su representación mediante un espectrograma (Figura2.22b).

2.4.2.2 Coeficientes cepstrales de frecuencia Mel (MFCCs)

Antes de entrar en detalles sobre los coeficientes cepstrales de frecuencia Mel es importante mencionar en que consiste la *escala de Mel*, que se basa en la simulación de la percepción psicoacústica de un oído humano frente a los tonos puros de las distinta frecuencia. Esta escala es lineal por debajo de *1* kHz y logarítmica por encima de este umbral como se observar en la Figura 2.23 [64].

Para conseguir esta transformación se divide la escala frecuencial de Hz. en contenedores y se transforma cada contenedor en otro correspondiente para la *escala de Mel*, utilizando un banco de filtros triangulares superpuestos como los que se pueden ver en la Figura 2.24 [65].

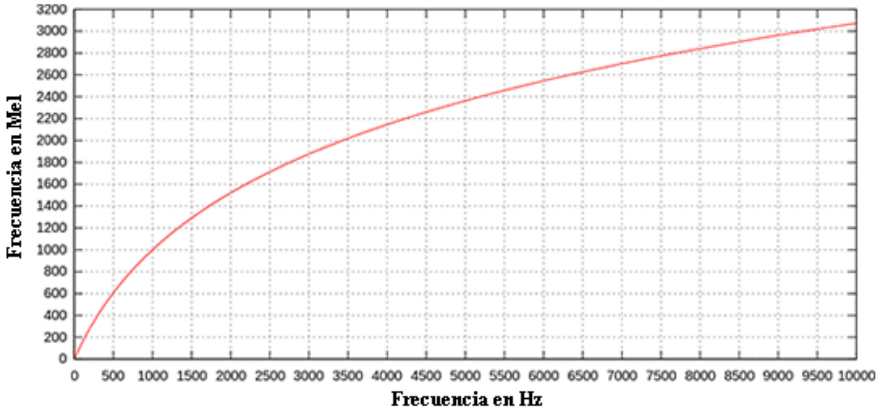


Figura 2.23: Representación de la relación entre las *frecuencias de la escala de Hz. y la escala de Mel*

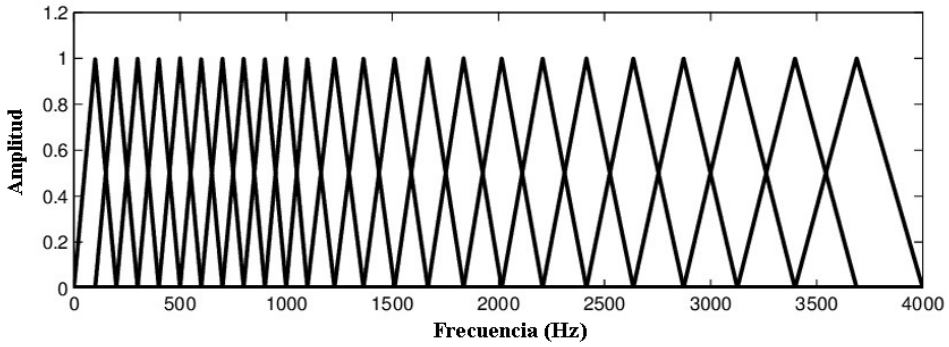


Figura 2.24: Representación de los bancos de filtros triangulares en la escala de Mel

Los coeficientes cepstrales de frecuencia Mel o *"Mel Frequency Cepstrum Coefficients (MFCCs)"* [66] son coeficientes utilizados para la representación del habla basados en la percepción auditiva humana que surgen de la necesidad de extraer características de las componentes de una señal de audio, obviando todas aquellas que posean información poco valiosa como el ruido de fondo, emociones, volumen, tono, etc. [67].

Para calcular los vectores de características de los MFCCs se aplica un filtro de pre-énfasis a la señal de audio, posteriormente se divide en tramas y se le aplica una función de enventanado, utilizando típicamente una ventana Hamming o de Hanning con el objetivo de eliminar los bordes de la señal y darle más relevanciaa la parte central de la trama (lo que mejora la representación frecuencial). A continuación se aplica a cada trama una transformada discreta de Fourier o *Discrete Fourier Transform* (DFT) para obtener su contenido espectral. Esta información es pasada a la *escala de Mel* mediante un banco de filtros y posteriormente se calcula la energía para cada uno de los filtros [64].

Finalmente se aplica la transformada discreta del coseno o *Discrete cosine transform* (DCT), obteniendo los MFCCs de cada trama. Estos valores serían los que se consideran como MFCCs, aunque opcionalmente se pueden añadir otros coeficientes conocidos como coeficientes *delta* y/o coeficientes *delta-delta*, que miden la variación de los MFCCs en una ventana alrededor de la que se está procesando [64].

Los coeficientes *delta* son denominados como "coeficientes de velocidad", puesto que representan la variación de los coeficientes MFCCs sobre un instante de tiempo. Por otro lado, los coeficientes *delta-delta* son conocidos como "coeficientes de aceleración", ya que representan la variación de los coeficientes *delta* alrededor de un instante de tiempo [68].

Los MFCCs se pueden representar como imágenes, pudiéndose suministrar como datos de entrada para técnicas de aprendizaje profundo como las CNNs, por ejemplo para la detección y clasificación de distintas clases de eventos sonoros. La representación de los MFCCs de la señal de audio de la Figura 2.22a usa un banco de *40* filtros triangulares superpuestos, que se visualiza como se muestra en la Figura 2.25.

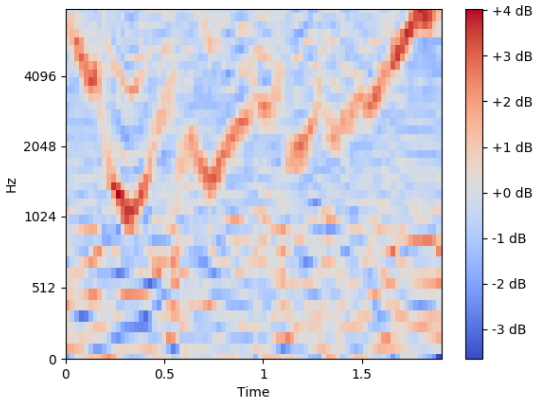
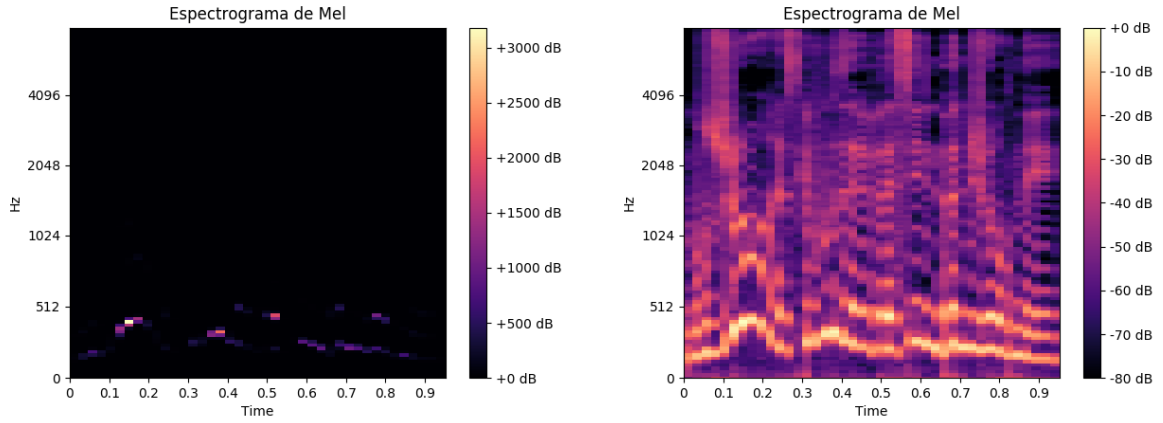


Figura 2.25: Representación de los MFCCs de una señal de audio que contiene un grupo de personas hablando

2.4.2.3 Espectrogramas de Mel

Dado que un *espectrograma* es una representación de tiempo frente a frecuencia de una señal de audio, un *espectrograma de Mel* consiste en una representación de tiempo frente a frecuencias en *escala de Mel* de una señal de audio, en donde las diferentes características muestran diferentes patrones en el espectro energético.



(a) Representación del *espectrograma de Mel* de una señal de audio

(b) Representación del *espectrograma de log-Mel* de una señal de audio

Figura 2.26: Representación de una señal de audio que contiene un grupo de personas hablando

El *espectrograma de Mel* se realiza de la siguiente forma: se aplica un filtro de pre-énfasis a una señal de audio, se divide la señal en tramas aplicándole un enventanado a cada una, después se le realiza una *DFT* a cada trama y se aplica un banco de filtros triangulares superpuestos para transformar las potencias espectrales procedentes de las *DFTs* de las diferentes tramas a la *escala de Mel*. Si se representara esta matriz resultante como si fuera una imagen, el resultado obtenido es lo que se conoce como *espectrograma de Mel*. Sin embargo, a menudo, un *espectrograma Mel* se escala logarítmicamente aplicando el logaritmo natural a cada valor de la matriz resultante, conociéndose así como *espectrograma de log-Mel*, ya que la forma logarítmica del *espectrograma de Mel* suele ayudar a identificar mejor las características de las señales de una forma más similar al oído humano. Por lo tanto, el *espectrograma de log-mel* corresponde al tiempo frente a la representación de frecuencia log-mel. A la hora de usar sistemas de aprendizaje profundo que disponen de una gran cantidad de datos y clasificadores sólidos como las CNNs, utilizar *espectrogramas de Mel* (o *espectrogramas de log-Mel*) puede funcionar mejor que los MFCCs [69].

En la Figura 2.26 se puede apreciar la representación del *espectrograma de Mel* y el *espectrograma de log-Mel* de la señal de audio correspondiente a la Figura 2.22a.

2.5 Python

Para la realización de este trabajo se ha utilizado el lenguaje de programación"*Python*" [70] por ser el lenguaje de programación más utilizado en la mayoría de los sistemas SED recopilados en el capitulo 3.

Su definición formal es la siguiente: "*Python es un lenguaje de alto nivel, interpretado, de propósito general, dinámicamente tipado, fuertemente tipado, de código libre, multiparadigma y multiplataforma*" [71].

2.5.1 Librerías importantes de Python

2.5.1.1 TensorFlow

Tensorflow [72] es la librería de *Python* más extendida orientada al cálculo numérico y el aprendizaje automático a gran escala. Además de para *Python*, se encuentra disponible para múltiples lenguajes de

programación y está preparado para poder ejecutarse sobre una gran variedad de dispositivos y arquitecturas, como pueden ser: "CPUs", "GPUs", o "clústeres de servidores". Esta librería es de código abierto y es desarrollada por *Google*.

Tensorflow usa como modelo de programación *grafos de flujos de datos*, en donde los *nodos* en un grafo representan operaciones matemáticas mientras que las *conexiones* o "*links*" del grafo representan los conjuntos de datos multidimensionales llamados *tensores*. Entre otras operaciones, se pueden construir y entrenar redes neuronales (ANNs) [73].

Tensorflow puede calcular automáticamente los gradientes que se necesitan para optimizar las variables del grafo a fin de que el modelo funcione mejor. Esto se debe a que el grafo es una combinación de expresiones matemáticas simples, por lo que el gradiente de todo el grafo se puede calcular utilizando la regla de cadena para el cálculo de las derivadas al optimizar la función de coste.

2.5.1.2 Keras

Keras [74] es una librería de redes neuronales artificiales de código abierto que se encuentra desarrollada en Python y puede ejecutarse sobre diferentes plataformas como *Tensorflow*. *Keras* está diseñado para construir mediante bloques la arquitectura de cada red neuronal, permitiendo construir redes complejas como las CNNs y CRNNs, además de otros bloques "más tradicionales" y entrenar modelos *deep learning* [75].

2.5.1.3 Sklearn

Sklearn [76] también conocido como *Scikit-Learn* es una librería de Python que integra una amplia gama de algoritmos de aprendizaje automático de última generación para afrontar problemas de aprendizaje supervisados y no supervisados de mediana escala.Esta librería hace hincapié en la facilidad de su uso, su buen rendimiento, su documentación y su coherencia.

2.5.1.4 Numpy

Numpy [77] es una librería de Python que proporciona un objeto de matriz multidimensional, varios objetos derivados como matrices enmascaradas y una variedad de rutinas para operaciones rápidas en matrices. *Numpy* te permite llevar a cabo operaciones matemáticas, álgebra lineal, operaciones estadísticas, simulaciones,...

Una creciente cantidad de librerías científico y matemáticas basadas en Python utilizan matrices NumPy. Por lo que para usar de manera eficiente gran parte (quizás incluso la mayoría) del software científico/matemático basado en Python conviene saber cómo usar las matrices NumPy.

2.5.1.5 Matplotlib

Matplotlib [78] es una librería para hacer gráficos de matrices en Python. Aunque tiene su origen en la emulación de los comandos gráficos de MATLAB, es independiente de MATLAB. Aunque *Matplotlib* está escrito principalmente en Python, hace un uso intensivo de *NumPy* y otros códigos de extensión para proporcionar un buen rendimiento incluso para matrices grandes. Esta biblioteca está diseñada con la filosofía de que se puedan crear gráficos simples utilizando pocos comandos.

2.5.1.6 Librosa

Librosa [79] es una librería de Python destinada al análisis de audio y música. Esta librería posee una gran cantidad de funcionalidades a la hora de trabajar con archivos de audio. *Librosa* se suele utilizar como punto de partida para trabajar con datos de audio en una amplia gama de aplicaciones como pueden ser tareas de reconocimiento de voz, encontrar características personales de un audio, clasificaciones de eventos sonoros, etc. Para los sistemas de detección y clasificación de eventos sonoros en concreto, es muy típico utilizar esta librería durante la etapa de pre-procesamiento de audio.

Capítulo 3

Sistemas y bases de datos disponibles

"La verdadera sabiduría está en reconocer la propia ignorancia."

Sócrates

3.1 Introducción

Para la realización de este trabajo se requirió de un periodo de recolección de información previo en el cual, uno de los objetivos consistía en encontrar el mayor número posible de sistemas eficientes relacionados con el reconocimiento y clasificación de diferentes eventos sonoros.

En este proceso se identificaron *14* sistemas que, en un principio, se consideraron válidos para someterlos a un análisis más profundo y seleccionar cuál de ellos se tomaría como referencia para probar y evaluar su rendimiento. En la sección 3.2 se realiza una descripción de cada sistema prestando gran atención en sus propósitos o aplicaciones, arquitectura, bases de datos con las que se ha probado, etc.

Otro de los objetivos fue la recolección del mayor número posible de bases de datos de eventos sonoros posibles Para este propósito se seleccionaron aquellas que tuvieran cierta diversidad de clases de eventos sonoros con una cantidad suficiente de datos de audio para cada clase (a ser posible: sonidos cotidianos, urbanos o relacionados con la seguridad) o en su defecto, que tuvieran un menor número de clases pero una gran cantidad de datos de audio

La calidad a la hora de seleccionar las bases de datos es una tarea importante pues limitan las modificaciones que se realizan en el sistema de referencia. Dicho sistema de referencia es entrenado previamente con una base de datos con unas características determinadas para posteriormente evaluar su rendimiento en otra base de datos diferente. Ambos conjuntos de datos tienen unas características similares y ambos poseen al menos, algunas clases en común. La descripción de las bases de datos empleadas se encuentra desarrollado en la sección 3.3.

3.2 Sistemas disponibles

3.2.1 Sistema 1: Acoustic event classification system using convolutional neural networks

Sistema diseñado por Stefan Kahl, Hussein Hussein, Etienne Fabian, Jan Schloßhauer, Danny Kowerko y Maximilian Eibl, descrito en [2] y propuesto para el taller INFORMATIK 2017 WS34 [80], cuyo objetivo es la creación de redes de aprendizaje profundo (Deep Learning) aplicadas en el área del habla y reconocimiento de voz.

Como datos de entrada se utilizan espectrogramas que se realizan a partir de las distintas señales de audio pertenecientes a cada evento acústico de la base de datos. Dichos espectrogramas son tratados posteriormente como imágenes con una resolución de *512x256* píxeles, *ancho x alto* donde el ancho se corresponde con la escala del tiempo y el alto con la escala de la frecuencia. Cada espectrograma representa fragmentos de *3* segundos de una señal de audio.

En un principio, para entrenar este sistema se utilizó un conjunto de datos propio que contiene *20* eventos sonoros diferentes que fueron recopilados en dos sesiones de grabación, denominadas como TUC e Intenta. En este conjunto de datos se combinaron sonidos humanos y ambientales, los cuales son producidos por personas en situaciones críticas como llamadas de auxilio, caídas al suelo, etc. Sin embargo, el sistema puede utilizarse para entrenar un modelo basado en su propio conjunto de datos o cualquier otro conjunto de datos disponibles públicamente. Como arquitectura se utiliza una CNN de ocho capas, lo que supone que las entradas de la red son específicamente imágenes a partir de los espectrogramas realizados.

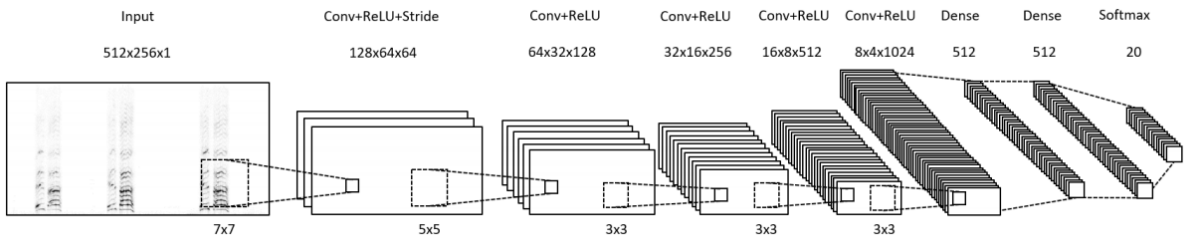


Figura 3.1: Representación de la arquitectura del Sistema 1. Reproducido de [2].

Como se puede ver en la Figura 3.1,la arquitectura se encuentra formada por una capa de entrada, varios bloques convolucionales, varias FC y una capa de activación softmax a la salida que permite obtener las probabilidades de cada una de las clases analizadas. En cuanto a los bloques convolucionales, estos están formados por una capa convolucional, una capa de *batch normalization* y una activación ReLU. En ellos se emplean *strides* de 2x2 en el primer bloque convolucional para hacer frente a grandes entradas y un *MaxPooling* de tamaño *2* para reducir las dimensiones espaciales después de cada convolución.

Todo el código del sistema se encuentra implementado en Python usando las librerías de NumPy [77], Theano [81] y Lasagne [82] para la generación de modelos, OpenCV [83] para el procesamiento de las imágenes y scikit-learn para el cálculo de métricas [76] y Matplotlib [78] para las visualizaciones.

3.2.2 Sistema 2: Acoustic event detection system using recurrent neural networks

Sistema diseñado por Kasey Ann, descrito en [84]. Su objetivo es crear un sistema de clasificación de eventos acústicos suficientemente válido para utilizarlo como referencia para una propuesta futura. Consiste en realizar buenas predicciones para eventos acústicos relacionados con maquinaria defectuosa.

Este sistema está enfocando en utilizar el aprendizaje automático para la clasificación de eventos acústicos utilizando un tipo específico de RNN denominada LSTM, descrita en la sección 2.3.2.

Para el pre-procesamiento de datos de audio, estos se remuestrean a *22000* Hz y se calculan sus MFCCs utilizando la biblioteca de Pyhton librosa [79].

A la hora de realizar el entrenamiento del sistema, se emplearon como datos de entrenamiento y validación, los archivos de audio pertenecientes a la carpeta *1* y *2* del conjunto de datos `Urbansound8k` [85], detallados en la sección 3.3.6) además de *10000* muestras de audios extraídas del sitio web `Freesound` [86] sobre sonidos de fondo urbano.

Este sistema se diseñó utilizando dos redes LSTM y una capa densa Para este sistema se realizaron varias pruebas modificando varios parámetros del sistema, pero los mejores resultados se consiguieron utilizando la función de activación Tanh durante el entrenamiento y un valor de *dropout* de *0.5*.

La arquitectura emplea Python ML TensorFlow [72] junto con la API de red neuronal de alto nivel Keras [74] ejecutándose sobre él. Para trazar datos se utilizan Matplotlib [78], pandas [87] y seaborn [88]. Para operaciones matriciales y vectoriales optimizadas se utiliza NumPy [77], para obtener métricas se utiliza scikit-learn [76] y para el procesamiento de audio se usa librosa [79].

3.2.3 Sistema 3: Sound event localization, detection, and tracking of multiple overlapping stationary and moving sources using convolutional recurrent neural network (SELDnet)

Sistema diseñado por Sharath Adavanne, Archontis Politis, Joonas Nikunen y Tuomas Virtanen, descrito en [89], que consiste en una extensión del trabajo descrito en [3] y [90].

Este sistema consiste en una red neuronal recurrente convolucional diseñada para la localización y detección de eventos sonoros de múltiples fuentes de sonidos superpuestos en el espacio tridimensional.

La entrada a la red consta de archivos de audio multicanal procesados, de los cuales se extraen los componentes de fase y magnitud para utilizarlos como características independientes. Se toma una secuencia de fotogramas de espectrogramas consecutivos de los diferentes archivos de audio y se predicen todas las clases de eventos que se encuentran activos para cada uno de estos fotogramas junto con su ubicación espacial. De esta forma, se predice la actividad temporal y la trayectoria de la dirección de llegada ("dirección de llegada o *Direction Of Arrival* (DOA)") para cada clase de evento acústico. Para mapear la secuencia de tramas en dos salidas en paralelo se utiliza una red neuronal recurrente convolucional ("CRNN") formada por una CNN seguida de una RNN. La primera salida consiste en una rama de detección de eventos sonoros, también conocidas como "SED", donde se realiza una tarea de clasificación de múltiples clases haciendo uso de múltiples etiquetas. Esto permite a la red estimar simultáneamente la presencia de múltiples eventos acústicos para cada espectrograma representando una pequeña instancia de tiempo. La segunda salida se corresponde a la localización de las ubicaciones espaciales de los eventos sonoros activos. En ella se utilizan las estimaciones DOA para cada clase de evento sonoro catalogados como activos.

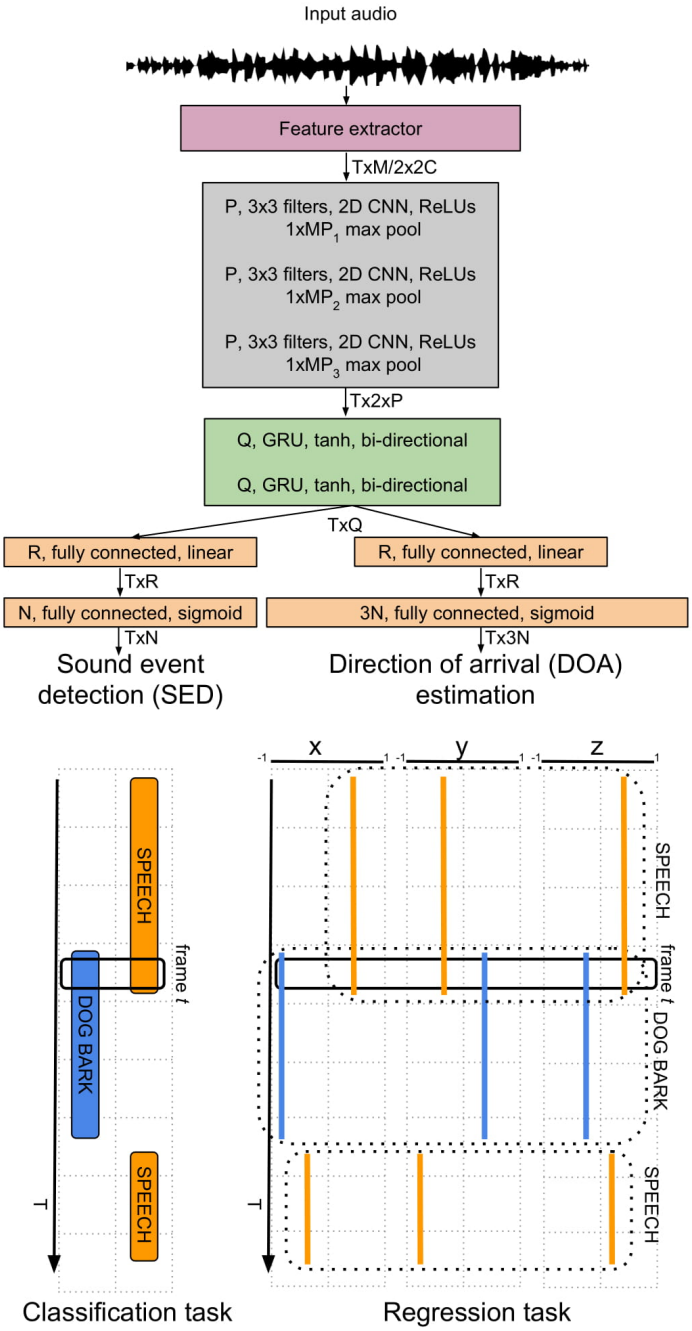


Figura 3.2: Representación de la arquitectura del Sistema 3. Reproducido de [3].

Como se puede observar en la Figura 3.2, la arquitectura de la red está compuesta por varios bloques convolucionales. que permiten a la CNN aprender las características relevantes requeridas para la localización La salida de los bloques convolucionales se redimensiona para transmitirla a las capas de RNN bidireccionales utilizados para aprender la información del contexto temporal. Específicamente, se utilizan varias capas GRU que hacen uso de activaciones Tanh.

A estas capas le siguen dos ramas de capas FC en paralelo, una para cada estimación de SED y DOA. La última capa FC en la rama SED consta de los N nodos que utilizan la activación sigmoide. Cada nodo se corresponde a una de las N clases de eventos sonoros que se pretende detectar. El uso de la activación sigmoide permite identificar múltiples clases que estén activas simultáneamente. La última capa FC en la rama DOA consta de un total de $3N$ nodos que utilizan una activación Tanh, donde cada una de las N clases de eventos sonoro está representada por 3 nodos correspondientes a la ubicación del evento en los ejes espaciales x, y, y z, respectivamente.

Esta arquitectura denominada `SELDnet` proporciona dos salidas, una correspondiente a la rama SED y otra perteneciente a la rama DOA. Dichas salidas se encuentran en el rango continuo de 0 a 1 para cada clase en el caso de la primera y en el rango continuo de -1 a 1 para la segunda. Cuando ambas salidas están disponibles se informa de qué evento sonoro se encuentra activo para un instante de tiempo y se elige su respectiva estimación de DOA si la salida de SED excede un umbral de 0.5.

Este sistema se propuso como parte de uno de los *challenge* de investigación en el famoso *workshop* de detección y clasificación de escenas y eventos acústicos detección y clasificación de escenas y eventos acústicos o *Detection and Classification of Acoustic Scenes and Events* (DCASE) de IEEE AASP [91]. Fue presentado como sistema de referencia para la tarea 3 en los *challenge* DCASE 2019 [92] y DCASE 2020 [93], las cuales tienen como objetivo la localización y detección de eventos sonoros.

3.2.4 Sistema 4: Two-stage sound event localization and detection using log mel space intensity vector and generalized cross-correlation

Sistema diseñado por Yin Cao, Turab Iqbal, Qiuqiang Kong, Miguel B. Galindo, Wenwu Wang y Mark D. Plumbley, descrito en [4] y propuesto para el *workshop* de la tarea 3 del *challenge* DCASE 2019 [92] de IEEE AASP [91]. Su objetivo es diseñar sistemas que sean capaces de localizar y reconocer de forma conjunta una colección de eventos sonoros individuales y sus respectivos tiempos de inicio y desplazamiento temporales.

La tarea 3 del *challenge* DCASE 2019 [92] de IEEE AASP [91] proporciona dos conjuntos de datos, "TAU Spatial Sound Events 2019 – Ambisonic" o "TAU Spatial Sound Events 2019 - Microphone Array", disponibles en [94] [95]. Los participantes pueden elegir uno o ambos conjuntos de datos Ambas colecciones de datos consisten en un conjunto de entrenamiento y evaluación. La eficacia de cada sistema presentado se evalúa mediante métricas individuales para la estimación de SED y DOA.

Para la SED, se utiliza la media armónica de la precisión (F) y la tasa de error (ER) calculada en segmentos de un segundo para todos los datos de evaluación en función del número total de falsos positivos, falsos negativos y verdaderos positivos. De este modo, para un método SED ideal se tendría una `ER` de 0 y `F` de 0.

Para la estimación de DOA utilizamos dos métricas de ventanas temporales: error de DOA y recuperación de la ventana temporal. El error DOA es el error angular promedio en grados entre los DOA predichos y de referencia mientras que la métrica de recuperación informa sobre el número de ventanas temporales en las que los DOA estimados y de referencia son desiguales.

Esta tarea presenta el sistema `SELDnet` [89], descrito en 3.2.3, como sistema base. Desde él se puede partir y realizar modificaciones con el objetivo de conseguir mejores resultados. Para este sistema se procesan los archivos de audio obteniendo vectores de intensidad es la escala Mel para la etapa SED y vectores de característica de correlación cruzada generalizada con transformación de fase (denominados vectores "*GCC-PHAT*" por sus siglas en inglés: "*correlación cruzada generalizada o* Generalized Cross Correlation *(GCC) transformada de fase o* Phase Transform *(PHAT)*") para la etapa de estimación DOA.

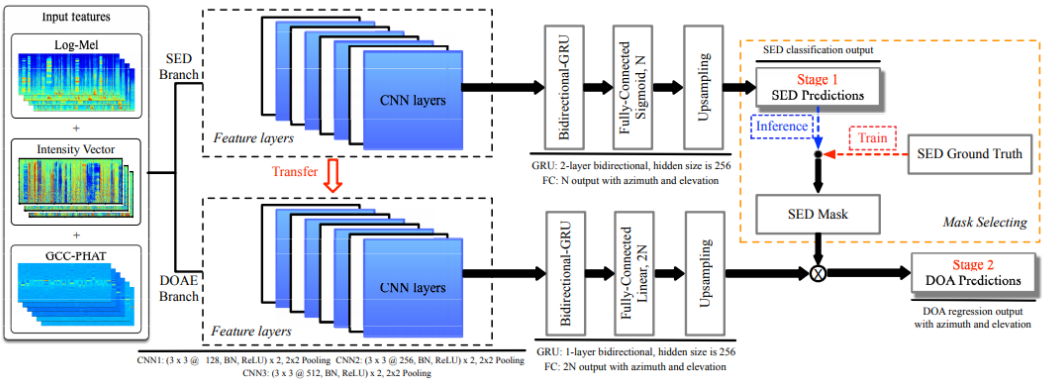


Figura 3.3: Representación de la arquitectura del Sistema 4. Reproducido de [4].

Tal y como muestra la Figura 3.3, durante el entrenamiento, las características extraídas se envían primero a la rama SED. Al igual que la arquitectura `SELDnet`, este sistema utiliza bloques convolucionales a la entrada de la red formados por una capa convolucional, una capa de normalización por lotes, una activación de ReLU y una agrupación promedio. Cada capa convolucional consta de dos conversiones 2D, operaciones de convolución con una *matriz de Kernel* de *3x3*, un *stride* de *1x1* y un tamaño de relleno (*padding*) de *1x1*.

Después de los bloques convolucionales, los datos se envían a una capa de agrupación de promedios globales para reducir la dimensión seguida de dos capas que utilizan unidades GRU bidireccionales. El tamaño de salida se mantiene y su salida se envía a una capa FC con un tamaño de salida *N*, que se corresponde con el número de clases de eventos. La función de activación sigmoide se utiliza posteriormente con un muestreo ascendente en la dimensión temporal para asegurar que el tamaño de salida sea consistente. A continuación se pueden obtener las predicciones de SED a través de un umbral de activación. La entropía cruzada binaria se utiliza para esta tarea de clasificación de etiquetas múltiples.

A continuación, se entrena la rama de estimación de DOA. Los bloques convolucionales se transfieren directamente desde la rama SED y se ajustan con precisión. La salida de la capa totalmente conectada para la rama de estimación de DOA es un vector de valores lineales *Nx2*, que representan los ángulos de acimut y elevación para los *N* eventos. Luego, son enmascarados durante el entrenamiento para determinar si los ángulos correspondientes están actualmente activos. Finalmente, se elige el error absoluto medio como pérdida de regresión de estimación de DOA. De esta forma, la rama SED calculará primero las predicciones SED, que luego se utilizan para obtener la estimación DOA.

3.2.5 Sistema 5: A neural network that enables high-throughput, automated annotation of birdsong (TweetyNet)

Sistema diseñado por Yarden Cohen, David Nicholson, Alexa Sanchioni, Emily K. Mallaber, Viktoriya Skidanova y Timothy J. Gardner, descrito en [5]. Este sistema apodado "`TweetyNet`" consiste en una ANN que permite la anotación automatizada de alto rendimiento del canto de los pájaros.

`TweetyNet` [5] consiste en el primer algoritmo automatizado para la anotación del canto de los pájaros aplicable a un canto complejo como el canto de un canario mediante una arquitectura de red neuronal. Dicha red se entrena con una pequeña cantidad de datos etiquetados a mano utilizando métodos de aprendizaje supervisados. Los datos utilizados para formar la base de datos consta de audios de aves provenientes de una recopilación de varias bases de datos públicas y de grabaciones realizadas con el propósito de utilizarlas para el desarrollo del sistema.

Este sistema implementa una CRNN y por tanto, se encuentra formado por dos tipos de capas: *1) capas convolucionales*, comunes en tareas de visión por computadora para aprender características de imágenes *2) capas recurrentes*, a menudo utilizadas para predecir secuencias. Utilizando como capas recurrentes, *capas recurrentes bidireccionales* LSTM con la intención de capturar las regularidades que poseen las vocalizaciones de los pájaros cantores en múltiples escalas temporales.

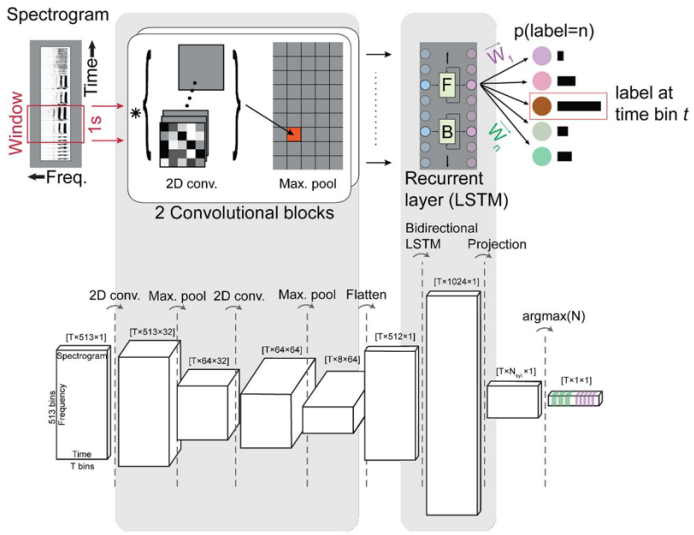


Figura 3.4: Representación de la arquitectura del Sistema 5. Reproducido de [5].

`TweetyNet` toma como entrada una ventana dividida en intervalos de tiempo de un espectrograma que se transmite a la entrada del sistema. A través de en una secuencia de pasos se genera una etiqueta para cada intervalo de tiempo dentro de la ventana.

Como se puede ver en la Figura 3.4, a la entrada de este sistema se encuentran dos bloques convolucionales donde se realizan convoluciones de la matriz de entrada con una matriz de kernel de tamaño *5x5* con *padding* ajustable para conservar las mismas dimensiones respecto a la entrada. A continuación se aplica una función de activación ReLU y posteriormente un filtro *MaxPooling* como método de *subsampling* para reducir las dimensiones. Para terminar con esta parte del sistema, al final del último bloque convolucional se adapta la salida y se transmite a una capa recurrente formada por unidades de memoria a corto plazo LSTM. En esta capa el número de neuronas es igual al número de intervalos de tiempo en la ventana del espectrograma. La salida de esta capa LSTM se transmite a una capa FC con el mismo número de neuronas que tipos de eventos sonoros sea capaz de distinguir el sistema. Finalmente, mediante una activación lineal y una capa de filtro *argmax*, se obtendrá el evento activo con mayor probabilidad en cada instante de tiempo.

3.2.6 Sistema 6: Sound classification system using Deep Learning

Sistema diseñado por Mike Smales, descrito en [96]. Este sistema es el resultado del "Proyecto Capstone de Nanodegree para Ingeniero de Aprendizaje Automático de Udacity" [97], el cuál consiste en diseñar un sistema que sea capaz de clasificar sonidos urbanos mediante aprendizaje profundo.

Para lograr este propósito se optó por utilizar una CNN con una arquitectura que consta de cuatro *bloques convolucionales* formados por una capa convolucional con una *matriz de kernel* de dimensiones *2x2* que aplica una activación ReLU, un filtro *MaxPooling* de tamaño *2x2* y un *dropout* con valor de *0.2*. La salida se obtendrá mediante una capa FC con tantas neuronas como números de eventos sonoros a clasificar. Por último, mediante una activación *softmax* se determina el evento activo en cada instante de tiempo.

Como base de datos, este sistema utiliza para entrenar y evaluar el conjunto de `Urbansound8K` [85], mencionado en la sección 3.3.6. Para procesar los diferentes archivos de audios se usan MFCCs. En este caso se usa una escala de frecuencia espaciada cuasi-logarítmica para visualizar el espectro de frecuencias de cada muestra de audio y cómo varían durante un período de tiempo muy corto, mostrando una representación basada en la percepción auditiva humana.

3.2.7 Sistema 7: Sound Event Detection system with Depthwise Separable and Dilated Convolutions (DnD-SED)

Sistema diseñado por Konstantinos Drossos, Stylianos I. Mimilakis, Shayan Gharib, Yanxiong Li y Tuomas Virtanen, descrito en [98]. Consiste en un método para la sustitución de las CNNs 2D utilizadas como extractor de características por convoluciones separables en profundidad. Así, las RNNs son sustituidas por convoluciones dilatadas dando lugar a las conocidas CRNN utilizadas en la detección de eventos sonoros. El método se nombró como: "*Sound Event Detection with Depthwise Separable and Dilated Convolutions*", o resumiendo,"`DnD-SED`".

El propósito general de este sistema es abordar una serie de problemas que se presentan en las CRNNs: *1)* la gran cantidad de parámetros que usa el método SED a causa de cada canal de las CNNs, *2)* el tamaño de las matrices de peso de las RNNs, *3)* el tiempo de entrenamiento, *4)* dificultad en el flujo del gradiente.

Como conjunto de datos para la evaluación de su rendimiento, el sistema utilizó "`TUT-SED Synthetic 2016`" [99], detallado en la sección 3.3.3, que contiene *16* clases de eventos sonoros. Como entrada a la red se proporciona un vector de características de los archivos de audio. Donde inicialmente son procesados por unos bloques convolucionales separables en profundidad *2D* que se encuentran en cascada, denominados: "`DWS-CNN`", basados en el modelo `MobileNets` [100]. Cada bloque tiene una capa de convolución *2D* basada en una convolución separable en profundidad seguida de una activación ReLU, un proceso de normalización por lotes y un proceso de submuestreo de características.

La salida del último bloque `DWS-CNN` se da como entrada a un módulo de identificación de patrón temporal que consiste en una capa de convolución *2D* basada en las convolución dilatada, "`DIL-CNN`". Este bloque a su vez va seguido de una capa FC que actúa como clasificador, donde la salida de esta capa FC se corresponde con las predicciones que el sistema realiza para cada instante de cada clase en un rango de *0* a *1*. Posteriormente, se determina si un evento sonoro se encuentra activo usando un umbral de *0.5*.

Para validar su correcto funcionamiento se realizaron una serie de experimentos comparando los resultados de este sistema con una CRNN que utiliza el modelo descrito en [101],teniendo en cuenta la puntuación media (`F`), la tasa de error medio (`ER`) y la cantidad de parámetros utilizados. A través de estos

experimentos se pudo apreciar que efectivamente existía una disminución considerable en la complejidad computacional y un aumento simultáneo en el rendimiento en la SED, logrando una reducción en la cantidad de parámetros y del tiempo de entrenamiento, un aumento de la puntuación media (F) y una reducción de la tasa de error media (ER).

3.2.8 Sistema 8: Sound event detection system based on convolutional neural network

Sistema diseñado por Arseniy Gorin, Nurtas Makhazhanov y Nickolay Shmyrev, descrito en [102]. Propuesto para el *workshop* de la tarea 3 el en *challenge* DCASE 2016 [103] de IEEE AASP [91]. Su objetivo es diseñar sistemas que sean capaces de reconocer eventos acústicos en condiciones de fuentes múltiples.

El conjunto de datos utilizado para esta tarea es: "TUT Sound events 2016" [104] [105] [106]. Este conjunto se divide en un subjuntivos de entrenamiento y otro de evaluación. Contiene archivos de audio etiquetados de dos escenas acústicas: "Hogar (interior)" y "área residencial (exterior)", dentro de las cuales se pueden distinguir diferentes tipos de eventos sonoros.

En este sistema se utiliza una CNN para detectar y clasificar eventos polifónicos en un contexto temporal capacitada para trabajar con las grabaciones de las escenas acústicas de "hogar" y "áreas residenciales". El pre-procesamiento de datos correspondiente a la extracción de características consiste en la aplicación de bancos de filtros log Mel. Para ello, se muestrean los archivos de audios a *16* kHz. Luego, se extraen *60* características del banco de filtros en tramas de *25* ms con una superposición de *15* ms. Agregando la primera y segunda derivadas calculadas dentro de una ventana temporal de *9* ventanas de señal.

Respecto a la arquitectura, la CNN toma como entrada 60 cuadros de características del banco de *filtros log Mel* con derivadas a la que se le agrega ruido gausiano con una desviación estándar de *0.01* durante el entrenamiento.

El primer bloque de convolución consiste en una capa de convolución con *80* filtros, *kernel* de tamaño *6x6*, *stride* de *1x1*), *MaxPooling* de *4x3* y *stride* de *1x3*).

El segundo bloque de convolución está formado por la segunda capa de convolución que contiene *80* filtros, *matriz de kernel* de *1x3*, stride de *1x1*, *MaxPooling* de *1x3* y *stride* de *1x3*.

Tras el segundo bloque de convolución y previo a la salida, se encuentran dos capas FC con *1024* neuronas por capa. A la capa de salida se le aplica la función de activación *sigmoide* para extraer las probabilidades de cada evento. Si la clase se detecta dentro de una ventana en más de *20* fotogramas en un intervalo de *1* segundo, el evento se considera como detectado.

Para todas las capas se utilizan funciones de activación ReLU. Para reducir el sobreajuste se aplica un *Dropout* con una tasa de *0.2* a la primera capa de convolución y con una tasa de *0.5* a las dos capas FC. La regularización de peso L2 también se utiliza para todas las capas con una pequeña penalización de *0.0001*. La red se entrena utilizando el optimizador ADAM con una tasa de aprendizaje, *learning rate*, de *0,001* optimizando la entropía cruzada. El entrenamiento finaliza cuando la pérdida de validación no mejora durante *10* épocas.

Como resultados, este sistema logra una mejora promedio de la tasa de error relativo del *7.7 %* respecto al sistema de referencia de esta tarea, pero no puede detectar eventos cortos con datos de entrenamiento limitados.

3.2.9 Sistema 9: Acoustic Event Detection of General Sounds system

Sistema diseñado por Michael Peitler, propuesto como trabajo de fin de máster, descrito en [107]. Consiste en la implementación de un sistema de detección de eventos acústicos proponiendo dos topologías distintas, diferenciándose en sus clasificadores.

En este trabajo se introducen sistemas de detección de eventos acústicos basados en la teoría de la percepción auditiva humana y en la parte final se propone un sistema para detectar eventos relevantes para la seguridad en espacios públicos. En este proyecto se presenta una colección de bases de datos de las que se extraen las diferentes muestras a utilizar. El conjunto de características se extraen mediante MFCCs, *características MP7* y *características basadas en el operador de energía Teager* (para mas información consultar [107]).

La primera topología utiliza clasificadores basados en GMMs. Cada clase de evento y el sonido de fondo son modelados por un GMM. La clase más probable se predice a continuación con la ayuda de la estimación máxima.

La segunda topología desarrollada reemplaza los GMMs por las máquinas de soporte vectorial o *Support Vector Machines* (SVMs). Se usa una SVM para cada clase que se va a detectar, la cual utiliza los datos de la clase de evento a detectar como datos positivos y los datos de todos los demás eventos, así como los sonidos de fondo como datos negativos. Por lo tanto, cada clasificador SVM entrega independientemente un resultado binario que indica si el evento específico ha sido detectado o no.

El sistema se probó con cinco clases de eventos de audio que se clasifican y distinguen del sonido de fondo: `Grito, disparo, rotura de vidrio, explosión y voz humana`.

La evaluación de marco se realiza con medidas de clasificación adecuadas. La precisión, la recuperación, la exactitud y la puntuación F1 se calculan para cada clase. Estas medidas se promedian para todo el sistema y se calcula la tasa de error de eventos acústicos. Según los resultados obtenidos se tiene que reemplazar los GMM entrenados generativamente con SVM, obteniéndose mejores resultados de clasificación promedio.Debido a los recursos computacionales limitados, en este proyecto no se realizó ninguna optimización especial para mejorar el rendimiento con SVM, por lo que no se pudo encontrar el modelo óptimo.

3.2.10 Sistema 10: Large-scale weakly supervised audio classification system using gated convolutional neural network

Sistema diseñado por Qiuqiang Kong and Yong XU, Wenwu Wang y Mark D. Plumbley, descrito en [6] y propuesto para el *workshop* de la tarea 4 del *challenge* DCASE 2017 [108] de IEEE AASP [91]. Su objetivo es diseñar sistemas capaces de detectar eventos sonoros a gran escala utilizando datos de entrenamiento débilmente supervisados pensados para poder implantarse en automóviles inteligentes, ciudades inteligentes y áreas relacionadas.

El conjunto de datos a utilizar consiste en extractos de vídeos de *YouTube* que se centran en el transporte y las advertencias debido a sus aplicaciones industriales. Para este conjunto de datos, la clasificación de etiquetas han sido basadas en la plataforma `AudioSet Ontology` [109]: "un conjunto de datos de ontología y etiquetado humano para eventos de audio" de *Google*. El conjunto de datos seleccionado consta de *17* eventos de sonido divididos en dos categorías: "*Advertencia*" y "*vehículos*".

- *Sonidos de advertencia:* `Bocina de tren (441), bocina de aire o bocina de camión (407), alarma de coche (273), pitidos inversos (337), sirena de`

`ambulancia (624), sirena de coche de policía (2399), sirena de camión de bomberos (2399), sirena de defensa civil (1506) y gritos (744).`

- *Sonidos de vehículos:* `Bicicleta (2020), patinete (1617), coche (25744), coche pasando (3724), autobús (3745), camión (7090), motocicleta (3291) y tren (2301).`

En la lista anterior se muestra cada clase y junto a ella el número aproximado de clips de *10* segundos de duración, teniendo en cuenta que cada clip puede corresponder a más de un evento de sonido simultáneamente.

El conjunto de entrenamiento tiene etiquetas débiles que indican la presencia de un evento de sonido determinado en un extracto de un vídeo sin proporcionar marcas de tiempo. Para el conjunto de validación y evaluación, se proporcionan etiquetas sólidas con marcas de tiempo con el fin de evaluar el rendimiento.

Esta tarea se divide en dos subtareas, ya que la detección de eventos sonoros dentro de un clip de *10* segundos se debe realizar mediante dos niveles: "Sin marcas de tiempo (similar al etiquetado de audio)" y "con marcas de tiempo (hora de inicio y fin)".

La arquitectura de este sistema se puede apreciar en la Figura 3.5.

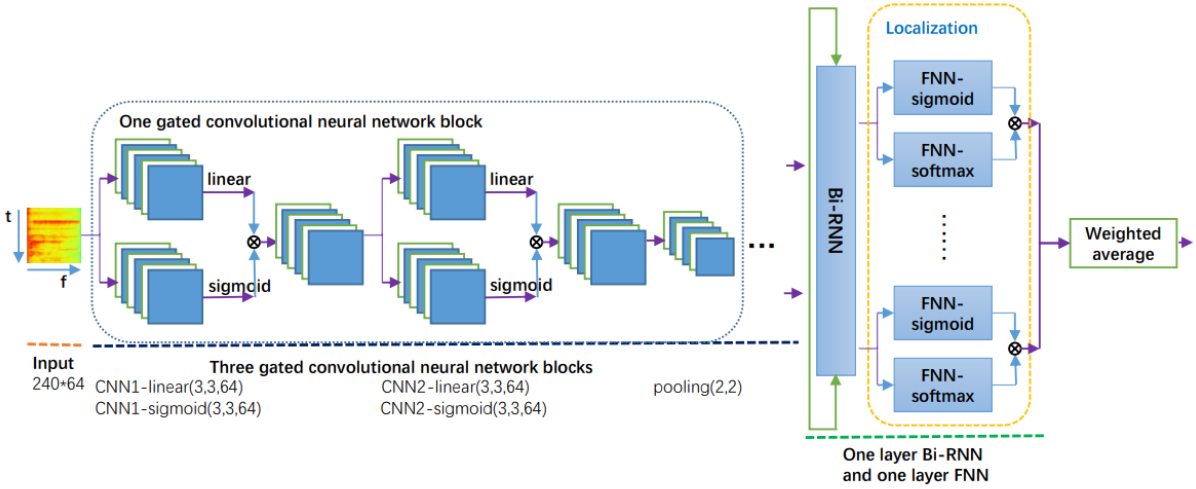


Figura 3.5: Representación de la arquitectura del Sistema 10. Reproducido de [6].

Para la subtarea de etiquetado de audio sin marcas de tiempo, se propuso un método basado en una CRNN para predecir las etiquetas de audio. Como pre-procesado de audio, se opta por utilizar la técnica de espectrogramas de log-Mel para conseguir representaciones de tiempo-frecuencia mediante tramas para todas las grabaciones de audio. Estos espectrogramas de log-Mel son suministrados como entrada de la red hacia las capas convolucionales para extraer características de alto nivel y posteriormente se adopta una capa bidireccional recurrente (Bi-RNN) para capturar la información del contexto temporal seguida de una capa de retroalimentación (FNN) Finalmente, la probabilidad de predicción de cada etiqueta de audio o clips que poseen una duración de *10* segundos, se obtiene promediando los valores de todos los fotogramas comprendidos en cada intervalo de *10* segundos.

Para la CRNN se utilizan además las unidades lineales con puertas o *Gated Linear Units (GLUs)* como funciones de activación en lugar de las funciones de activación de ReLU. Las GLUs pueden controlar la cantidad de información de los distintos espectrogramas que pasa de una capa a la siguiente. De esta manera, la red aprenderá a prestar atención a los eventos de audio e ignorar los sonidos no relacionados.

Este funcionamiento es similar a los mecanismos de activación en las memorias a corto plazo (LSTM) o unidades recurrentes controladas (GRU).

A diferencia de la subtarea de etiquetado de audio sin conocer las ubicaciones temporales de cada evento, la subtarea de detección con marcas de tiempo necesita predecir las ubicaciones temporales de cada evento de audio. Por este motivo, se introduce una capa de retroalimentación (FNN) adicional con *softmax* como función de activación para ayudar a inferir las ubicaciones temporales de cada clase que ocurre.

En la fase de entrenamiento se aplica la *función de pérdidas de entropía cruzada binaria* entre las probabilidades predichas y la etiqueta de las grabación de audio. Como optimizador se utiliza ADAM y los pesos de la red neuronal se actualizan mediante el gradiente de los pesos calculados a través de retropropagación o *back propagation*.

En cuanto a resultados, este sistema ocupó el primer lugar en la subtarea de etiquetado de audio (sin marcas de tiempo) y el segundo lugar en la subtarea de detección de eventos de sonido (con marcas de tiempo) en la tarea 4 del *challenge* DCASE 2017 [108] de IEEE AASP [91].

3.2.11 Sistema 11: Training general-purpose audio tagging networks with noisy labels and iterative self-verification

Sistema diseñado por Matthias Dorfer y Gerhard Widmer, descrito en [110] y propuesto para el *workshop* de la tarea 2 del *challenge* DCASE 2018 [111] de IEEE AASP [91]. En él se propone el diseño de sistemas capaces de realizar un etiquetado de audio de uso general con un gran número de categorías utilizando datos con anotaciones de confiabilidad variable.

El *challenge* se basa en la construcción de un sistema que sea capaz de asignar una de las *41* posibles etiquetas de clases de eventos sonoros para cada fragmento de una grabación de audio. Utilizando para ello, espectrogramas de los audios segmentados como entrada.

Esta tarea utiliza el conjunto de datos `FSDKaggle2018`, consta de muestras de audio extraídas de `Freesound` [86] y anotadas con un vocabulario de *41* etiquetas de diferentes eventos sonoros procedentes de la plataforma `AudioSet Ontology` [109]. Lo que hace que este conjunto de datos y la tarea sean especiales es el hecho de que solo *3700* de las *9500* etiquetas del conjunto total son verificadas manualmente. Las etiquetas restantes no están verificadas y contienen hasta un *30-35* % de etiquetas incorrectas en algunos tipos de eventos sonoros.

Para el pre-procesamiento de los datos de audio, las señales de audio se remuestrean a *32* KHz y se calcula una Transformada de Fourier de tiempo reducido (STFT) utilizando ventanas de Hanning de *1024* muestras para finalmente calcular sus *espectrogramas*.

Para la parte de aprendizaje de características se utiliza una red convolucional profunda para el reconocimiento de imágenes a gran escala. La arquitectura se basa en una red neuronal convolucional basada en el modelo [112], donde se utiliza una gran cantidad de capas convolucionales, *matrices de Kernel* de *3x3*, *stride* de *1x1*, función de activación PReLU, *padding* para mantenga la dimensión de la salida igual a la de la entrada y una capa de *Batch Normalization*.

La agrupación espacial se lleva a cabo mediante *5* capas de agrupación máxima (*MaxPooling*), que se realiza justo después de algunas capas convolucionales. Los filtros de *MaxPooling* tienen una dimensión de *2x2* con un *stride* de *2x2*. Además, en algunas capas se aplica un *Dropout* con valores de *0.3* y *0.5*. Al final de la red, tras las diferentes capas convolucionales se encuentra una capa FC y una activación *softmax* para obtener finalmente la salida de la red. Durante el entrenamiento se utiliza el optimizador ADAM

con una tasa de aprendizaje inicial *learning rate* de *0.001* y un tamaño de mini-lote de *100* muestras. Cada modelo se entrena para *500* épocas.

Como resultados se obtuvo que el modelo entrenado alcanzó una puntuación F superior a *0.8* para todas las clases a excepción de *3* clases, incluso muchas de las clases se obtuvieron una puntuación perfecta de *1*. Teniendo en cuenta las etiquetas ruidosas y que hay *41* clases diferentes para distinguir, este es un resultado notable.

3.2.12 Sistema 12: General-purpose audio tagging system from noisy labels using convolutional neural networks

Sistema diseñado por Turab Iqbal, Qiuqiang Kong, Mark D. Plumbley y Wenwu Wang, descrito en [113] y propuesto para el *workshop* de la tarea 2 del *challenge* DCASE 2018 [111] de IEEE AASP [91]. En él se propone el diseño de sistemas capaces de realizar un etiquetado de audio de uso general con un gran número de categorías utilizando datos con anotaciones de confiabilidad variable.

Como se ha comentado en el sistema *11* (descrito en 3.2.11), la tarea 2 de este *challenge* se basa en la construcción de un sistema que sea capaz de asignar una de las *41* posibles etiquetas de clases de eventos sonoros para cada fragmento de una grabación de audio. Realizando espectrogramas de los audios segmentados como entrada utilizando el conjunto de datos `FSDKaggle2018`.

Para el pre-procesamiento de las señales de audio de este sistema, el primer paso que se realiza es extraer los segmentos no silenciosos de las señales de audio. Para ello se segmenta la señal de audio en cuadros y se determina el umbral de la energía cuadrática media de cada cuadro.

Después de la eliminación del silencio, los segmentos extraídos se consideran entradas independientes. Estos segmentos se remuestrean a *32* kHz y se transforman en *espectrogramas de frecuencia mel en escala logarítmica (log-Mel)*, donde se utilizan *2* conjuntos de parámetros para extraer las características log-Mel: una "configuración estándar (configuración A)" y una "configuración de banda estrecha (configuración B)".

- En *la configuración A*, se realizan los espectrogramas de log-Mel utilizando un tamaño de ventana de *1024*, con un tamaño de salto de *512* y *64* bandas de Mel.
- En *la configuración B*, se realizan los espectrogramas de log-Mel utilizando un tamaño de ventana de *512*, con un tamaño de salto de *256* y *64* bandas de Mel.

Al utilizar estas configuraciones se producen resultados complementarios, pareciendo que las diferentes resoluciones capturan características diferentes.

Después de la extracción de características, cada vector de características se divide en fragmentos de un tamaño fijo, lo que resuelve el problema de la duración variable de los clips. Cuando la longitud del vector de características es menor que la longitud del fragmento, se rellena. Cuando es mayor, pero no uniformemente divisible por la longitud del fragmento, se agrega un fragmento adicional para alinearlo con el final del vector de características para que incluya el resto.

Se selecciona un tamaño de fragmentos de *128x64*, donde el primer eje es la dimensión temporal. Esto corresponde a fragmentos de *2* segundos y fragmentos de *1* segundo para las configuraciones *A* y *B*, respectivamente. Aproximadamente el *80 %* de los segmentos del conjunto de entrenamiento tienen más de *1* segundo de duración y el *60 %* son más de *2* segundos.

Para entrenar este sistema utilizando el conjunto de entrenamiento que se proporciona para esta tarea, se usan *2* tipos de redes neuronales: CNN y CRNN, donde en cada tipo, también se utilizan *2* variantes:

"una variante que usa convoluciones estándar y otra que usa convoluciones cerradas". Dado que durante la extracción de características hay *2* configuraciones para obtener los espectrogramas log-Mel, esto da un total de *8* modelos de entrenamiento en total.

Comenzando con la CNN estándar, equivalente a la red "`VGG13`" propuesta en [112] (utilizada también en el sistema anterior 3.2.11). Cada bloque convolucional consta de dos capas convolucionales seguidas de una capa de agrupación máxima que divide a la mitad cada dimensión espacial. Las capas convolucionales utilizan una función de activación ReLU, así como la normalización por lotes como forma de regularización. Después de los bloques convolucionales, se aplica la agrupación promedio global, es decir, cada mapa de características se promedia en ambas dimensiones. Finalmente, se utiliza una capa FC con una función de activación *softmax* para generar las predicciones.

La arquitectura CRNN es una extensión de la red `VGG13`. En lugar de aplicar la combinación de promedios globales después de las convoluciones, solo se promedia la dimensión de frecuencia para que se conserve la información temporal. A continuación, se aplica una capa bidireccional recurrente para aprender la dinámica temporal de la entrada, generando un vector para instante de tiempo. Se promedian estos vectores y se envían a la última capa, que consiste (al igual que la CNN estándar) en una capa FC con una función de activación *softmax*.

Las otras dos arquitecturas son redes *GCNN* y *GCRNN*, que son variantes de las redes `VGG13` y CRNN, respectivamente. La diferencia es que cada capa convolucional se reemplaza por una capa convolucional cerrada, descritas en [114]. La idea de una capa cerrada se inspira en los mecanismos de entrada que se encuentran en las RNNs y se utiliza para controlar la información que se propaga a capas más profundas.

Para afrontar entrenamientos con la presencia de etiquetas no verificadas, este sistema propone *2* técnicas:

- La primera técnica consiste en ponderar la función de pérdida de entrenamiento de modo que su magnitud se reduzca para ejemplos no verificados, ya que, si un ejemplo está etiquetado incorrectamente causaría que la pérdida calculada fuera también incorrecta y debería ignorarse. Al no saber si la etiqueta no es correcta porque la etiqueta no está verificada, se establece un parámetro en la ecuación de función de pérdidas que hace alusión a la "confianza" de que las etiquetas no verificadas sean correctas.

- La segunda técnica es hacer uso del pseudoetiquetado, que consiste en volver a etiquetar los ejemplos no verificados antes del entrenamiento utilizando un clasificador previamente entrenado. Para que el pseudoetiquetado sea eficaz, la tasa de error del clasificador debe ser menor que la tasa de ruido de las etiquetas originales, ya que la tasa de error se considera como la nueva tasa de ruido. En este caso, el clasificador previamente entrenado es el mismo sistema descrito pero sin utilizar pseudoetiquetado. Otra forma de pseudoetiquetado propuesta es "promover" ejemplos de no verificados a verificados al corroborar la etiqueta con las predicciones del clasificador previamente entrenado.

Puesto que se trabaja con un conjunto de datos pequeño y con el fin de evitar el sobreajuste del sistema se utilizó una técnica de aumento de datos conocida como Mixup [115]. Esta técnica se encarga de generar nuevos datos durante el entrenamiento mezclando aleatoriamente pares de entradas y sus valores objetivo asociados.

Para entrenar los distintos modelos, el conjunto de entrenamiento se dividió en cinco *folds* de validación cruzada, asegurando que hubiera un número similar de ejemplos verificados en cada *folds* y se utiliza la *función de entropía cruzada* como función pérdidas y el optimizador ADAM.

Todos los hiperparámetros se seleccionaron en función de las evaluaciones de un conjunto de validación que solo contenía etiquetas verificadas. Al generar las predicciones, las cuatro primeras épocas se seleccionan en función del rendimiento dado utilizando el conjunto de validación evaluándose con la métrica de precisión media.

Por otro lado, puesto que las entradas a las redes neuronales son fragmentos de secciones del clip de los audios originales del conjunto de datos, las predicciones de fragmentos se fusionan para producir predicciones a nivel de clip haciendo uso de la media geométrica, ya que es menos sensible a los valores atípicos que la media aritmética. Sin embargo, con las predicciones a nivel de clip, las cuatro primeras épocas se fusionan utilizando la media aritmética.

Para combinar las predicciones de los diferentes modelos, se utiliza un método conocido como apilamiento, donde las predicciones del modelo base se utilizan como características para entrenar un clasificador de segundo nivel, el cual utiliza la regresión logística con una penalización *L2* y usa las ponderaciones de clases para compensar el desequilibrio de clases y las ponderaciones de las muestras.

En cuanto a los resultados obtenidos, el conjunto de *8* sistemas apilados lograron el tercer lugar en la clasificación de la tarea 2 del *challenge* DCASE 2018 [111].

3.2.13 Sistema 13: Audio tagging system focusing on label noise, data augmentation and its efficient learning

Sistema diseñado por Il-Young Jeong y Hyungui Lim, descrito en [116] y propuesto para el *workshop* de la tarea 2 del *challenge* DCASE 2018[111] de IEEE AASP[91]. En él que se propone el diseño de sistemas capaces de realizar un etiquetado de audio de uso general con un gran número de categorías utilizando datos con anotaciones de confiabilidad variable.

Como se ha comentado en el sistema *11* (descrito en 3.2.11), la tarea 2 de este *challenge* se basa en la construcción de un sistema que sea capaz de asignar una de las *41* posibles etiquetas de clases de eventos sonoros para cada fragmento de una grabación de audio. Realizando espectrogramas de los audios segmentados como entrada utilizando el conjunto de datos `FSDKaggle2018`.

Para este sistema a la hora de realizar el pre-procesamiento de datos se aplicó un remuestreo de datos, donde se probó a remuestrear las señales de los archivos de audio a *16* kHz, *32* kHz y *44.1* kHz. Una frecuencia de muestreo baja puede perder información útil en alta frecuencia, pero será más fácil analizar un rango temporal más largo ya que el tamaño de los datos es menor. Además del remuestreo también se aplicó el método Mixup de aumento de datos conocido mencionado anteriormente.

En este sistema se probaron dos modelos diferentes de procesamiento de audio, que están "basado en log-Mel" y "basado en forma de onda". La arquitectura de este sistema está formado por *3* bloques: Un modulo de bajo nivel, un modulo DenseNet y un modulo de clasificación.

Para el modelo basado en log-Mel, el modulo de bajo nivel está formado por una normalización por lotes, una transformación al dominio log-Mel de las entradas, un redimensionamiento para que puedan asemejarse a una imagen en escala de grises, otra normalización por lotes y una capa convolucional de *Kernel 3x3* y un relleno o *padding* a la salida de la red.

Para el modelo basado en forma de ondas, el módulo de bajo nivel está formado por una normalización por lotes y una capa convolucional con *Kernel 1x3*, relleno o *padding* y una activación linear.

El módulo DenseNet están basados en la arquitectura DenseNet[117], que consiste en una red CRNN densamente conectada en la que cada capa recibe entradas adicionales de todas las capas anteriores y pasa sus propios mapas de características a todas las capas posteriores utilizando concatenación.

En este módulo para el caso del modelo basado en log-Mel, se realiza una normalización por lotes, una activación ReLU, una capa convolucional *Kernel* de dimensiones *1x1*, una capa de *padding* y una activación linear. Posteriormente le sigue otra normalización por lotes, otra capa convolucional con *Kernel* de *3x3*), una "*Squeeze and Excitation Network*"[118] y una agrupación máxima, *MaxPooling* con filtros de dimensiones *2x2*.

Para el caso del modelo basado en forma de onda, se modifica la dimensión de la *matriz de kernel* de *3x3* de la segunda capa convolucional por una *matriz de kernel* con dimensiones *1x3* y se reemplaza la agrupación máxima con filtros de dimensiones *2x2* por filtros de *1x2*.

En cuanto al clasificador está formado por *8* capas FC en paralelo con el mismo número de neuronas que clases de eventos detecte la red seguidas cada una de una activación *softmax*, donde la probabilidad de detección de cada evento resultará ser la media de las *8* probabilidades de las diferentes capas.

En este sistema se llevó a cabo el enmascaramiento de pérdidas mediante lotes, aprovechando que se conoce qué datos se encuentran etiquetados correctamente y cuáles no, utilizando esto para modificar la función de pérdida convencional para ignorar los datos ruidosos para cada mini-lote, consiguiendo con esto, eliminar los datos que poseen un mayor error en el cálculo del gradiente.

En cuanto a los resultados, para la tarea 2 del *challenge* DCASE 2018 [111] de IEEE AASP, se presentaron *3* resultados de predicciones utilizando diferentes combinaciones de los modelos descritos y de las frecuencias de muestreo a utilizar en cada caso.

- El primer conjunto de modelos se realiza utilizando todas las frecuencias de muestreo probadas (*16*, *32* y *44.1* kHz) y modelos (basados en log-Mel y formas de onda). Cada configuración se entrena mediante el uso de *5 folds* de validación cruzada *CV*, por lo que el número total de modelos para el conjunto es de: *3 x 2 x 5*, siendo frecuencias de muestreo, modelo y *CV* respectivamente. Esta configuración consiguió una puntuación de precisión media de *0.975*.
- El segundo conjunto de modelos utiliza *16* kHz y *32* kHz como las frecuencias de muestreo. El número total de modelos para el conjunto es de: *2 x 2 x 5*, siendo frecuencias de muestreo, modelo y *CV* respectivamente. Obteniendo también una puntuación de precisión media de *0.975*.
- El tercer conjunto de modelos sólo utiliza la frecuencia de muestreo de *32* kHz y un modelo basado en log-Mel, por lo que se utilizaron *5* modelos de validación cruzada *CV* para este conjunto y se obtuvo una puntuación de precisión media de *0.972*.

3.2.14 Sistema 14: Sound event detection system using spatial features and convolutional recurrent neural network

Sistema diseñado por Sharath Adavanne y Tuomas Virtanen, descrito en [7] y propuesto para el *workshop* de la tarea 3 del *challenge* DCASE 2017 [119] de IEEE AASP [91], cuyo objetivo es diseñar sistemas que sean capaces de reconocer eventos de sonido en condiciones de fuentes múltiples.

Esta tarea utiliza el conjunto de datos de `TUT Sound events 2017` [104] [120] [121], que se encuentra dividido en dos subconjuntos, entrenamiento y evaluación. Ambos subconjuntos contienen archivos de audio etiquetados de *6* clases de eventos sonoros (`chirrido al frenar` *`[brakes squeaking]`*`, coche` *`[car]`*`, niños` *`[children]`*`, camión` *`[large vehicle]`*`, gente hablando` *`[people speaking]`* `y gente caminando` *`[people walking]`*) pertenecientes a la escena acústica "calle (exterior)".

Como entrada al sistema se le suministra las características procesadas correspondientes a una señal de audio compuesta por la agrupación de varios archivos de audio, donde la señal de audio generada puede

contener un canal (señal mono) o dos canales (señal estéreo) de audio, extrayendo las características para cada canal de audio en ventanas de tiempo consecutivas. Estas características de audio son introducidas en una red neuronal convolucional recurrente multicanal, que asigna las características de audio a las etiquetas de eventos de sonido en el conjunto de datos, dando como salida de la red valores en el rango de *[0, 1]* para cada clase de evento sonoro utilizado, siendo "*1*" presencia de evento y "*0*" su ausencia. Dependiendo de si se trabaja con señales estéreos o monos, la dimensión de la entrada de la red serán diferente. Si a la hora de generar la entrada al sistema se trabaja con distintos archivos de audio con canales variables (algunos archivos monos y otros estéreos) existe la posibilidad de convertir todos los archivos a mono.

El pre-procesamiento de datos correspondiente a la extracción de características acústicas de los archivos de audio se puede dar mediante dos maneras distintas para entradas de audio binaural (estéreo) y una manera para las entradas de audio de un solo canal (mono), donde todas las características se extraen en una longitud de salto de *20* ms para mantener el mismo número de tramas y los diferentes archivos de audio se remuestrean con una misma frecuencia de muestreo (por defecto de *44.1* KHz).

Para la extracción de características enfocada a entradas de audio de un solo canal (mono) se utilizan espectrogramas de log-Mel, siendo esta técnica también conocida como "*log mel-band energy*" (*mbe*), la cual se ha utilizado ampliamente para las tarea de SED. Para este método, *mbe* utiliza una ventana de Hamming de *40* ms de longitud y *40* bandas Mel en el rango de frecuencia de *0-22500* Hz. Para una entrada de audio determinada con una cantidad "*F*" de fotogramas este bloque de extracción de características tiene como resultado una salida de dimensiones "*F x 40*".

En cuanto a la extracción de características enfocada a entradas de audio de un solo binaural (estéreo) se utiliza el método conocido como *binaural log mel-band energy* (*bin-mbe*), el cual extrae las características de forma similar a *mbe* en cada uno de los canales binaurales, lo que da como resultado una salida de dimensiones "*F x 80 (40 * 2)*".

Otra manera de realizar la extracción de características para entradas de audio binaural (estéreo) consiste en utilizar ventanas de resolución múltiple a la hora de aplicar la energía binaural log-Mel-band (*bin-mbe*), haciendo referencia a esta técnica como: "*bin-mul-mbe*". Específicamente, se usan tres tamaños de ventana diferentes (*1024*, *4096* y *16384*) y se extraen las característica *mbe* en cada una de las ventanas y en cada uno de los canales binaurales. Este bloque de extracción de características da como resultado una salida de dimensiones "*F x 240 (40 * 3 * 2)*".

Respecto a la arquitectura, tal y como se puede ver en la Figura 3.6, este sistema usa una CRNN que utiliza *3* bloques convolucionales inicialmente, *2* capas de unidades recurrentes con compuerta bidireccional (GRU) y *2* capas completamente conectadas (FC), siendo la última de estas la capa de salida de la red.

Los bloques convolucionales se utilizan para aprender patrones locales invariantes a partir de la función de audio. Cada bloque de convolución esta compuesto por una capa de convolución con una etapa de normalización por lotes (*Batch Normalization*), una *función de activación*, un *subsampling* mediante filtros de agrupación máxima (*MaxPooling*) y un *dropout*. Los *filtros receptivos (matrices de kernel)* de las capas de convolución son de dimensiones *3x3* con un *stride* de *1x1* y un *padding* que rellena la salida generada con ceros para mantener la dimensión de la salida igual a la de la entrada. Después de la capa de convolución se aplica una normalización por lotes, *Batch Normalization*, seguida de la aplicación de una función de activación ReLU y una agrupación máxima, *MaxPooling*, con el objetivo de reducir la dimensión final a *T x 2 x N*, donde *N* es el número de filtros en la capa final de convolución. La agrupación máxima, *MaxPooling*, se realiza sólo en el eje de frecuencia, haciendo esto para preservar la resolución de

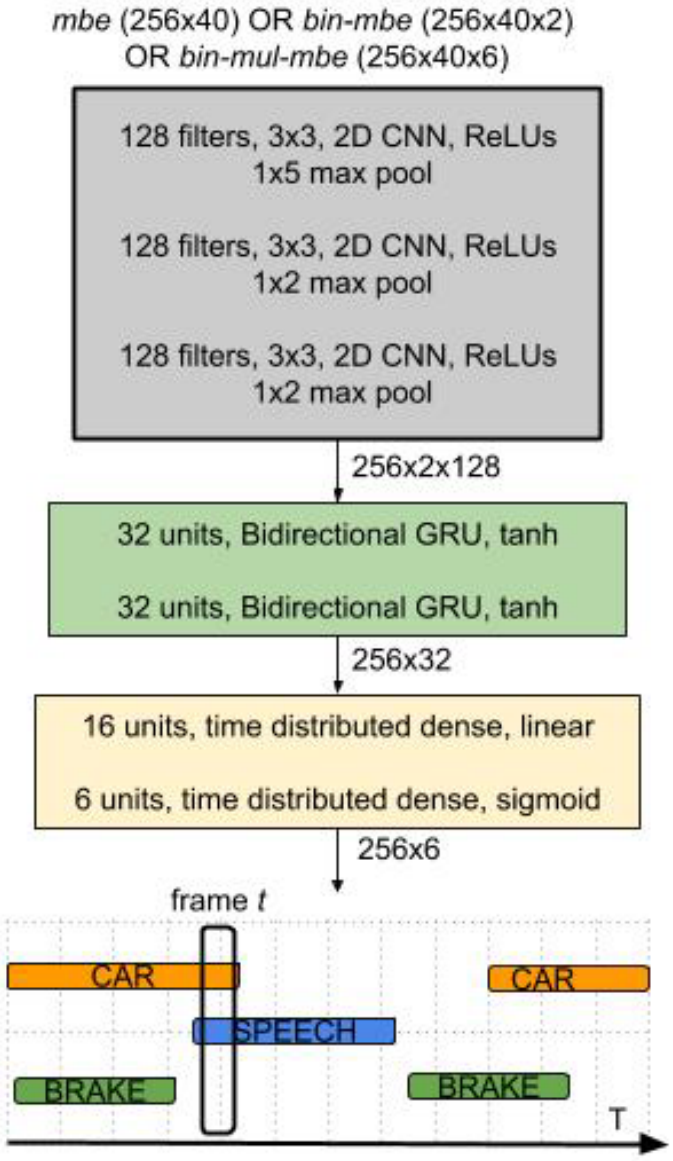


Figura 3.6: Representación de la arquitectura del sistema de referencia utilizado en el capitulo de *Implementación*. Reproducido de [7].

tiempo de la entrada. Seguido a la agrupación máxima de cada bloque se utiliza un *dropout* de valor *0.5* como regularizador.

Tras los bloques convolucionales se encuentran *2* capas GRU de *32* neuronas cada una, cuyo propósito es el de poder aprender patrones de actividad temporal a largo plazo. A continuación de las capas GRUs se encuentran *2* capas FC con *32* y C neuronas (siendo "C" el número total de clases de eventos sonoros). La segunda capa FC actúa como una capa de predicción que posee una función de activación *sigmoide* para poder producir una salida por cada clase de evento sonoro.

El entrenamiento de la red se realiza durante *500* épocas utilizando la *función de pérdida de entropía cruzada binaria* y el optimizador ADAM con una tasa de aprendizaje de *0.0001*. Se emplea la parada anticipada o *learning rate* para detener el ajuste excesivo de la red a los datos de entrenamiento, deteniendo el proceso antes de las *500* épocas si la tasa de error, *ER* evaluada sobre el conjunto de prueba no mejora durante*125* épocas.

Durante cada época del entrenamiento, el modelo generado se evalúa comparando las predicciones realizadas sobre el conjunto de validación frente a su etiquetado real. En este sistema, al realizar predicciones sobre un conjunto de datos, los valores de las probabilidades de detección de cada clase de evento sonoro se encuentran en un rango de entre *0* y *1* para cada instante puesto que se utiliza una función de activación *sigmoide* en la última capa de la red, tal y como se ha mencionado anteriormente.

Para poder clasificar qué eventos se consideran activos en las predicciones realizadas, se utiliza un umbral de *0.5*.

Las métricas que se utilizan durante el entrenamiento son: *la tasa de error por segmento (*ER*)* y *la puntuación media (*F*) o "F-score"* calculada en segmentos de un segundo de duración utilizando micropromediado.

La puntuación media (F) se calcula mediante la siguiente fórmula:

$$F = \frac{2 \cdot \sum_{k=1}^{K} TP(k)}{2 \cdot \sum_{k=1}^{K} TP(k) + FP(k) + \sum_{k=1}^{K} FN(k)'} \tag{3.1}$$

Donde:

- K representa el número de segmentos totales de un segundo.
- *TP(k)* representa los verdaderos positivos para cada segmento de segundo, que se corresponde con el número de etiquetas de eventos sonoros activos tanto en las predicciones como en las etiquetas verdaderas.
- *FP(k)* representa los falsos positivos para cada segmento de segundo, que se corresponde con el número de etiquetas de eventos sonoros activos en las predicciones pero inactivas en las etiquetas verdaderas.
- *FN(k)* representa los falsos negativos para cada segmento de segundo, que se corresponde con el número de etiquetas de eventos sonoros activos en las etiquetas verdaderas pero inactivas en las predicciones.

La ER se mide como:

$$ER = \frac{\sum_{k=1}^{K} S(k) + \sum_{k=1}^{K} D(k) + \sum_{k=1}^{K} I(k)}{\sum_{k=1}^{K} N(k)} \tag{3.2}$$

Donde:

- K representa el número de segmentos totales de un segundo.
- *N(k)* es el número total de eventos de sonido activos en las etiquetas verdaderas del segmento k.
- *S(k)* representa las sustituciones para cada uno de los K segmentos de un segundo que se calculan mediante: $S(k) = min(FN(k), FP(k))$
- *D(k)* representa las eliminaciones para cada uno de los K segmentos de un segundo que se calculan mediante: $D(k) = max(0, FN(k) - FP(k))$
- *I(k)* representa las inserciones para cada uno de los K segmentos de un segundo que se calculan mediante: $I(k) = max(0, FP(k) - FN(k))$

Respecto a la implementación de la red neuronal se realizó utilizando *Keras* [74] con *Theano* [81].

En cuanto a los resultados para la tarea 3 del *challenge* DCASE 2017[119] de IEEE AASP, los diferentes sistemas presentados utilizando los métodos de pre-procesado de audio mencionados anteriormente ocuparon los primeros lugares en la clasificación.

Finalmente, este sistema fue seleccionado como el sistema de referencia a utilizar para el capítulo de implementación. desarrollado en la sección 4) de este trabajo, donde se trabaja con este sistema probando su funcionamiento y posteriormente evaluando su rendimiento realizando ciertas modificaciones.

3.3 Base de datos disponibles

3.3.1 ESC-50: Dataset for Environmental Sound Classification

El conjunto de datos `ESC-50` [122] es una colección de datos de audios ambientales de *2000* grabaciones adecuadas para métodos de evaluación comparativa. Son extraídos de una base de datos denominada `ESC` [123] que consta de una abundante compilación unificada de *250000* extractos auditivos sin etiquetar extraídos de grabaciones disponibles a través de la página web de `Freesound` [86]. La razón de no haber seleccionado el conjunto `ESC` al completo viene dada porque a la hora de recopilar bases de datos para este trabajo, solo interesan las que contuvieran archivos de audio etiquetados que tuvieran la posibilidad de utilizarse en una ANN de detección y clasificación de eventos sonoros. El conjunto de datos consta de grabaciones de *5* segundos de duración, muestreadas a *44.1* kHz, utilizan un canal mono. Se encuentran organizadas en *50* clases semánticas, con *40* muestras por clase, agrupadas en *5* categorías principales:

- *Animales*: `Perro, gallo, cerdo, vaca, rana, gato, gallina, insectos (voladores), oveja y cuervos.`
- *Paisajes sonoros naturales y sonidos de agua*: `Lluvia, olas del mar, fuego crepitante, grillos, pájaros piando, gotas de agua, viento, echar agua, gotear el agua del baño y tormenta.`
- *Sonidos humanos que no son del habla*: `Bebé llorando, estornudos, aplausos, respiración, tos, pasos, risas, cepillar los dientes, ronquidos y beberse una bebida.`
- *Sonidos interiores o domésticos*: `Golpe de puerta, click del ratón, teclado escribiendo, crujidos de madera de puerta, abrir un refresco (lata), lavadora, aspiradora, reloj despertador, tic del reloj y rotura de cristales.`

- *Ruidos exteriores o urbanos*: Helicóptero, motosierra, sirena, bocina, motor, entrenar, campanas de iglesia, avión y fuegos artificiales.

3.3.2 IEEE DCASE 2016 Challenge: Task 2 Datasets

Este conjunto de datos se corresponde con el proporcionado para el *workshop* de la tarea 2 del *challenge* DCASE 2012[124] de IEEE AASP[91], en el que se propone el diseño de sistemas capaces de detectar eventos de sonidos de oficina superpuestos en mezclas sintéticas.

Este conjunto de datos consiste en archivos de audio con su etiquetado correspondiente, que contienen eventos de sonido aislados para cada una de las clases de eventos sonoros disponibles. Además contiene mezclas sintéticas de los mismos en múltiples condiciones de *SNR* y densidad de eventos.

El material se grabó en un ambiente tranquilo, utilizando el micrófono de cañón *AT8035* conectado a una grabadora *ZOOM H4n*. Los archivos de audio se muestrean a *44.1* kHz y son monofónicos. Los parámetros que controlan el material sintetizado incluyen la presencia/ausencia de eventos superpuestos, el número de eventos por clase y la relación evento-fondo *EBR* con valores de *-6*, *0* y *6* dB. Todos los archivos de audio se encuentran anotados con la hora de inicio, la hora de finalización y la etiqueta de los eventos sonoros activos.

Este conjunto se encuentra dividido en dos conjuntos de datos: un conjunto de desarrollo [125] y un conjunto de evaluación [126].

A su vez, el conjunto de desarrollo se encuentra dividido en un subconjunto de datos de entrenamiento y otro de validación. El subconjunto de entrenamiento consta de *20* archivos de audio aislados, mientras que el conjunto de validación consta de *18* grabaciones de *2* minutos en diversas condiciones.

El conjunto de evaluación consta de varias mezclas sintéticas de archivos de audio de los diferentes clases de eventos en varios niveles de *SNR*.

Las *11* clases de eventos sonoros que contiene este conjunto son: Aclararse la garganta, tosido, golpear una puerta (llamar), portazo, cajón, gente riéndose, teclear, llaves (poner sobre la mesa), pasar de página, teléfono sonando y gente hablando.

3.3.3 TUT-SED Synthetic 2016: Synthetic dataset for sound event detection research

TUT-SED Synthetic 2016 [99] es una base datos de audio que contiene señales de mezcla generadas artificialmente a partir de muestras de eventos de sonido aislados. Para realizar este conjunto de datos se obtuvieron *994* muestras de eventos de sonido de la web "Sound Ideas" [127]. Las señales de mezcla se crean seleccionando y mezclando aleatoriamente eventos de sonido aislados de *16* clases con una polifonía máxima de cinco eventos en cada instante. De las *16* clases se seleccionaron segmentos con una duración de entre *3* y *15* segundos.

Los segmentos procesados se utilizan para sintetizar *100* mezclas polifónicas, destinando el 60 %para datos de entrenamiento, 20 % para validación y 20 % para evaluación. La cantidad total de material de audio en el conjunto de datos es de *566* minutos.

Las *16* clases de eventos sonoros que contiene este conjunto son: Alarmas y sirenas, llanto de bebé, canto de pájaros, autobús, gato maullando, aplausos de la multitud, multitud aclamando, perro ladrando, pasos, aplastar vidrio, arma de fuego, paseo a caballo, mezclador, motocicleta, lluvia y trueno.

3.3.4 TUT Sound events 2016

El conjunto de datos `TUT Sound events 2016` [104] [105] [106] se utilizó para el *workshop* de la tarea 3 del *challenge* DCASE 2016 [103] de IEEE AASP [91], cuyo objetivo es diseñar sistemas que sean capaces de reconocer eventos acústicos en condiciones de fuentes múltiples. El conjunto de datos "`TUT Sound events 2016`" fue creado a partir de un subconjunto de `TUT Acoustic scene 2016` [128] [129].

`TUT Sound events 2016` consta de grabaciones capturadas cada una en un lugar diferente de dos escenas acústicas: "hogar (interior) y zona residencial (exterior)". Para cada ubicación se capturó una grabación de audio de *3-5* minutos de duración. El equipo utilizado para la grabación consta de un micrófono interno binaural Soundman OKM II Klassik / studio A3 electret y una grabadora de ondas Roland Edirol R-09 que utiliza una frecuencia de muestreo de *44.1* kHz y una resolución de *24* bits.

Los eventos de sonido en cada grabación fueron anotados por dos asistentes de investigación utilizando etiquetas de sonido elegidas libremente, garantizando que los sonidos seleccionados sean comunes para una escena acústica y que haya suficientes ejemplos para aprender modelos acústicos.

Clases de eventos de sonido seleccionados finalmente fueron las siguientes:

- *Casa*: `Crujido, chasquido, alacena, cuchillería, platos, cajón, tintineo de vidrio, impacto de objeto, gente caminando, lavado de platos, grifo de agua corriendo.`
- *Barrio residencial*: `Golpe, canto de los pájaros, coche pasando, niños gritando, gente hablando, gente caminando y viento que sopla.`

El conjunto de datos de TUT Sound events 2016 consta de dos subconjuntos: "*conjunto de datos de desarrollo* [105] y *conjunto de datos de evaluación* [106]". La partición de datos en estos subconjuntos se realizó en función de la cantidad de ejemplos disponibles para cada clase y se tuvo en cuenta la ubicación de grabación. Idealmente, los subconjuntos deberían tener la misma cantidad de datos para cada clase.

El conjunto de entrenamiento [105] consta de *22* grabaciones de audio de dos escenas acústicas: *10* grabaciones con un total de *36.16* minutos de la escena acústica "hogar (interior)" y *12* grabaciones con un total de *42* minutos correspondientes a la escena acústica "zona residencial (exterior)".

El conjunto de evaluación [106] consta de *10* grabaciones de audio de dos escenas acústicas: *5* grabaciones con un total de *17.49* minutos de la escena acústica "hogar (interior)" y *5* grabaciones con un total de *17.49* minutos correspondientes a la escena acústica "zona residencial (exterior)".

3.3.5 TUT Sound events 2017

El conjunto de datos `TUT Sound events 2017` [104] [120] [121] se empleo en el *workshop* de la tarea 3 en el *challenge* DCASE 2017 [119] de IEEE AASP [91]. Su objetivo es diseñar sistemas que sean capaces de reconocer eventos de sonido en condiciones de fuentes múltiples. El audio del conjunto de datos "`TUT Sound events 2017`"es un subconjunto de `TUT Acoustic scene 2017` [130] [131].

El conjunto de datos de `TUT Sound events 2017` consta de grabaciones de escenas acústicas en la calle con varios niveles de tráfico y otras actividades. La escena fue seleccionada por representar un entorno de interés para la detección de eventos sonoros relacionados con actividades humanas y situaciones de peligro. Para cada ubicación de grabación, se capturó una grabación de audio de *3-5* minutos de duración. El equipo utilizado para la grabación consta de un micrófono interno binaural *Soundman OKM II Klassik/studio A3 electret* y una grabadora de ondas *Roland Edirol R-09* que utiliza una frecuencia de muestreo de *44.1* kHz y una resolución de *24* bits.

Las clases de eventos se seleccionaron para representar sonidos comunes relacionados con la presencia humana y el tráfico, seleccionando *6* clases de eventos: `Chirrido de un vehículo al frenar, coche, niños, camión (large vehicle), gente hablando y gente caminando.`

El conjunto de datos de `TUT Sound Events 2017` consta de dos subconjuntos: "*conjunto de datos de entrenamiento* [120] y *conjunto de datos de evaluación* [121]". La partición de datos en estos subconjuntos se realizó en función de la cantidad de ejemplos disponibles para cada clase de evento.

Emplea validación cruzada con cuatro *folds* y cada grabación se use exactamente una vez como datos de prueba. Al crear los *folds*, la única condición impuesta fue que el *subconjunto de prueba o validación* no contuviera clases no disponibles en el *subconjunto de entrenamiento*.

El conjunto de datos de entrenamiento [120] consta de *24* grabaciones de audio de la escena acústica: "calle (exterior)" con una duración total de *92.08* minutos.

El conjunto de datos de evaluación [121] consta de *8* grabaciones de audio de la escena acústica: "calle (exterior)" con una duración total de *29.09* minutos.

3.3.6 Urbansound8k

`Urbansound8k` [85] extraido de `Freesound` [86], es un conjunto de datos de audio recopilado con el objetivo de abordar dos de las barreras principales en clasificación automática de sonido urbano: "la falta de una taxonomía común" y "la escasez de grandes conjuntos de datos anotados". Este *set* contiene *8732* extractos de sonidos urbanos en formato *WAV*. Es un conjunto etiquetado, con un total de *10* clases y una duración máxima *4* segundos cada extracto. La frecuencia de muestreo, la profundidad de bits y el número de canales no se encuentran estandarizados para cada archivo. Los audios se encuentran repartidos en *10 folds* con el mismo número de archivos de cada clase en cada uno.

Las *10* clases de evento acústico que contiene este conjunto son: `Aire acondicionado, bocina de automóvil, niños jugando, ladrido de perro, taladrado, motor al ralentí, disparo de arma, martillo neumático, sirena y música callejera.`

3.3.7 FSDnoisy18k

`Freesound Dataset noisy18k` o `FSDnoisy18k` [132] es un conjunto de datos de audio recopilado de `Freesound` [86] con el objetivo de fomentar la investigación del ruido de etiquetas en la clasificación de eventos de sonido. Contiene *42.5* horas de audio en *20* clases de eventos sonoros, incluida una pequeña cantidad de datos etiquetados manualmente y otra cantidad mayor de datos con etiquetas ruidosos.

La parte etiquetada de manera manual consiste en clips de audio cuyas etiquetas se clasifican como presentes en el clip y predominantes, lo que significa que la etiqueta es correcta. La parte ruidosa de los datos consiste en clips de audio que no recibieron validación humana. Por lo tanto, la parte ruidosa presenta una cierta cantidad de ruido de etiqueta.

Las *20* clases de este conjunto son: `Guitarra acústica, bajo, aplausos, moneda (cayendo), crash de un platillo, platos, ollas y sartenes, motor, pedo, fuego, fuegos artificiales, vidrio, hi hat, piano, lluvia, golpe, chirrido, lagrimeo, pasos, viento y escritura.`

3.3.8 Marsyas

Marsyas [133] es una base de datos musical utilizada en [134]. Esta recopilación se realizó durante el *2000* y *2001* de una variedad de fuentes, incluidos CD personales, radio o grabaciones de micrófono para representar una variedad de condiciones de grabación. El conjunto de datos consta de *1000* pistas de audio de *30* segundos de duración cada una. Todos archivos se encuentran estandarizados como archivos monofónicos de *1650* bits, *22050* Hz y en formato *WAV*.

Este conjunto de datos contiene *10* géneros representado cada uno de ellos por *100* pistas: *Blues, Classical, Country, Disco, Hiphop, Jazz, Metal, Pop, Reggae y Rock*.

3.3.9 Clotho dataset

Clotho dataset [135] es un conjunto de datos de subtítulos de audio que consta de *4981* archivos, donde cada uno de ellos contiene cinco muestras. Todos los sonidos provienen de la plataforma Freesound [86] y los subtítulos son de colaboración colectiva mediante Amazon Mechanical Turk [136] y anotadores de países de habla inglesa. Palabras únicas, entidades nombradas, y la transcripción de voz se eliminan con pos-procesamiento. Los archivos de audio tienen una duración de *15* a *30* segundos y las muestras tienen una longitud de *8* a *20* palabras.

3.3.10 DBR dataset

DBR dataset es un conjunto de datos de audio ambiental creado para el seminario de licenciatura en procesamiento de señales en la Universidad Tecnológica de Tampere. Las muestras del conjunto de datos se obtuvieron de la plataforma Freesound [86].

El conjunto de datos consta de *3* clases, cada una con *50* muestras, donde las clases son: Perro, pájaro y lluvia. El nombre DBR, viene de sus siglas en inglés: Dog, Bird, Rain.

Los archivos de audio se encuentran en el formato *WAV* y se encuentran divididos en tres carpetas, una para cada clase. Para cada archivo se aporta además, un archivo en formato *yaml* de metadatos y un archivo de texto de anotación.

3.3.11 An open dataset for research on audio field recording archives (freefield1010)

Freefield1010 [137] es un conjunto de datos abierto y gratuito provienen de la plataforma Freesound [86]. Consta de *7690* clips de audio etiquetados con una duración mínima de *10*. El conjunto de datos está diseñado para su uso en investigaciones relacionadas con la minería de datos en grabaciones de campo o paisajes sonoros.

Las *7* clases de eventos sonoros que contiene este conjunto son: cantos de pájaros, sonidos urbanos, sonidos de gente, sonidos de la naturaleza, sonidos de trenes, voces y sonidos de agua.

3.3.12 Freesound Loops 4k (FSL4)

Freesound Loops 4k (FSL4) [138] es un conjunto de datos de audio que forma parte de la publicación [139]. Contiene aproximadamente *4000* archivos de audio provenientes de la plataforma

Freesound [86] mediante la búsqueda de las palabras *loop* y *bpm*. El conjunto de datos contiene una serie de archivos de audio sin procesar en diferentes formatos y un archivo con los metadatos de los audio.

3.3.13 Freesound One-Shot Percussive Sounds

El conjunto de datos Freesound One-Shot Percussive Sounds [140] contiene *10254* sonidos de percusión (evento único) extraídos de la plataforma Freesound [86]. Todas las muestras de audios son de *1* segundo y se encuentran muestreadas a *16* kHz.

Este conjunto de datos se utilizó para entrenar el sistema descrito en [141].

3.3.14 SimSceneTVB Learning

SimSceneTVB Learning [142] es un conjunto de datos con *600* escenas de sonido simuladas de *45* segundos cada una que representa entornos de sonido urbanos. El conjunto de datos se divide en dos partes: un subconjunto de entrenamiento formado por *400* escenas y un subconjunto de prueba compuesto por *200* escenas.

Cada escena se compone de tres fuentes principales: tráfico, voces humanas y pájaros según un escenario original que se compone de forma semi-aleatoria en base a cinco ambientes: Parque, calle tranquila, calle ruidosa, calle muy ruidosa y plaza. Los archivos de audio base utilizados para la simulación se obtienen de las plataformas Freesound [86] y LibriSpeech [143].

3.3.15 SimSceneTVB Perception

SimSceneTVB Perception es un conjunto de datos de audio formado por *100* escenas sonoras de *45* segundas cada una que representan entornos sonoros urbanos. Incluyen: *6* escenas grabadas en París, *19* escenas simuladas usando simScene [144] para replicar escenarios grabados, *6* grabaciones de este corpus y *75* escenas simuladas usando simScene [144]. Las últimas emplean diversos escenarios nuevos con tráfico, voces humanas y fuentes de aves. Los archivos de audio base utilizados para la simulación se obtienen de las plataformas Freesound [86] y LibriSpeech [143].

3.3.16 Sound Events for Surveillance Applications (SESA)

Sound Events for Surveillance Applications (SESA) [145], como su nombre indica, es un conjunto de datos de eventos de sonido para aplicaciones de vigilancia obtenidos de las plataforma Freesound [86]. El conjunto de datos se subdividió en *480* archivos para la fase de entrenamiento y *105* para evaluación. Todos los archivos de audio tienen el formato *WAV*, son audios monocanal de *16* kHz y *8* bits de hasta *33* segundos de duración. Las distintas clases de eventos sonoros que registra esta base de datos son: Sonidos casuales (no representa una amenaza), disparos, explosiones y sirenas (también contiene alarmas).

3.3.17 FSD50K

Freesound Dataset 50K o FSD50K [146] es un conjunto de datos abierto con sonidos etiquetados manualmente que contiene *51197* clips de audios proveniente de la plataforma Freesound [86] y distribuidos de manera desigual en *200* clases. Su clasificación ha sido tomada de la plataforma AudioSet

Ontology [109], a excepción de las clases: "platillo crash, acciones grupales humanas, voz humana, sonidos respiratorios, y sonidos domésticos y sonidos de hogar".

El contenido de audio se compone principalmente de eventos acústicos producidos por fuentes de sonido físicas y mecanismos de producción, incluidos sonidos humanos, sonidos de cosas, animales, sonidos naturales, instrumentos musicales, etc. Los clips tienen una duración variable de *0.3* a *30* segundos, debido a la diversidad de clases de sonido y las preferencias de los usuarios de Freesound [86] al grabar sonidos.

Todos los clips de audio se proporcionan como archivos de audio *mono PCM* de *16* bits y *44.1* kHz sin comprimir. Los clips de audio se agrupan en un conjunto de entrenamiento y un conjunto de evaluación, sin contener clips repetidos en cada conjunto.

El *set*de entrenamiento consta de *40966* clips de audio con un total de *80.4* horas y una duración media de *7.1* segundos. Donde las etiquetas son correctas, pero en ocasiones pueden estar incompletas. Contiene además *114271* etiquetas "manchadas" o no verificadas manualmente. Para este conjunto se proporciona una división de entrenamiento y validación.

El conjunto de evaluación consta de *10231* clips de audio con un total de *27.9* horas de audio y una duración media de clip de *9.8* segundos, en donde las etiquetas son correctas y completas para el vocabulario considerado. Este conjunto contiene además *38596* etiquetas "manchadas".

Las *200* clases de eventos sonoros son: Aceleración y sonido de motor, acordeón, guitarra acústica, aire acondicionado, bocina de aire y bocina de camión, aeronave, motor de avión, alarma, despertador, saxofón alto, ambulancia (sirena), animal, aplausos, flecha, fuego de artillería, balbuceo, llanto de bebé, risa de bebe, ladrido, tambor de bajo, bajo, fagot, bañera (llenado o lavado), grito de batalla, abeja y avispa, campana, bicicleta, timbre de bicicleta, pájaro, vuelo de pájaro y aleteo de alas, vocalización y canto de pájaros, gemido, licuadora, barco y vehículo acuático, hirviendo, auge (retumbar), instrumento de cuerda frotada, instrumento de metal, sonidos respiratorios, eructar y eructos, estallar, autobús, señal de ocupado, zumbido, cámara, pistola de fogueo, coche, alarma de coche, coche pasando, caja registradora, gato, ganado de vacas y bovinos, graznar, violonchelo, zumbido del teléfono móvil y alerta vibratoria, motosierra, canto, charla, aplausos, masticación, pollo y gallo, niño cantando, niño hablando, niños jugando, niños gritando, repicar, tintineo y sonido metálico, ardilla, chirridos y pitidos, coro, picar, picar (comida), risa, campana de iglesia, sirena de defensa civil, aplausos, clarinete, clip-clop, reloj, cloqueo, ruido, moneda (cayendo), teclado, conversación, arrullo, tos, cencerro, grieta, crepitar, platillo crash, grillo, croar, cuervo, multitud, cacareo y gallo garabato doo, arrugar y crujir, aplastar, llorando y sollozando, abrir o cerrar armario, cubiertos, platillo, taladro dental y taladro de dentista, tono de marcación, didgerido, ding-dong, platos y ollas y sartenes, perro, puerta, timbre de la puerta, contrabajo, abrir o cerrar cajón, perforar, goteo, tambor, kit de batería, redoble de tambores, pato, guitarra eléctrica, Piano eléctrico, afeitadora eléctrica y maquinilla de afeitar eléctrica, cepillo de dientes eléctrico, órgano electrónico, vehículo de emergencia, motor, arranque del motor, explosión, pedo, canto femenino, discurso femenino y mujer hablando, limando (escofina), llenar (con líquido), chasquear los dedos, fuego, alarma de incendios, camión

de bomberos y su sirena, petardo, fuegos artificiales, avión y avión de ala fija, petardo, volar, sirena, cuerno francés, rana ,freír (comida), descarga cerrada, hacer gárgaras, fadear, engranajes, risilla, vaso, glockenspiel, cabra, gong, ganso, gruñido, guitarra, gaviota, disparos, gluglú, secador de pelo, martillo, órgano, hammond, manos, armónica, arpa, clave, sonidos y latidos del corazón, helicóptero, platillos, silbido, bocinazo, ulular, caballo, aullido, aullido (viento), alboroto y ruido del habla y balbuceo del habla, voz humana, acciones grupales humanas, de marcha en vacío, insecto, martillo neumático, cascabel, teclado (musical), tintineo de llaves,sonidos domésticos, sonidos de hogar, golpe, risa, cortacésped, motor ligero (alta frecuencia), líquido, ametralladora, canto masculino, habla masculina y hombre hablando, percusión de mazo, mantra, maraca, marimba y xilófono, ventilador mecánico, mecanismos, motor medio (frecuencia media), maullar, horno microondas, mugir, mosquito, vehículo de motor (carretera), motocicleta, ratón, música, narración y monólogo, relincho, oboe, océano, gruñir, orquesta, órgano, búho, cinta de embalaje y cinta adhesiva, pantalón, percusión, piano, cerdo, paloma, pavo, pizzicato, instrumento de cuerda pulsada, coche de policía (sirena), herramienta eléctrica, elevalunas eléctricos, impresora, poleas, ronroneo, curandero, coche de carreras y automovilismo, lluvia en la superficie, gota de agua, golpecitos, trinquete, traqueteo, sonajero (instrumento), piano de Rhodes, rimshot, tono de llamada, rugido, roedores y ratas y ratones corriendo, susurro de hojas, velero, lijado, aserradura, saxofón, tijeras, rascarse (técnica de interpretación), grito, chillido, máquina de coser, romper, oveja, embarcación, barajar cartas, suspiro, canto, cuenco cantor, fregadero (llenando o lavando), sirena, sitar, patineta, patinando, golpe, puerta corrediza, chapotear, chasquido, tambor, estornudo, risa disimulada, oler, ronquidos, snort (caballo), saxofón soprano, habla, sintetizador de voz, salpicaduras, astilla, rociar, graznido, chirrido de vapor, silbato de vapor, guitarra de acero y guitarra slide, remover, corriente, rasguear, metro, canto sintético, tabla, pandereta, grifo, desgarro, teléfono, sonido del timbre de teléfono, marcación telefónica y DTMF Theremin, limpieza de garganta, golpe, trueno y tormenta, thunk, garrapata, TIC Tac, tímpanos, botar el agua del baño, herramientas, cepillo de dientes, ruido del tráfico y de la calzada, entrenar, bocina de tren, silbato de tren, trombón, camión, trompeta, campanas tubulares, máquina de escribir, mecanografía, ukelele, aspiradora, bocina de vehículo, cierre de velcro y velcro, vibráfono, violín, Caminata y pisadas, Grifo de agua, cascada, olas y surf, susurro, viento, instrumento de viento e instrumento de viento madera, ruido del viento (en un micrófono), madera, escritura, bostezo, grito "Yip Yodeling", cremallera (ropa).

3.3.18 SONYC Urban Sound Tagging (SONYC-UST)

SONYC Urban Sound Tagging (SONYC-UST) [147] es un conjunto de datos sonoros etiquetados procedentes de una red de sensores acústicos urbanos. Está orientado en el desarrollo y evaluación de sistemas de escucha de máquinas para una monitorización realista del ruido urbano. En concreto, los archivos de

audios fueron grabados desde la red de sensores acústicos SONYC para el control de la contaminación acústica urbana a través de *60* sensores diferentes en la ciudad de Nueva York. Voluntarios de la plataforma de ciencia ciudadana *zooniverso* [148] marcaron la presencia de *23* clases que fueron elegidas en consulta con el *Departamento de Protección Ambiental de la Ciudad de Nueva York* [149]. Estas *23* clases se pueden agrupar a su vez en otras *8* clases.

Las grabaciones se dividen en tres conjuntos: entrenamiento, validación y prueba. Los conjuntos de entrenamiento y validación están separados con respecto al sensor del que procede cada grabación y el conjunto de prueba se desplaza en el tiempo.

El subconjunto de entrenamiento contiene *13538* grabaciones de *35* sensores, el subconjunto de validación está formado por *4308* grabaciones de *9* sensores y el subconjunto de prueba por *669* grabaciones de *48* sensores. Todas las grabaciones son de *10* segundos con polifonía variable, grabadas con micrófonos idénticos con configuraciones de ganancia idénticas.

La distribución de las clases de eventos sonoros es la siguiente:

- *Motor*: Motor de sonido pequeño, motor de sonido medio, motor de sonido grande, motor de tamaño incierto..
- *Paisajes sonoros naturales y sonidos de agua*: Lluvia, olas del mar, fuego crepitante, grillos, pájaros piando, gotas de agua, viento, echar agua, gotear el agua del baño y tormenta.
- *Impacto de maquinaria*: Taladro de roca, martillo neumático, azada, destornillador, otra maquinaria de impacto desconocida.
- *Impacto sin maquinaria*: Impacto sin maquinaria.
- *Señal de alerta*: Bocina del automóvil, alarma del automóvil, sirena, señal acústica inversa X: otra señal de alerta desconocida.
- *Música*: Música fija, música móvil, camión de helados, música de fuente incierta.
- *Voz humana*: Conversación de persona o grupo pequeño, gritos de persona o grupo pequeño, multitud grande, habla amplificada, voz humana desconocida.
- *Perro*: Perro ladrando.

3.3.19 FSDKaggle2018

Freesound Dataset Kaggle 2018 o FSDKaggle2018 [150] es un conjunto que contiene *11073* archivos de audio anotados con *41* etiquetas diferentes pertenecientes a distintos eventos sonoros. Su estructura a la hora de realizar la clasificación de estas etiquetas ha sido tomada desde la plataforma AudioSet Ontology [109] que se ha utilizado para el *workshop* de la tarea 2 del *challenge* DCASE 2018 [111] de IEEE AASP [91].

FSDKaggle2018 es un subconjunto reducido de Freesound Dataset (FSD) [151] con datos de audio de propósito general a gran escala y en progreso. Está compuesto por contenido de Freesound [86] anotado con etiquetas de AudioSet Ontology [109]. A la hora de generar el conjunto de datos FSDKaggle2018, varias de las etiquetas se han verificado manualmente, mientras que el resto no y

por lo tanto, algunas de ellas podrían ser inexactas. Todas los archivos de audio de este conjunto tienen una sola etiqueta, es decir, solo se le asigna un tipo de evento, aunque es posible que contenga más de un evento simultáneamente. Los archivos de audio son *mono PCM* de *16* bits y *44.1* kHz sin comprimir con una duración de entre *0.3* y *30* segundos. El conjunto `FSDKaggle2018` se encuentra distribuido en dos subconjuntos: entrenamiento y evaluación.

El subconjunto de entrenamiento contiene aproximadamente *9500* muestras, de las cuales, alrededor de *3700* tienen anotaciones verificadas manualmente y alrededor de *5800* tienen anotaciones no verificadas. Las anotaciones no verificadas carecen de una estimación de calidad de, al menos, entre un *65* % y un *70* % en cada categoría. El número mínimo de muestras de audio por categoría en este subconjunto es de *94* y *300* respectivamente.

El subconjunto de prueba se compone de *1600* muestras con anotaciones verificadas manualmente y con una distribución de categorías similar a la del subconjunto de entrenamiento. La duración total en este caso es de aproximadamente *2* horas.

Las *41* clases de eventos sonoros que contiene este conjunto son: `Guitarra acústica, aplausos, ladrido, bombo, eructos, autobús, violonchelo, timbre, clarinete, teclado de computadora, tos, cencerro, contrabajo, abrir o cerrar cajón, piano eléctrico, pedo, chasquido de dedos, fireworks, flauta, glockenspiel, gong, disparo o tiroteo, armónica, hi-hat ,tintineo de llaves, llamar a la puerta, risa, maullido, horno de microondas, oboe, saxofón, tijeras, romper, caja (instrumento de percusión),crujido, pandereta, desgarro, teléfono, trompeta, violín o fiddle, escritura.`

3.3.20 FSDKaggle2019

`Freesound Dataset Kaggle 2019` o `FSDKaggle2019` [152] es un conjunto de datos de audio que contiene *29266* archivos de audio anotados con *80* etiquetas diferentes pertenecientes a distintos eventos sonoros. Su estructura a la hora de realizar la clasificación de estas etiquetas ha sido tomada desde la plataforma `AudioSet Ontology` [109], utilizada en el *workshop* de la tarea 2 del *challenge* DCASE 2019 [153] de IEEE AASP [91]. La competición tenía como objetivo proporcionar información sobre el desarrollo de clasificadores de eventos acústico de amplia aplicación capaces de hacer frente al ruido de las etiquetas y las condiciones mínimas de supervisión.

`FSDKaggle2019` emplea clips de audio de las siguientes fuentes: texttttFreesound Dataset (FSD) [151] y las bandas sonoras de un grupo de vídeos de *Flickr* tomados del conjunto de datos de `Yahoo Flickr Creative Commons 100M (YFCC100M)` [154]. El conjunto `FSDKaggle2019` se divide a su vez en dos subconjuntos de entrenamiento y uno de evaluación.

El primer subconjunto de entrenamiento consta de datos etiquetados manualmente procedentes de `(FSD)` [151] con *75* clips de audio por cada clase excepto en algunos casos donde hay menos. Este subconjunto consta de *4970* clips de audio con un número medio de etiquetas por clip de *1.2*. La duración total del subconjunto es de *10.5* horas, siendo la duración de los clips de audio variable de *0.3* a *30* segundos. Las etiquetas son correctas pero potencialmente incompletas.

El segundo subconjunto de entrenamiento contiene los archivos de audio ruidosos (sin validación humana) de vídeos de *Flickr* tomados del conjunto de datos `YFCC10M` [154]. Este subconjunto consta de *19815* clips de audio con un número medio de etiquetas por clip de *1.2*. La duración total del subconjunto es de *80* horas, con un promedio de *300* de clips por clase donde la duración de los clips de audio varía

de *1* a *15* segundos. Las etiquetas se generan automáticamente y son ruidosas a propósito. El ruido de la etiqueta puede variar ampliamente en cantidad y tipo según la clase de evento sonoro.

El subconjunto de prueba se utiliza para la evaluación del sistema y consta de datos etiquetados manualmente procedentes de (FSD) [151]. Este subconjunto consta de *4481* clips de audio con un número medio de etiquetas por clip de *1.4*. La duración total del subconjunto es de *12.9* horas con, entre *50* y *150* clips por clase. La duración de los clips de audio varía de *0.3* a *30* segundos. Las etiquetas son correctas y completas. No obstante, algunos clips podrían presentar contenido acústico adicional sin etiquetar.

En FSDKaggle2019 existen *80* clases de eventos sonoros que cubren diversos temas: Aceleración con sonido de motor, acordeón, guitarra acústica, guitarra eléctrica, glockenspiel, gong, bajo,bombo, xilófono, armónica, silbido, hi-hat, aplausos, ladrido, maullido, piar de pájaros, bañera (llenado o lavado), timbre de bicicleta, eructos, autobús, zumbido, coche pasando, vítores (ovación con aplausos), masticar, chink and clink, campana de iglesia, dar una palmada, teclado de computadora, crujido, grillo, multitud, armario abierto o cerrado, cubiertos, platos y ollas y sartenes, cajón abierto o cerrado, goteo, pedo, relleno (con líquido), chasquido de dedos, freír (comida), jadeo, Gorgoteo (sonido burbujeante hueco), llaves tintineando, golpe, hombre cantando, habla infantil y niño hablando, hombre hablando, mujer cantando, mujer hablando, ventilador mecánico, horno de microondas, motocicleta, Impresora, ronroneo, carreras de autos, gota de lluvia, correr, tijeras, grito, destrozar (shatter), suspiro, fregadero (llenado o lavado), monopatín, portazo, estornudar, chirrido, arroyo, rasgueo, toque (dar un golpecito), tic-tac, inodoro, ruido de tráfico y de carretera, goteo con un chorrito, pasos, grifo de agua, olas, susurrar, escribir, grito (para animar), cremallera (ropa).

Capítulo 4

Implementación y resultados experimentales

"Si quieres cambiar al mundo, cámbiate a ti mismo."

Mahatma Gandhi

4.1 Introducción

En este capítulo se seleccionará un sistema de referencia y se describirá todo el proceso realizado para trabajar a partir de este, explicando el método de trabajo seguido, la funcionalidad de los diferentes archivos necesarios (scripts de `Python` utilizados para el pre-procesado de audio de las bases de datos, el entrenamiento de la red del sistema de referencia y sus derivados, además de para la evaluación de los modelos generados de cada sistema), todas las modificaciones realizadas al sistema de referencia y los resultados obtenidos de cada una de ellas, entre otras cosas.

Como acabamos de mencionar, el primer paso es escoger uno de los sistemas disponibles (descritos en el apartado 3.2) como sistema de referencia dentro de este conjunto de sistemas eficientes relacionados con el reconocimiento y clasificación de diferentes eventos sonoros. Nos referiremos a este sistema como sistema de referencia.

Este sistema de referencia será ejecutado para supervisar cómo se desarrolla dicho entrenamiento y comprobar la validez de su funcionamiento (si ofrece inicialmente resultados similares a los especificados por sus creadores).

Una vez que se ha comprobado que el sistema funciona correctamente, el siguiente paso es realizar una serie de modificaciones (detalladas más adelante) que se resumen en:

- *Evaluar* el sistema de referencia utilizando otra base de datos que posee algunas de las clases de eventos sonoros de la base de datos inicial (base de datos con la que el sistema se entrenó de forma predeterminada), en lugar de los datos de evaluación extraídos de su base de datos inicial.

- *Entrenar* y *evaluar* el sistema con otra base de datos diferente a la base de datos inicial con la que el sistema se entrenó de forma predeterminada. A este sistema lo llamaremos `sistema urban` durante el resto de la memoria.

- *Entrenar* y *evaluar*, tanto el `sistema de referencia` (con su base de datos predeterminada) como el `sistema urban` (con otra base de datos utilizada para entrenar y evaluar el sistema) realizando *modificaciones en la arquitectura de la CRNN*.

Previamente a exponer todas las modificaciones realizadas a partir del sistema de referencia, se va a describir la composición del sistema de referencia desde el punto de vista de los diferentes archivos que son necesarios para su funcionamiento.

Esta misma dinámica se aplicará para cada nuevo sistema resultante de una modificación realizada a partir del sistema de referencia (anteriormente mencionado), haciendo alusión a los archivo que se utilizan y describiendo los archivos nuevamente creados o modificados, especificando su funcionamiento o los cambios cometidos respecto al archivo original.

Para cada uno de estos sistemas generados (incluyendo el sistema de referencia) se adjuntarán los resultados obtenidos tras realizar la evaluación de sus predicciones respecto a un conjunto de evaluación, reflejándolos en tablas que muestran las distintas métricas evaluadas y se comentarán estos resultados posteriormente.

4.2 Método de trabajo

Para desarrollar esta serie de propósitos comentados en este capitulo, se ha trabajado con un ordenador remoto que posee un sistema operativo `GNU/Linux`, en el que se han descargado todas las herramientas necesarias (sistemas, bases de datos, librerías para la ejecución de los sistemas, etc).

Se ha utilizado `Python` como lenguaje de programación y algunas de sus librerías externas (las más importantes descritas en 2.5.1) para poder leer, crear, modificar y ejecutar los diferentes archivos (`scripts de Python`) asociados al sistema de referencia, sus modificaciones y todos aquellos archivos que se han utilizado para poder evaluar las predicciones procedentes de los diferentes modelos generados. Por si fuera necesario el uso del múltiples versiones de `Python` utilizó un entorno virtual `Pyenv` [155], aunque finalmente solo se utilizó la versión *3.7.3* de `Python` que se instaló en este entorno virtual con todas las librerías necesarias.

En cuanto a la organización, todos los archivos que se han utilizado en este capítulo, han sido distribuidos en *3* directorios diferentes dentro del ordenador remoto.

- Uno de los directorios se corresponde con una carpeta donde se almacenan las diferentes bases de datos a utilizar. Dentro de esta carpeta, cada base de datos se encontrará dentro de otra carpeta independiente, donde se encontraran los diferentes archivos de audio, sus etiquetas y los archivos procesados (una vez que se haya realizado el pre-procesamiento de estos datos).

- Otro de los directorios se trata de la carpeta donde se encuentran todos los archivos correspondientes al sistema de referencia (descritos en el apartado 4.3.2), además de los scripts que se utilizan para realizar modificaciones a este sistema para evaluar el rendimiento de estos cambios posteriormente (las modificaciones realizadas se comentan en el apartado 4.2.3). En este directorio también se encuentran los scripts encargados de realizar y almacenar las predicciones de los diferentes modelos entrenados sobre los conjuntos de evaluación asignados, con el objetivo de suministrar estos datos posteriormente al sistema de evaluación.

- El último directorio consiste en la carpeta donde reside el sistema de evaluación (descrito en el apartado 4.2.1.1) que se utiliza para evaluar el rendimiento de las predicciones realizadas por los modelos generados sobre unos conjuntos de datos de evaluación, midiendo varias métricas.

4.2.1 Método de evaluación de sistemas

Como se ha mencionado a lo largo de este trabajo, un modelo generado a partir de un sistema SED, entrenado mediante el uso de una o varias ANNs, se encarga de realizar predicciones sobre el conjunto de datos de audio que se le proporcione. El objetivo es que estas predicciones sean lo más acertadas posible, pudiendo predecir las diferentes clases de eventos sonoros (para las que este sistema se haya diseñado) que se encuentren activas para cada instante en los datos de audio proporcionados.

La capacidad de identificar estas clases de eventos sonoros en datos que el sistema no haya empleado durante su entrenamiento (conjunto de entrenamiento y conjunto de validación) puede ser bastante distante de la esperada en algunas ocasiones. Es por ello que habitualmente, una vez que un sistema se haya entrenado, éste debe ser evaluado con un conjunto de datos externos diferente de los datos empleados durante el entrenamiento (conjunto de entrenamiento y conjunto de validación) llamado comúnmente conjunto de evaluación. En algunas bases de datos se suele tener en cuenta este conjunto, destinándole una partición de los datos totales, pero en otras ocasiones, esta partición hay que realizarla de manera manual.

Evaluar la eficacia de un modelo[1] respecto a este tipo de datos es crucial para saber si el sistema funciona correctamente o no cumple unos requisitos mínimos de calidad y poder valorar si su funcionamiento es mejor o peor respecto a otros sistemas con el mismo cometido (o al mismo sistema con algunas modificaciones).

Durante este apartado de implementación contaremos con distintos sistemas a los que hemos aplicado algunas modificaciones sobre un sistema de referencia. La manera en la que estos sistemas generan las predicciones es siempre la misma, generando para cada ventana temporal procesada un *valor* para cada clase de evento sonoro en un rango de *0* a *1*, que representa la probabilidad de que un evento se encuentre activo en ese instante.

4.2.1.1 Sistema de evaluación: Sed eval

Para evaluar las diferentes métricas de cada sistema respecto a un conjunto de datos de evaluación, se emplea un sistema conocido como `Sed eval` [156]. Inicialmente este sistema se se utilizó para evaluar el rendimiento de los diferentes sistemas presentados en el *workshop* de la tarea 3 del *challenge* DCASE 2017 [119] de IEEE AASP [91]; de donde se seleccionó nuestro sistema de referencia.

`Sed eval` [156] es una herramimenta escrita en Python y de código abierto que proporciona una forma estandarizada y transparente de evaluar los sistemas de detección de eventos acústicos y los sistemas de clasificación de escenas acústicas, donde en esta ocasión se utilizará la parte destinada a detección de eventos acústicos.

Para utilizar esta herramienta se necesita, como mínimo (en el caso de evaluar un único modelo), un archivo de texto (en formato CSV) que contengan una lista de eventos sonoros detectados (predicciones) de un conjunto de datos y un archivo de texto (en formato CSV) que contengan una lista de eventos sonoros de referencia de ese conjunto de datos (etiqueto verdadero), pudiendo estar los eventos en cualquier orden.

Si un sistema genera más de un modelo, como puede ser el caso del sistema de referencia y sus modificaciones, que utilizan una estrategia de validación cruzada de *4* pliegues (o *folds*); a esta herramienta habrá que suministrarle *4* archivos con las predicciones (*1* de cada pliegue) y *4* archivos de texto de etiquetado real (1 de cada pliegue).

[1] Cuando se hable de un `modelo`, durante este trabajo, se hace referencia siempre a un modelo generado a partir de un proceso de entrenamiento supervisado.

Estos archivos (`archivo de etiquetado verdadero` y `archivo de predicciones`) pertenecientes a cada modelo, contienen un conjunto de listas de eventos sonoros con una distribución de un evento por cada fila con el siguiente formato: `[inicio del evento] [delimitador] [finalización del evento] [delimitador] [etiqueta]`, utilizando como delimitadores compatibles: *,, ; y tabulación.*

En la herramienta `Sed eval`, los cuatro pliegues de validación cruzada se tratan como un solo experimento, lo que significa que las métricas se calculan solo después de entrenar y probar todos los pliegues, no como promedio de los pliegues individuales ni como promedio del rendimiento de clase individual. Las medidas intermedias de todos los pliegues se acumulan antes de calcular las métricas, tal y como se describe en [157].

Para realizar las evaluaciones ejecutaremos un *script* de Pyhton (*sound_event_eval.py*), que se encuentra dentro de la carpeta `evaluators` situada en la carpeta raíz de `Sed eval`, el cual realizará el papel de evaluador. Este *script* toma como argumento la lista de archivos de texto de `etiquetado verdadero` y `predicciones`, la cual es introducida en el documento *file_list.txt* que se encuentra en su misma carpeta.

Esta lista contiene pares de nombres de archivo, concretamente, un par por cada fila: El primer nombre de cada fila hace referencia al nombre del archivo que contiene una lista de eventos de referencia (etiquetado verdadero) y el segundo, al archivo que contiene una lista de eventos de las estimaciones de un modelo (predicciones).

El formato para cada fila es: `[archivo de referencia] [delimitador] [archivo de estimación]`, y los delimitadores compatibles son: *,, ; y tabulación.* A la hora de evaluar, como vamos a utilizar siempre *4* pliegues (generamos *4* modelos para cada sistema entrenados con diferentes particiones de la base de datos a utilizar) en nuestro caso, se emplean *4* filas (*1* para cada pliegue con su archivo de referencia y de estimación).

Una vez ejecutado el archivo *sound_event_eval.py* se obtienen unos resultados que muestran la evaluación de diferentes métricas descritas en [156] que se encuentran divididas en dos categorías principales: *Métricas basadas en segmentos y métricas basadas en eventos.*

- *Métricas basadas en segmentos*: la etiqueta real y la predicción del sistema se comparan en una cuadrícula de tiempo fijo; los eventos acústicos se marcan como activos o inactivos en cada segmento.
- *Métricas basadas en eventos*: la etiqueta real y la predicción del sistema se comparan a nivel de instancia de evento.

4.2.1.2 Estadísticas intermedias

4.2.1.2.1 Métricas basadas en segmentos

- `Verdadero positivo`: el archivo de referencia y el de predicciones del sistema indican que un evento está activo en un segmento.
- `Falso positivo`: el archivo de referencia indica que un evento se encuentra inactivo en un segmento, pero el archivo de predicciones del sistema indica que está activo.
- `Falso negativo`: el archivo de referencia indica que un evento se encuentra activo en un segmento, pero el archivo de predicciones del sistema indica que está inactivo.
- `Verdadero negativo`: el archivo de referencia y el de predicciones del sistema indican que un evento está inactivo en un segmento.

4.2.1.2.2 Métricas basadas en eventos

- `Verdadero positivo`: se indica un evento en el archivo de predicciones del sistema que tiene una posición temporal que se superpone con la posición temporal de un evento con la misma etiqueta en el archivo de referencia. Por lo general, se permite cierto *collar* (margen) de error para el inicio y el desplazamiento, o una tolerancia con respecto a la duración del evento del archivo de referencia.
- `Falso positivo`: un evento en el archivo de predicciones del sistema no tiene correspondencia con un evento con la misma etiqueta en el archivo de referencia dentro de la tolerancia permitida.
- `Falso negativo`: un evento en el archivo de referencia no tiene correspondencia con un evento con la misma etiqueta en el archivo de predicciones del sistema dentro de la tolerancia permitida.

4.2.1.3 Promedios utilizados

- `Micropromedio`: las estadísticas intermedias se agregan sobre todos los datos de prueba y luego se calculan las métricas; cada instancia tiene la misma influencia en el valor métrico final.
- `Macropromedio`: las estadísticas intermedias se agregan por clase, se calculan métricas basadas en clases y luego se calcula el promedio de las métricas basadas en clases; cada clase tiene la misma influencia en el valor métrico final.

Los promedios micro y macro pueden dar como resultado valores muy diferentes cuando las clases están muy desequilibradas o el rendimiento en clases individuales es muy diferente.

4.2.1.4 Métricas implementadas

En este apartado se va a definir las diferentes métricas que se encarga de evaluar el sistema `Sed eval`. La mayoría de las métricas se pueden calcular `en base a segmentos o eventos` utilizando `micro o macro` promediados. Para definir las diferentes métricas se va a utilizar la nomenclatura para los siguientes términos:

- *TP* para referirse al número total de `verdaderos positivos` (del ingles *True positive*).
- *FP* para referirse al número total de `falsos positivos` (del ingles *False positive*).
- *FN* para referirse al número total de `falsos negativos` (del ingles *False negative*).
- *TN* para referirse al número total de `verdaderos negativos` (del ingles *True negative*).

4.2.1.4.1 Precisión, recuperación y puntuación F Estas métricas se pueden calcular en base a segmentos o eventos, micro o macro promediados.

- `Precisión (P):`
$$P = \left(\frac{TP}{TP + FP}\right) \tag{4.1}$$
- `Recuperación (R):`
$$R = \left(\frac{TP}{TP + FN}\right) \tag{4.2}$$
- `Puntuación media (F):`
$$F = \left(\frac{2 \cdot P \cdot R}{P + R}\right) \tag{4.3}$$

4.2.1.4.2 Sensibilidad, especificidad, exactitud y exactitud equilibrada La sensibilidad, especificidad y exactitud solo se calculan como métricas basadas en segmentos, ya sea utilizando micro o macro promediados.

- `Sensibilidad (S):`

$$S = \left(\frac{TP}{TP + FN}\right) \tag{4.4}$$

- `Especificidad (E):`

$$E = \left(\frac{TP}{TP + FN}\right) \tag{4.5}$$

- `Exactitud (EX):`

$$EX = \left(\frac{TP + TN}{TP + TN + FP + FN}\right) \tag{4.6}$$

- `Exactitud equilibrada (EXq):`

$$EXq = \left(\frac{0{,}5 \cdot TP}{TP + FN} + \frac{0{,}5 \cdot TN}{TN + FP}\right) \tag{4.7}$$

4.2.1.4.3 Tasa de sustitución , tasa de inserción, tasa de eliminación y tasa de error

- Basadas en segmentos
  - `Tasa de sustitución en un segmento` k o `S(k)`: es la tasa de eventos que muestra el archivo de referencia para los que no se predijo un evento correcto en el archivo de predicciones, pero sí otro evento diferente. Una sustitución equivale a tener un falso positivo y un falso negativo en el mismo segmento. No es necesario designar qué evento erróneo sustituye a cuál.

$$S(k) = min(FN(k), FP(k)) \tag{4.8}$$

  - `Tasa de inserción en un segmento` k o `I(k)`: es la tasa de eventos predichos que no son correctos (se contabilizan los falsos positivos después de las sustituciones).

$$I(k) = max(0, FP(k) - FN(k)) \tag{4.9}$$

  - `Tasa de eliminación en un segmento` k o `D(k)`: es la tasa de eventos en el archivo de referencia que no son correctos (se contabilizan los falsos positivos después de las sustituciones).

$$D(k) = max(0, FN(k) - FP(k)) \tag{4.10}$$

  - `Tasa de error (ER):`

$$ER = \left(\frac{\sum_{k=1}^{K}(S(k)) + \sum_{k=1}^{K}(D(k)) + \sum_{k=1}^{K}(I(k))}{\sum_{k=1}^{K}(N(k))}\right) \tag{4.11}$$

    Donde *N(k)* es el número de eventos en el segmento k indicados en el archivo de referencia.

- Basadas en eventos
  - `Tasa de sustitución`: tasa de eventos predichos con posición temporal correcta pero etiqueta de clase incorrecta.

- `Tasa de inserción`: tasa de eventos predichos no contabilizados como correctos o sustituidos.
- `Tasa de eliminación`: tasa de eventos del archivo de referencia que no se consideran correctos o sustituidos.
- `Tasa de error (ER)`:

$$ER = \left(\frac{S + D + I}{N}\right) \tag{4.12}$$

Donde N es el número total de eventos indicados en el archivo de referencia.

4.2.2 Generación de las predicciones de los diferentes modelos

Dado que los archivos proporcionados del sistema de referencia solo se encargan de generar un modelo para cada pliegue de validación cruzada, hubo que diseñar un script de Python que se encargara de que cada modelo correspondiente a cada pliegue de validación cruzada de cada sistema realice predicciones sobre sus respectivos conjuntos de evaluación (para cada pliegue), las almacene y las adapte al formato de archivo de lista de eventos que se corresponda con los `archivos de estimación` que se le suministran al sistema de evaluación (`Sed eval`).

A este archivo se le nombró como `convertidor_salida_evaluacion.py`, el cual se situó en el directorio correspondiente a la carpeta del sistema de referencia.

Este archivo se diseñó para ejecutarse una vez para cada sistema, suministrando *1* archivo de lista de eventos para cada uno de los modelos de validación cruzada (*4* archivos de lista de eventos en total) correspondiente a las predicciones sobre el conjunto de evaluación asignado a cada pliegue de validación cruzada.

Para ello, este script requiere que se le especifique:

- La ruta de la carpeta donde se encuentran los archivos procesados del conjunto de datos sobre el que se realizarán las predicciones.
- La ruta de la carpeta donde se depositaran los archivos de listas de eventos.
- La ruta de los modelos correspondientes a cada pliegue de validación cruzada.

Del mismo modo, para generar los archivos de las listas de eventos que se corresponden con los *archivos de referencia* (etiquetado verdadero) que se suministran al sistema de evaluación (`Sed eval`), se puede hacer uso de un archivo llamado `convertidor_salida_evaluacion_NO_PREDICT.py`.

Este archivo consiste en una ligera modificación del archivo `convertidor_salida_evaluacion.py`, el cual, se encarga esta vez de seleccionar las etiquetas asignadas a los archivos de audio correspondientes a los conjuntos de evaluación de cada pliegue (en lugar de las predicciones que realizan los modelos) y adaptarlas al formato deseado (archivos de listas de eventos para cada pliegue de validación cruzada). El archivo `convertidor_salida_evaluacion_NO_PREDICT.py` también se situó en el directorio correspondiente a la carpeta del sistema de referencia.

4.2.3 Modificaciones y evaluaciones realizadas

Para este trabajo se propuso generar varios sistemas a partir de realizar ciertas modificaciones a un sistema de referencia, con el objetivo de evaluar posteriormente el rendimiento de cada sistema comparando sus resultados obtenidos. Las diferentes evaluaciones que se realizan son:

- Evaluación del *sistema de referencia* (sin realizar ningún cambio).
- Evaluación del *sistema de referencia utilizando otra base de datos como conjunto de evaluación.*
- Evaluación del *sistema de referencia que utiliza una base de datos diferente con otras clases de eventos sonoros* (nuevo conjunto de entrenamiento, validación y evaluación para cada pliegue de validación cruzada procedente de otra base de datos). Para poder hacer referencia a este sistema cómodamente sin tener que mencionar todo lo anterior, a este sistema se le apodó como `sistema urban` (debido a que la base de datos que utiliza es Urbansound8k).
- Evaluación del *sistema de referencia y el sistema urban añadiendo una capa convolucional en cada bloque convolucional.*
- Evaluación del *sistema de referencia y el sistema urban modificando el valor de* Dropout, analizando para valores de *0.5* (predeterminado) a *0.3* y *0.7* respectivamente, creando *2* nuevas variantes para cada sistema.
- Evaluación del *sistema de referencia y el sistema urban modificando la dimensión de las* matrices de Kernel *de las capas convolucionales.* La dimensión de las *matrices de Kernel* se modifican de *3x3* (predeterminado) a *5x5* y *7x7* respectivamente, creando *2* nuevas variantes para cada sistema.
- Evaluación del *sistema de referencia y el sistema urban utilizando la función de activación* ELU en los bloques convolucionales en lugar de utilizar la función ReLU que se utiliza de forma predeterminada.
- Evaluación del *sistema de referencia y el sistema urban utilizando la función de activación* PReLU en los bloques convolucionales en lugar de utilizar la función ReLU que se utiliza de forma predeterminada..

4.3 Sistema de referencia

4.3.1 Descripción y condiciones experimentales

El sistema escogido en este trabajo finalmente como sistema de referencia se trata del propuesto por Sharath Adavanne y Tuomas Virtanen [7] para el *workshop* de la tarea 3 del *challenge* DCASE 2017 [119] de IEEE AASP [91], el cual, se describe en el apartado 3.2.14.

Este sistema realiza una tarea SED, teniendo como objetivo reconocer distintos eventos sonoros en condiciones de fuentes múltiples utilizando una CRNN. La descripción más detallada de este sistema se encuentra en el apartado 3.2.14, al igual que en [7].

Este sistema utiliza *4* pliegues de validación cruzada, por lo que al entrenar este sistema se generan *4* modelos diferentes (1 para cada pliegue). Debido a esto, a la hora de evaluar el rendimiento del sistema (mediante las predicciones que realice cada modelo respecto a su conjunto de evaluación) hay que tener en cuenta los *4* modelos.

La base de datos con la que ha sido entrenado este sistema se trata de `TUT Sound events 2017` [104] [120] [121], la cual contiene dos subconjuntos: un subconjunto para el entrenamiento (conjunto de datos de entrenamiento y validación) y otro para evaluación (conjunto de datos de evaluación), que contienen *6* clases de eventos sonoros: `Chirrido al frenar` *`(brakes squeaking)`*`, coche` *`(car)`*`, niños` *`(children)`*`, camión` *`(large vehicle)`*`, gente hablando` *`(people speaking)`* `y gente caminando` *`(people walking)`*.

Esta base de datos esta compuesto de una serie de archivos de audio de escenas de calle que contienen sus archivos de etiquetado asociados, donde se encuentran registrados cada uno de los eventos pertenecientes a las *6* clases de eventos que se manifiestan en cada escena.

El subconjunto destinado al entrenamiento de la ANN contiene *4* distribuciones diferentes de datos de entrenamiento y datos de validación de los archivos de audio de escenas de calle disponibles (*1* distribución diferente para cada pliegue de validación cruzada). Para cada una de estas distribuciones, se utilizan todos los archivos de audio disponibles en este conjunto pero utilizando una diferente organización (hay archivos que en una distribución se encuentran dentro de los datos de entrenamiento y en otra distribución, estos archivos se encontraran como datos de validación).

El subconjunto de evaluación, por otra parte, consiste en un conjunto de archivos de audio de escenas de calle con las que se podrá evaluar el rendimiento de las predicciones de cada modelo (esta vez solo hay *1* distribución a diferencia del subconjunto destinado al entrenamiento) frente a datos nuevos que el sistema no ha utilizado durante su entrenamiento.

De todas las variantes que contiene este sistema a la hora de pre-procesar los archivos de audio se ha utilizado la versión que trabaja con archivos de audio mono (en el caso de que los archivos suministrados no lo sean, los convierte en mono) y por tanto, como método de extracción de características se utilizan espectrogramas de log-Mel para capturar los vectores de energía en las diferentes bandas de mel en escala logarítmica, también conocido este método como *mbe*.

4.3.2 Requisitos de ejecución

Para el funcionamiento de este sistema se requieren los archivos correspondientes de la base de datos `TUT Sound events 2017` y los diferentes scripts que utiliza este sistema para poder generar los *4* modelos de validación cruzada. Estos scripts se describirán a continuación.

4.3.2.1 Script `feature.py`

El script `feature.py` realiza el pre-procesamiento de los archivos de audio y su etiquetado del subconjunto de datos destinado al entrenamiento (datos de entrenamiento y datos de validación) de la ANN perteneciente a la base de datos con la que se trabaja (`TUT Sound events 2017`).

El pre-procesamiento que se realiza no es otro que el comentado para la extracción de características enfocada a entradas de audio de un solo canal (mono) en el apartado 3.2.14, donde se describe el sistema de referencia. Se generan espectogramas de log-Mel para cada archivo de audio con una ventana de Hamming de *40* ms de longitud y *40* bandas mel en el rango de frecuencia de *0-22500* Hz. Una vez generado los F espectogramas de log-Mel, para un archivo de audio procesado, las dimensiones de la matriz de datos del audio procesado de cada archivo de audio es de F *x 40*.

En este caso, a partir de estos archivos de audio y su etiquetado, se generan *4* divisiones (una para cada pliegue de validación cruzada del sistema). Para cada división, la base de datos se divide en dos partes diferentes: Una parte que contiene los datos con los que el sistema entrena cada modelo y otra parte orientada al testeo o validación de los modelos durante el entrenamiento. Cada división (entrenamiento + validación) contienen todos los datos de audio y etiquetado de la base de datos orientada al entrenamiento, pero para caso, se utiliza una repartición diferente de los datos entre las particiones de entrenamiento y validación.

Este script genera un archivo que contiene *4* matrices para cada división (*4* archivos en total), que serán los datos que toma como entrada la red:

- Una matriz X que contiene los archivos de audio procesados del conjunto de entrenamiento normalizados.
- Una matriz Y que contiene el etiquetado procesados de la matriz X.
- Una matriz X_{test} que contiene los archivos de audio procesados del conjunto de validación normalizados.
- Una matriz Y_{test} que contiene el etiquetado procesados de la matriz X_{test}.

Debido a que se trabaja con archivos de audio mono, se le asigna el valor True a la variable *is_mono* (que de forma predeterminada se encuentra en False) de este script para convertir a mono aquellos archivos que posean más de un canal de audio. Para este script hay que facilitar la ruta de la carpeta donde se encuentra el subconjunto destinado al entrenamiento de la base de datos `TUT Sound events 2017` (audios y etiquetado del conjunto de entrenamiento y validación) y la ruta donde se depositarán los archivos que contengan las *4* matrices de datos mencionadas anteriormente (*4* archivos en total, *1* para cada pliegue de validación cruzada).

4.3.2.2 Script `sed.py`

El script `sed.py` se encarga de realizar el entrenamiento de un sistema capaz de reconocer eventos acústicos en condiciones de fuentes múltiples mediante una CRNN, generando *4* modelos (uno por cada pliegue de validación cruzada). Para ello, toma como entradas los datos procesados de audio y su etiquetado pertenecientes a los conjuntos de entrenamiento y validación de la base de datos `TUT Sound events 2017`; procedentes de la ejecución del script `feature.py`. Al encargarse del entrenamiento de la red, es en `sed.py` donde se describe su arquitectura, concretamente en una de sus funciones llamada `get_model` (en la que se realizaran modificaciones para sistemas posteriores). La arquitectura de la red utilizada para este entrenamiento se describe en el apartado 3.2.14, donde se introduce el sistema de referencia.

Este script exporta el mejor modelo para cada pliegue de validación cruzada y una imagen que contiene *2* gráficas que muestran los valores de unas métricas determinadas para todas las épocas del entrenamiento: la primera gráfica muestra el valor de las pérdidas de entrenamiento y las pérdidas de validación; mientras que la segunda imagen muestra la puntuación media (F) y la tasa de error (ER) en segmentos de un segundo.

4.3.2.3 Otros scripts

Además de los scripts `feature.py` y `sed.py`, el sistema de referencia también contiene otros scripts necesarios para su funcionamiento:

- El script `metrics.py` es un archivo complementario que implementa las métricas centrales del módulo de evaluación de detección de eventos acústicos del sistema `Sed eval` descrito en el apartado 4.2.1.1.
- El script `utils.py` es un archivo complementario que tiene algunas funciones de utilidad.

También, para poder evaluar las predicciones que realizan los modelos generados que provienen de las *4* validaciones cruzadas sobre sus conjunto de evaluación, fue necesario crear otros *2* script:

Uno de estos scripts se trata de una versión modificada del script `feature.py`, llamado `feature_evaluacion_sistema_base.py`, que se encarga del pre-procesado de los archivos de audio del conjuntos de evaluación de la base de datos `TUT Sound events 2017` y sus etiquetas.

Los otros *2* scripts se encargan de generar predicciones utilizando los *4* modelos generados (cada uno procedente de uno de los pliegues de validación cruzada) sobre los datos del conjunto de evaluación para posteriormente almacenar las predicciones de cada modelo en un archivo de texto (1 para modelo) que contiene una lista de eventos detectadas, además de seleccionar el etiquetado real asignado en los datos del conjunto de evaluación (en lugar de las predicciones que realizan los modelos) y transformarlo en archivos de texto (1 para cada modelo) que contienen una lista de eventos de referencia. Estos archivos son: `convertidor_salida_evaluacion.py` y `convertidor_salida_evaluacion_NO_PREDICT.py`; y se encuentran descritos en el apartado 4.2.2.

4.3.3 Resultados del sistema de referencia

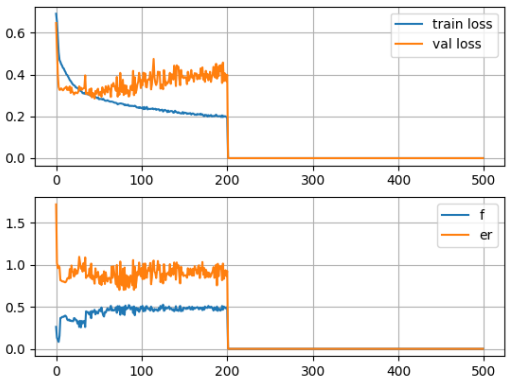
Durante el entrenamiento, las métricas que se miden y se tienen en cuenta para valor la mejoría de la red en las diferentes épocas son: `la tasa de error por segmento (ER)` y `la puntuación media (F) o F-score` calculada en segmentos de un segundo de duración utilizando micropromediado sobre el conjunto de validación promediado en cuatro pliegues. Los resultados obtenidos de estas métricas fueron de una *ER* de *0.593* y una *F* de *0.5852*; muy similares a los resultados proporcionados en la documentación de este sistema [7], los cuales, muestran una *ER* de *0.6* y una *F* de *0.57*.

En la figura 4.1 se pueden apreciar *2* gráficas para cada modelo de pliegue de validación cruzada que muestran las *pérdidas de entrenamiento y de validación*, al igual que *la puntuación media (F) y la tasa de error (ER) en segmentos de un segundo utilizando micropromediado del conjunto de validación* en las diferentes épocas del entrenamiento.

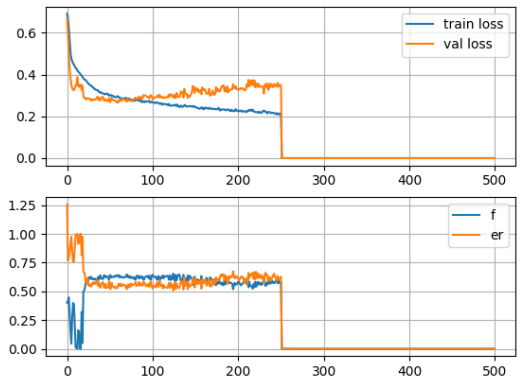
Como se ha comentado en el apartado 4.2.1.1 en el que se describe el sistema de evaluación utilizado (`Sed eval`), este sistema se encarga de evaluar el rendimiento de un archivos que contiene las predicciones realizadas por un modelo (estructuradas en forma de lista de eventos predichos) sobre un archivo que contiene una lista de eventos de referencia (que se corresponde con el etiquetado del conjunto de evaluación utilizado para realizar las predicciones) utilizado para esta tarea. El sistema de evaluación `Sed eval` ofrece como resultados las diferentes mediciones de varias métricas basadas en segmentos (de *1* segundo) o eventos, utilizando micro o macro promediado.

Las métricas que se utilizan durante el entrenamiento de este sistema son `la tasa de error por segmento (ER)` y `la puntuación media (F) o F-score` calculada en segmentos de un segundo de duración utilizando micropromediado, por lo que a estas métricas se les da más importancia que al resto a la hora de valorar si el sistema ofrece un buen rendimiento debido a que el sistema es entrenado optimizando estas métricas. En la tarea 3 del *challenge* DCASE 2017 [119] de IEEE AASP [91] (*challenge* donde se presentó el sistema de referencia [7]), los sistemas presentados se puntuaron teniendo en cuenta estas mismas métricas sobre el conjunto de evaluación de la base de datos `TUT Sound events 2017`.

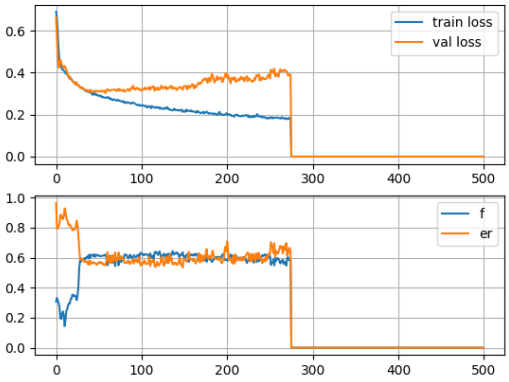
Para cada sistema utilizado en este capítulo de *Implementación* se extraen *2* tablas de resultados: una tabla que contiene `las métricas basadas en segmentos` y otra para `las métricas basadas en eventos` utilizando `micro` o `macro` promedio. Todas estas métricas se encuentran definidas en el apartado 4.2.1.4.



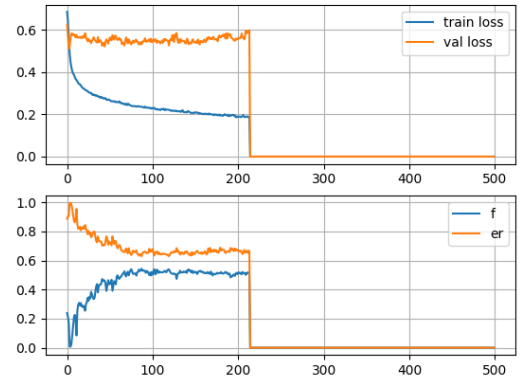
(a) Gráficas del entrenamiento del primer modelo de validación cruzada



(b) Gráficas del entrenamiento del segundo modelo de validación cruzada



(c) Gráficas del entrenamiento del tercer modelo de validación cruzada



(d) Gráficas del entrenamiento del cuarto modelo de validación cruzada

Figura 4.1: Gráficas de cada modelo de pliegue de validación cruzada del `sistema de referencia` que muestran las *pérdidas de entrenamiento y de validación*, al igual que *la puntuación media (F) y la tasa de error (ER) en segmentos de un segundo evaluadas sobre el conjunto de validación*

Métricas basadas en segmentos	
Longitud evaluada	6898.28 sg
Archivos evaluados	4
Longitud del segmento	1 sg
Métricas generales (micropromedio)	
Puntuación media (F1)	41.95 %
Precisión	44.60 %
Recuperación	39.59 %
Tasa de error (ER)	0.85
Tasa de sustitución	0.25
Tasa de eliminación	0.36
Tasa de inserción	0.24
Sensibilidad	39.59 %
Especificidad	90.85 %
Exactitud equilibrada	65.22 %
Exactitud	82.80 %
Métricas promedio por clase (macropromedio)	
Puntuación media (F1)	27.22 %
Precisión	39.37 %
Recuperación	28.91 %
Tasa de error (ER)	1.20
Tasa de eliminación	0.71
Tasa de inserción	0.49
Sensibilidad	28.91 %
Especificidad	90.04 %
Exactitud equilibrada	59.47 %
Exactitud	82.80 %

Tabla 4.1: Métricas basadas en segmentos del sistema de referencia

Dado que el sistema se ha entrenado enfocándose en las métricas basadas en segmentos comentadas y a pesar de que el sistema de evaluación proporciona también las métricas evaluadas basadas en eventos, solo se incluyen estas tablas cuando los resultados obtenidos para una implementación sean razonables.

Habiendo comprobado que los valores de las métricas durante el entrenamiento de este sistema son muy similares a los esperados, se realizan las predicciones sobre el conjunto de evaluación y se suministran al sistema de evaluación (`Sed eval`) un archivo con las predicciones del sistema y otro con las etiquetas verdaderas (de referencia) del conjunto de evaluación para cada modelo generado. En total hay *4* modelos del sistema, ya que utiliza *4* pliegues de validación cruzada (como se ha comentado en varias ocasiones).

Una vez evaluado las predicciones del sistema frente al conjunto de evaluación de la base de datos `TUT Sound events 2017`, se obtienen los resultados reflejados en las tablas 4.1 y 4.2.

Los resultados obtenidos para el sistema de referencia presentado en la tarea el *workshop* de la tarea 3 del *challenge* DCASE 2017 [119] de IEEE AASP [91], proporcionados en [158], muestran una *ER* y una *F* calculadas en segmentos de un segundo de duración utilizando micropromediado de *0.7914* y *41.7 %* respectivamente. Por otra arte, el sistema de referencia entrenado en este trabajo, tal y como se puede observar en la tabla 4.1 presenta unos resultados de *ER* y *F* calculadas en segmentos de un segundo de duración utilizando micropromediado de *0.85* y *41.95 %*.

Si comparamos los resultados obtenidos por el sistema de referencia presentado en el *challenge* DCASE y el entrenado para este trabajo podemos observar que hay cierta variación en ambos parámetros, siendo más destacada la variación en la métrica de *ER* en segmentos de un segundo de duración utilizando micropromediado. dado que se utilizó el mismo conjunto de datos para realizar la evaluación de este sistema (conjunto de evaluación de `TUT Sound events 2017`) y ambos sistemas presentaron métricas parecidas en los modelos generados durante el entrenamiento, se determinó que la discrepancia de ambos sistemas entre los resultados del conjunto de evaluación pueden deberse a limitaciones de hardware.

Métricas basadas en eventos	
Longitud evaluada	6898.28 sg
Archivos evaluados	4
Tiempo del collar	200 ms
Desplazamiento (longitud)	50 %
Métricas generales (micropromedio)	
Puntuación media (F1)	0.97 %
Precisión	0.74 %
Recuperación	1.38 %
Tasa de error (ER)	2.80
Tasa de sustitución	0.03
Tasa de eliminación	0.96
Tasa de inserción	1.82
Métricas promedio por clase (macropromedio)	
Puntuación media (F1)	0.51 %
Precisión	0.40 %
Recuperación	0.79 %
Tasa de error (ER)	3.00
Tasa de eliminación	0.99
Tasa de inserción	2.00

Tabla 4.2: Métricas basadas en eventos del sistema de referencia

A pesar de esta discrepancia entre resultados respecto al sistema presentado en la tarea 3 del *challenge* DCASE 2017 [119] de IEEE AASP [91], se puede concluir que para este objetivo los resultados son buenos debido a las limitaciones de la tarea, el número de datos que contiene la base de datos `TUT Sound events 2017` (bastante limitado) y de que para cada instante de tiempo la posibilidad de que se encuentre un evento activo perteneciente a una clase de evento sonoro a detectar es variable (pudiendo contener instantes de tiempo en donde no hay ningún evento activo, hasta instantes donde se encuentran varios de diferentes clases sonoras). Sin embargo, teniendo en cuenta que los resultados ideales de las métricas de *ER* y *F* calculadas en segmentos de un segundo de duración utilizando micropromediado son de *0* y *100 %* respectivamente, el rendimiento de este sistema enfocado a estas métricas en particular es bastante mejorable.

Otra de las observaciones que se pueden realizar, tal y como ocurre en el resto de sistema, consiste en que los resultados de las métricas basadas en eventos obtenidas en la tabla 4.2 son bastante peores en comparación a los resultados de las métricas basadas en segmentos debido a que durante el entrenamiento, los sistemas son entrenados para optimizar `la tasa de error por segmento (ER)` y `la puntuación media (F) o F-score` calculada en segmentos de un segundo de duración utilizando micropromediado.

A la hora de entrenar un sistema hay que tener ciertas consideraciones dependiendo de la utilidad que se le vaya a dar y enfocarse en perfeccionar algún tipo de métrica para ajustarse a las necesidades requeridas. En este caso, el sistema de referencias se encuentra optimizado para la detección de eventos en segmentos de un segundo utilizando micropromediado, lo cual puede ofrecer un buen porcentaje de acierto en las predicciones en general sobre un conjunto de datos totales pero no garantiza que funcione igual de bien para todas las clases. Como curiosidad, revisando los archivos de predicciones realizadas por los diferentes modelos de este sistema, se ha detectado que existe una gran dificultad para detectar las clases de eventos sonoros de `chirrido al frenar [brakes squeaking]` y `niños [children]` del conjunto de evaluación respecto al resto de eventos; pero al evaluar las métricas utilizando micropromediado, los resultados de las otras clases pueden compensar a los resultados de las clases mencionadas anteriormente.

4.4 Sistema de referencia evaluado con otra base de datos

4.4.1 Descripción y condiciones experimentales

La primera implementación que se realizó más allá de ejecutar y comprobar el correcto funcionamiento del sistema de referencia respecto a los resultados [158] que se muestran en la tarea 3 del *challenge* DCASE 2017 [119] (donde se presentó el sistema e referencia) consiste en evaluar el sistema de referencia utilizando otra base de datos diferente a la que se utiliza de forma predeterminada, con el fin de comprobar que tan eficaz podría resultar este sistema al detectar sonidos (utilizando las mismas clases de eventos sonoros) procedentes de otras bases de datos diferentes.

La base de datos que se utilizó para ello es una mezcla de archivos de varias bases de datos mencionadas en el apartado 3.3, en donde, para cada base de datos seleccionada, se han escogido los archivos de audio pertenecientes a una única clase de evento sonoro (perteneciente a una de las clases de eventos sonoro con las que el sistema fue entrenado). La intención inicial era utilizar el mismo número de clases de eventos sonoros que contiene la base de datos (`TUT Sound events 2017` [104]) que utiliza el sistema de referencia para entrenarse, pero finalmente no pudo ser así y solo se utilizaron *4* clases de eventos sonoros de las *6* que se usaron para su entrenamiento. Las *2* clases de eventos sonoros que no se tuvieron en cuenta son: chirrido al frenar *(brakes squeaking)* y camión *(large vehicle)*. Esto se debe a la dificultad de encontrar archivos de audios de estas clases en otras bases de datos con una calidad aceptable y que no tuvieran la posibilidad de encontrarse superpuestos con otros eventos sonoros no clasificados temporalmente por nuestro sistema entrenado.

Las bases de datos seleccionadas son:

- La base de datos `FSDKaggle2019` con los archivos de audio de la clase de evento sonoro coche *(car)*. Consta de *88* clips de audio de evento único con una duración de *0.3* a *30* segundos, extraídos de la parte del conjunto de entrenamiento que consta de datos etiquetados manualmente. Estos archivos de audio se situaron en una carpeta llamada `FSDKaggle2019_coche` dentro de la carpeta `Testeo` creada en el directorio que contiene las bases de datos.
- La base de datos `UrbanSound8K` con archivos de audio de la clase de evento sonoro niños *(children)*. Consta de *100* clips de audio de evento único con una duración aproximada de *4* segundos, extraídos de la primera carpeta de la base de datos (carpeta `fold1`). Estos archivos de audio se situaron en una carpeta llamada `UrbanSound8K_ninos` dentro de la carpeta `Testeo` creada en el directorio que contiene las bases de datos.
- La base de datos `IEEE DCASE 2016 Challenge: Task 2 Datasets` con archivos de audio de la clase de evento sonoro gente hablando *(people speaking)*. Consta de *20* clips de audio de evento único con una duración aproximada de *2* segundos, extraídos del subconjunto de entrenamiento. Estos archivos de audio se situaron en una carpeta llamada `dcase2016_habla` dentro de la carpeta `Testeo` creada en el directorio que contiene las bases de datos.
- La base de datos `ESC-50: Dataset for Environmental Sound Classification` con los archivos de audio de la clase de evento sonoro gente caminando *(people walking)*. Consta de *40* clips de audio de evento único con una duración aproximada de *5* segundos. Estos archivos de audio se situaron en una carpeta llamada `ESC-50master-pasos` dentro de la carpeta `Testeo` creada en el directorio que contiene las bases de datos.

Para organizar esta nueva base de datos, de las bases de datos recién mencionadas (`FSDKaggle2019, UrbanSound8K, IEEE DCASE 2016 Challenge: Task 2 Datasets`

y ESC-50: Dataset for Environmental Sound Classification), se extrajeron los archivos de audio de las *4* clases de eventos sonoros seleccionados y se ubicaron dentro de una nueva carpeta (llamada Testeo) creada en el directorio que contiene las bases de datos. A continuación, dentro de esta nueva carpeta se crean otras *4* carpetas nuevas, una para los archivos de cada clase de evento sonoro: coche *(car)*, niños *(children)*, gente hablando *(people speaking)* y gente caminando *(people walking)*.

Una vez habiendo organizado los archivos de las clases de eventos sonoros por carpetas, se realizan una serie de procesos para convertir estos archivos en el conjunto de datos de evaluación con el que se evaluarán nuevamente los *4* modelos del sistema de referencia previamente entrenados (entrenados con el sistema de referencia).

Como se ha comentado en el apartado 3.3.5, cada conjunto de datos (entrenamiento, validación y evaluación) de la base de datos TUT Sound events 2017 contiene varios archivos de audio y sus archivos de etiquetado. Estos archivos de audio consisten en grabaciones de una escena acústica (calle) donde se pueden distinguir sonidos de varias clases de eventos sonoros, pudiendo tener eventos solapados temporalmente en algunos instantes de tiempo.

Sin embargo, la distribución de los datos de esta nueva base de datos es diferente a la del conjunto de evaluación de la base de datos TUT Sound events 2017, ya que en principio solo contiene archivos (clips) de audio de evento único de cada clase de evento sonoro. Este motivo lleva a crear un nuevo script llamado feature_sistema_testeo.py, creado a partir del script feature.py (descrito en 4.3.2.1), que se encarga de todo el proceso de transformación de esta base de datos en *4* archivos (uno para cada modelo de pliegue de validación cruzada) que contengan una matriz de datos que represente todos los clips de audio procesados concatenados y otra matriz de datos con su etiquetado procesado.

Dado que los archivos de audio consisten en clips de audio y se puede conocer la duración de estos, al igual que la clase de evento sonoro a la que pertenece cada archivo (gracias a su ubicación en una carpeta u otra)... la generación del etiquetado de cada archivo es bastante sencillo y es una de las primeras tareas que realiza el script feature_sistema_testeo.py. Por tanto, para cada archivo de audio, se crea un archivo de texto (con el mismo nombre que el archivo de audio) que contiene su etiquetado; situando estos archivos en la carpeta etiquetas, que se encuentra ubicada en la carpeta raíz de esta base de datos.

Después de generar el etiquetado de los archivos de audio, el script feature_sistema_testeo.py realiza el pre-procesamiento de los archivos de audio. Este pre-procesamiento consiste en remuestrear los diferentes archivos de audio a *44.1* KHz (en el caso en el que no se encuentren muestreos a esa frecuencia), convertir todos los audios a mono y realizar espectogramas de log-Mel (llamada a esta técnica como *mbe* en la documentación del sistema de referencia) con las mismas características que utiliza en el script feature.py. Justo después de procesar cada archivo de audio se procesa su archivo de etiquetado asociado (transformándolo en un vector que indica mediante *1* o *0* que evento se encuentra activo o inactivo durante cada instante del archivo de audio), para posteriormente almacenar el audio y su etiquetado procesado de cada archivo de audio como dos matrices de datos diferentes en un nuevo archivo con el mismo nombre que el archivo de audio original dentro de la carpeta feat, que se encuentra ubicada en la carpeta raíz de esta base de datos.

El siguiente paso consiste en concatenar las diferentes matrices de audio procesado y sus correspondientes matrices de etiquetado (cada conjunto de matrices por separado otra parte) siguiendo un orden aleatorio. Esto se realiza para agrupar todas las matrices de audio procesado en una única matriz (el mismo procedimiento para las matrices de etiquetado), simulando que se trata de una escena acústica donde un evento ocurre justo después de otro de manera consecutiva. A diferencia del conjunto de eva-

luación de la base de datos TUT Sound events 2017, en este caso no hay eventos activos solapados temporalmente.

El último paso que realiza el script feature_sistema_testeo.py consiste en normalizar los datos y exportarlos. Los datos de audio procesados y de etiquetado de este conjunto de evaluación se almacenan como *2* matrices en nuevo archivo en la carpeta de Datos_finales, que se encuentra dentro de la carpeta raíz de esta base de datos. Esta exportación se realiza *4* veces para generar *4* archivos con estas matrices, debido a que utilizamos *4* modelos de validación cruzada del sistema y por comodidad, ya que de esta manera, se puede hacer uso los scripts de convertidor_salida_evaluacion.py y convertidor_salida_evaluacion_NO_PREDICT.py (utilizados para generar los archivos de predicciones y de referencia con lo que se evalúan las métricas del sistema sobre la base de evaluación) cambiando solamente la ruta de la carpeta donde se encuentran los archivos que contienen las matrices de datos y etiquetado procesado de este nuevo conjunto de evaluación.

Por tanto, el siguiente paso se trata de transformar estos archivos de matrices de audio/etiquetado procesado obtenidos de este nuevo conjunto de evaluación procedente de la nueva base de datos utilizando los scripts convertidor_salida_evaluacion.py y convertidor_salida_evaluacion_NO_PREDICT.py para generar los archivos de predicciones y de etiquetado real (archivos de referencia) de lista de eventos para cada modelo de pliegue de validación cruzada del sistema (tal y como se hace con el conjunto de evaluación de la base de datos TUT Sound events 2017).

Finalmente, se ejecuta el sistema de evaluación (Sed eval) utilizando estos archivos de listas de eventos para evaluar las diferentes métricas de las predicciones realizadas.

4.4.2 Resultados del sistema de referencia evaluado con otra base de datos

Los resultados obtenidos de las métricas al evaluar las predicciones generadas por los distintos modelos del sistema de referencia sobre esta nueva base de datos resultaron ser los que marcan la tabla 4.3.

Los resultados obtenidos tras evaluar el sistema de referencia con esta nueva base de datos muestran una *ER* y una *F* calculadas en segmentos de un segundo de duración utilizando micropromediado de *0.83* y *46.72 %* respectivamente. Si los comparamos con los resultados obtenidos al evaluar el sistema de referencia con el conjunto de evaluación de la base de datos que se utiliza de forma predeterminada (TUT Sound events 2017), se puede apreciar que con esta nueva base de datos se produce una mejora tanto en la *ER* (*0.85*) como en la *F* (*41.95 %*) calculadas en segmentos de un segundo de duración utilizando micropromediado. Ahora bien, esta mejora en estas métricas es bastante relativa y realmente no quiere decir que el sistema detecta mejor eventos sonoros en otras bases de datos diferentes a la que se utilizó para su entrenamiento.

Como se ha descrito en el aparatado 4.3.3, en los diferentes modelos de este sistema, se ha detectado que existe una gran dificultad para detectar las clases de eventos sonoros de chirrido al frenar *[brakes squeaking]* y niños *[children]* del conjunto de evaluación respecto al resto de eventos. Casualmente, la nueva base de datos utilizada para obtener estas métricas no contiene eventos pertenecientes la clase de evento sonoro chirrido al frenar *[brakes squeaking]*, una de las clases para las que el sistema tiene grandes dificultades de identificar, por lo que, este motivo puede ser más que suficiente como para que se aprecie una mejoría en los resultados de las métricas del sistema.

Con estos datos, se podría llegar a intuir que el rendimiento al utilizar los modelos generados del sistema de referencia para predecir las clases de eventos sonoros en otras bases de datos puede llegar a ser parejo al obtenido sobre las predicciones realizadas con el conjunto de evaluación de la base de datos TUT

Métricas basadas en segmentos	
Longitud evaluada	8036.61 sg
Archivos evaluados	4
Longitud del segmento	1 sg
Métricas generales (micropromedio)	
Puntuación media (F1)	46.72 %
Precisión	41.96 %
Recuperación	52.70 %
Tasa de error (ER)	0.83
Tasa de sustitución	0.38
Tasa de eliminación	0.10
Tasa de inserción	0.35
Sensibilidad	52.70 %
Especificidad	84.02 %
Exactitud equilibrada	68.36 %
Exactitud	78.38 %
Métricas promedio por clase (macropromedio)	
Puntuación media (F1)	37.91 %
Precisión	25.48 %
Recuperación	57.96 %
Tasa de error (ER)	ND
Tasa de eliminación	0.28
Tasa de inserción	ND
Sensibilidad	57.96 %
Especificidad	80.63 %
Exactitud equilibrada	59.63 %
Exactitud	78.38 %

Tabla 4.3: Métricas basadas en segmentos del sistema de referencia evaluado con otra base de datos

`Sound events 2017`, o al menos, el rendimiento al predecir las clases de eventos sonoros utilizadas en la nueva base de datos. Sin embargo, no se puede dar un veredicto final sobre ello, puesto que harían falta realizar más evaluaciones con otros conjuntos de evaluación que contengan una buena cantidad de datos de las *6* clases de eventos sonoros que el sistema es capaz de detectar.

Por otro lado, en la tabla 4.3 se puede observar que la tasa de inserción y la tasa de error (ER) para las métricas basadas en segmentos utilizando macropromedio no se encuentran disponibles debido a que la base de datos utilizada para la evaluación de las predicciones de los modelos del sistema de referencia no contiene muestras de todas las clases de eventos sonoros. A pesar de que el sistema muestre el resto de valores de las métricas cuando se utiliza macropromedio, la validez de estos resultados es nula debido a que pondera los resultados (o ausencia de ellos) de las métricas individuales asociadas a cada clase con la misma relevancia, incluyendo de esta forma, las clases no utilizadas en este conjunto de evaluación.

4.5 Sistema de referencia entrenado con otra base de datos (sistema urban)

4.5.1 Descripción y condiciones experimentales

Una de las propuestas iniciales para este trabajo consiste en entrenar el sistema de referencia utilizando otras clases de eventos sonoros pertenecientes a otras bases de datos diferentes a la que se usa de forma predeterminada, realizando una ampliación mínima del número de clases de eventos sonoros para las que este nuevo sistema sea capaz de detectar. Dado que la propuesta era utilizar un número de clases superior al de la base de datos que el sistema de referencia es capaz de identificar pero sin realizar una variación muy elevada, se decidió fijar el número de clases de eventos sonoros para este nuevo sistema en *8* frente a las *6* de la base de datos que se utiliza de forma predeterminada (`TUT Sound events 2017`).

Tras analizar los diferentes eventos sonoros disponibles en las bases de datos recopiladas y las características de estos dentro de las distintas bases de datos se decidió utilizar *8* de las *10* clases de eventos sonoros pertenecientes a la base de datos `Urbansound8k` (descrita en el apartado 3.3.6), de ahí el apodo de este nuevo sistema como `sistema urban`. Esta base de datos esta formada por *8732* archivos de clips de audio de eventos único (en cada clip solo hay un evento activo) de *4* segundos o menos de duración pertenecientes a *10* clases de eventos sonoros de sonidos urbanos. Las *8* clases de eventos sonoros escogidas de esta base de datos son: `aire acondicionado`, `bocina de automóvil`, `taladro`, `ladrido de perro`, `motor al ralentí`, `disparo de arma`, `martillo neumático`, y `sirena`.

Las principales razones por las que se seleccionaron estas clases de eventos sonoros pertenecientes a esta base de datos son:

- La gran cantidad de clips de audio que posee cada clase de evento sonoro en comparación a la mayoría de las otras clases pertenecientes a las otras bases de datos disponibles (en el apartado 3.3). Todas las clases de eventos sonoros no poseen el mismo número de archivos de audio, pero si contienen los necesarios para poder crear unos conjuntos de entrenamiento, validación y evaluación que contengan datos suficientes de cada clase.

- La buena distinción entre clases de eventos sonoros pertenecientes a una misma temática. Las clases de eventos sonoros de esta base de datos pertenecen a una temática urbana (una temática donde pueden encontrarse una gran variedad de eventos, algunos más similares que otros) y la distinción desde el punto de vista auditivo de una persona promedio entre cada clase es bastante notoria (no hay clases confusas como pueden ser, por ejemplo, las clases: camión pasando y coche pasando).

- Los clips de audio de esta base de datos solo contienen un evento activo en cada clip, de manera que no contiene eventos activos de distintas clases solapados temporalmente. Esto es muy importante si se quiere utilizar solo los archivos de algunas clases de eventos sonoros en especifico, en lugar tener que utilizar todos los archivos de la base de datos completa (como es este caso). Si los archivos poseen solapamiento de eventos para un instante de tiempo; en el caso de seleccionar solo algunas clases de eventos sonoros de la base de datos para volver a entrenar el sistema de referencia... estos archivos contendrían sonidos no deseados de otras clases de eventos sonoros no etiquetados, y por tanto, es probable que el sistema se entre incorrectamente.

- Las clases de eventos sonoros `niños jugando` y `música callejera`pertenecientes a la base de datos `Urbansound8k`, fueron descartadas debido a que la clase de evento sonoro `niños jugando` es muy similar a la clase de evento sonoro `niños [children]` perteneciente a la base de datos `TUT Sound events 2017` utilizada en el sistema de referencia; mientras que la clase `música callejera [street music]` se descartó porque se consideró la clase que podía dar lugar a más confusiones, ya que, dentro del concepto de música hay una gran variedad de sonidos que distan mucho unos de otros.

Inicialmente, los archivos de audio de la base de datos `UrbanSound8k` se encuentran repartidos en *10* pliegues (carpetas denominadas `fold1-fold10`) que contienen el mismo número de archivos de cada clase de evento sonoro. Para utilizar más cómodamente todos los archivos de las clases de eventos sonoros seleccionadas, se dispuso a cambiar esta distribución, reorganizando nuevamente los archivos de audio, agrupando los archivos de cada clase de evento sonoro por separados en nuevas carpetas dentro de una nueva carpeta llamada `Urban` creada en el directorio que contiene las bases de datos.

El número de archivos de audio y su ubicación para cada clase de evento sonoro es la siguiente:

- La clase de evento sonoro `aire acondicionado [air conditioner]` consta de *1000* archivos de audio que se situaron en una carpeta llamada `air_conditioner` dentro de la carpeta `Urban` creada en el directorio que contiene las bases de datos.
- La clase de evento sonoro `bocina de automóvil [car horn]` consta de *429* archivos de audio que se situaron en una carpeta llamada *car_horn* dentro de la carpeta `Urban` creada en el directorio que contiene las bases de datos.
- La clase de evento sonoro `taladro [drilling]` consta de *1000* archivos de audio que se situaron en una carpeta llamada *drilling* dentro de la carpeta `Urban` creada en el directorio que contiene las bases de datos.
- La clase de evento sonoro `ladrido de perro [dog bark]` consta de *1000* archivos de audio que se situaron en una carpeta llamada *dog_bark* dentro de la carpeta `Urban` creada en el directorio que contiene las bases de datos.
- La clase de evento sonoro `motor al ralentí [engine idling]` consta de *1000* archivos de audio que se situaron en una carpeta llamada *engine_idling* dentro de la carpeta `Urban` creada en el directorio que contiene las bases de datos.
- La clase de evento sonoro `disparo de arma [gun shot]` consta de *374* archivos de audio que se situaron en una carpeta llamada *gun_shot* dentro de la carpeta `Urban` creada en el directorio que contiene las bases de datos.
- La clase de evento sonoro `martillo neumático [jackhammer]` consta de *1000* archivos de audio que se situaron en una carpeta llamada *jackhammer* dentro de la carpeta `Urban` creada en el directorio que contiene las bases de datos.
- La clase de evento sonoro `sirena [siren]` consta de *929* archivos de audio que se situaron en una carpeta llamada *siren* dentro de la carpeta `Urban` creada en el directorio que contiene las bases de datos.

El script `feature_urban.py` que se utiliza para realizar el pre-procesado de los diferentes archivos de audio esta basado en el script `feature.py` que se utiliza para el pre-procesado de audio del sistema de referencia (comentado en el apartado 4.3.2.1) y en el script `feature_sistema_testeo.py` utilizado para el pre-procesamiento de los diferentes archivos de audio de un nuevo conjunto de evaluación utilizado para evaluar el sistema de referencia con otra base de datos (comentado en el apartado 4.4).

La generación del etiquetado de cada archivo es bastante sencillo y es una de las primeras tareas que realiza el script `feature_urban.py`, puesto que los archivos de audio consisten en clips de audio y se puede conocer la duración de estos, al igual que la clase de evento sonoro a la que pertenece cada archivo (gracias a su ubicación en una carpeta u otra). Para cada archivo de audio, se crea un archivo de texto (con el mismo nombre que el archivo de audio) que contiene su etiquetado; situando estos archivos en la carpeta `etiquetas`, que se encuentra ubicada en la carpeta raíz de esta base de datos (carpeta `Urban` creada en el directorio que contiene las bases de datos.).

Después de generar el etiquetado de los archivos de audio, el script `feature_sistema_testeo.py` realiza el pre-procesamiento de los archivos de audio. Este pre-procesamiento consiste en remuestrear los diferentes archivos de audio a *22.05* KHz (en el caso en el que no se encuentren muestreos a esa frecuencia), convertir todos los audios a mono y realizar espectogramas de log-Mel (llamada a esta técnica como *mbe* en la documentación del sistema de referencia) con las mismas características que utiliza en el script `feature.py` (utilizando en para generar los espectogramas de log-Mel una ventana de Hamming de *40*

ms de longitud y *40* bandas mel en el rango de frecuencia de *0-22500* Hz). El remuestreo a *22.05* KHz en lugar de a *44.1* KHz como se realiza en el sistema de referencia se debe a que la frecuencia de muestreo, la profundidad de bits y el número de canales no se encuentran estandarizados con los mismos valores para cada archivo en este conjunto, ya que conserva los archivos originales extraídos desde `Freesound` [86] y, por lo tanto, estos valores pueden variar de un archivo a otro, conteniendo archivos de audio con una frecuencia de muestreo bastante inferior a *44.1* KHz, por lo que se escoge el valor de *22.05* KHz.

Justo después de procesar cada archivo de audio se procesa su archivo de etiquetado asociado (transformándolo en un vector que indica mediante *1* o *0* que evento se encuentra activo o inactivo durante cada instante del archivo de audio), para posteriormente almacenar el audio y su etiquetado procesado de cada archivo de audio como dos matrices de datos diferentes en un nuevo archivo con el mismo nombre que el archivo de audio original (añadiendo al nombre del archivo original el sufijo_mon) dentro de la carpeta `feat`, que se encuentra ubicada en la carpeta raíz de esta base de datos.

Sobre el número total de eventos de audio registrados (que coincide con el número total de archivos de audio en este caso), se realizan *5* agrupaciones independientes que contienen cada una el *20 %* de los eventos de audio totales, donde, no se repite un mismo evento asociado a un archivo de audio de una partición a otra y todas las particiones contienen el mismo número de eventos de cada clase de evento sonoro. En cada agrupación, se concatena un *20 %* de los datos de audio procesado y las etiquetas de estos audio (por separado) de cada clase de evento sonoro consecutivamente de forma aleatoria sin repetir un mismo evento asociado a un archivo de audio y sin contener eventos solapados temporalmente.

Estas agrupaciones se forman para seleccionar datos diferentes destinados al conjunto de entrenamiento, validación y evaluación en cada uno de los *4* pliegues de validación cruzada utilizados para generar los *4* modelos durante el entrenamiento. La distribución de los datos para cada pliegue consiste en asignar un *60 %* del número de eventos totales de la base de datos al conjunto de entrenamiento (*3* de las *5* agrupaciones creadas), un *20 %* al conjunto de validación (*1* de las *5* agrupaciones creadas) y un *20 %* al conjunto de evaluación (*1* de las *5* agrupaciones creadas), seleccionando agrupaciones diferentes para cada conjunto de cada pliegue de validación cruzada.

Una vez habiendo seleccionado que datos conforman los diferentes conjuntos, se normalizan y se exportan como un archivo que contiene *3* matrices de audio procesado (*1* matriz para cada conjunto: entrenamiento, validación y evaluación) y *3* matrices de etiquetado procesado de cada una de las matrices de audio procesado para pligue de validación cruzada. Estos archivos se exportan dentro de la carpeta `Datos_finales`, situada dentro de la carpeta `Urban` creada en el directorio que contiene las bases de datos.

En cuanto al entrenamiento de la CRNN, el script que se utiliza para llevar a cabo esa tarea es una versión del script `sed.py` descrito en el apartado 4.3.2.2 con una ligera modificación llamado `sed_urban.py`. El cambio que se realiza respecto al script original (`sed.py`), además de cambiar la ruta de la carpeta donde se encuentran los archivos de los conjuntos de entrenamiento y validación para cada pligue de validación cruza, se trata de la modificación del valor de la variable *sr* que representa el valor de la frecuencia de muestreo en *Hz* de *44100* a *22050*, lo que afecta al número de frames en un segundo. Respecto a la arquitectura de la red original no se producen cambios más alla de que el tamaño de la salida de la red es modificado, contando con *8* neuronas en su última capa en lugar de *6* debido a que la red se adapta para detectar *8* clases de eventos sonoros en lugar de los *6* del sistema original. Este número de neuronas en la última capa se modifica automáticamente debido a las dimensiones de las matrices que contienen el etiquetado del conjunto de entenamiento suministrado al sistema.

Una vez realizado el entrenamiento de la red, se obtienen *4* modelos del sistema de diferentes pliegues de validaión cruzada. Con estos modelos generados se realizan las predicciones sobre el conjunto de

evaluación de la base de datos para generar los archivos de listas de eventos de predicciones que se le suministra al sistema de evaluación (`Sed eval`).

El archivo de lista de eventos de predicciones correspondiente a cada uno de los *4* modelos se genera ejecutando el script `convertidor_salida_evaluacion_sist_urban.py`, el cual, es una modificación el script `convertidor_salida_evaluacion.py` que se utiliza para generar estos archivos de lista de eventos de predicciones sobre un conjunto de evaluación cuando se trabaja con modelos procedentes del sistema de referencia o similares, capaces de detectar las *6* clases de eventos sonoros procedentes de la base de datos `TUT Sound events 2017`, en lugar de las *8* clases utilizadas en el sistema urban. Del mismo modo, para generar los archivos de lista de eventos de referencia (etiquetado verdadero) de cada conjunto de evaluación que sobre el que realiza las predicciones cada modelo, se generó el script `convertidor_salida_evaluacion_NO_PREDICT_sist_urban.py`, que cumple la misma función que el script `convertidor_salida_evaluacion_NO_PREDICT.py` (utilizado con modelos procedentes del sistema de referencia o similares con la capacidad de detectar las *6* clases de eventos sonoros procedentes de la base de datos `TUT Sound events 2017`) cuando se trabaja con modelos entrenados con las *8* clases utilizadas en el sistema urban.

Durante este trabajo, a la hora de generar los archivos de listas de eventos de predicciones y de referencia para los modelos generados a partir del sistema urban (modelos entrenados con las *8* clases mencionadas anteriormente en este apartado) y sus modificaciones en la arquitectura se utilizan los script: `convertidor_salida_evaluacion_sist_urban.py` y `convertidor_salida_evaluacion_NO_PREDICT_sist_urban.py`.

Una vez generado estos archivos para cada uno de *4* los modelos generados, se ejecuta el sistema de evaluación (`Sed eval`) utilizando estos archivos de listas de eventos para evaluar las diferentes métricas de las predicciones realizadas.

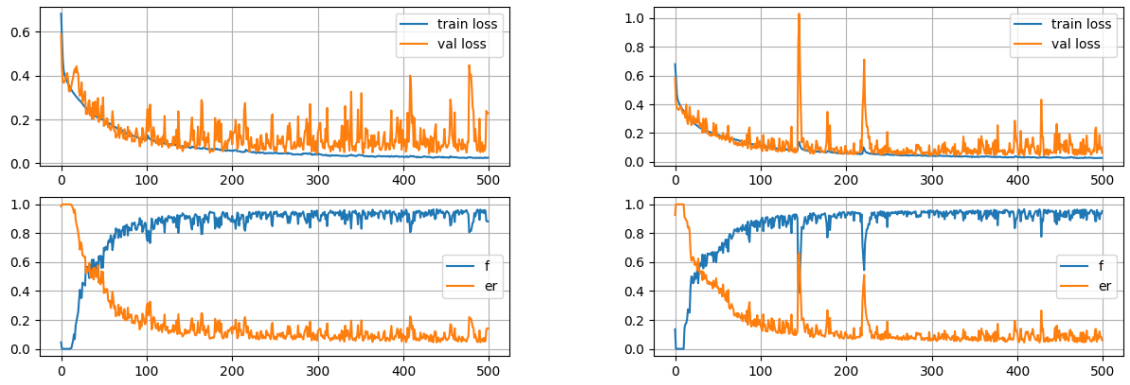
4.5.2 Resultados del sistema urban

Durante el entrenamiento, las métricas que se miden y se tienen en cuenta para valor la mejoría de la red en las diferentes épocas son: `la tasa de error por segmento (`*ER*`)` y `la puntuación media (`*F*`) o F-score` calculada en segmentos de un segundo de duración utilizando micropromediado sobre el conjunto de validación promediado en cuatro pliegues. Los resultados obtenidos de estas métricas fueron de una *ER* de *0.042* y una *F* de *0.9679*; valores bastante buenos y muy distantes a los obtenidos en el sistema de referencia mostrados en el apartado 4.3.3, los cuales, muestran una *ER* de *0.593* y una *F* de *0.5852*.

En la figura 4.2 se pueden apreciar *2* gráficas para cada modelo de pliegue de validación cruzada que muestran las *pérdidas de entrenamiento y de validación*, al igual que *la puntuación media (F) y la tasa de error (ER) en segmentos de un segundo utilizando micropromediado del conjunto de validación* en las diferentes épocas del entrenamiento.

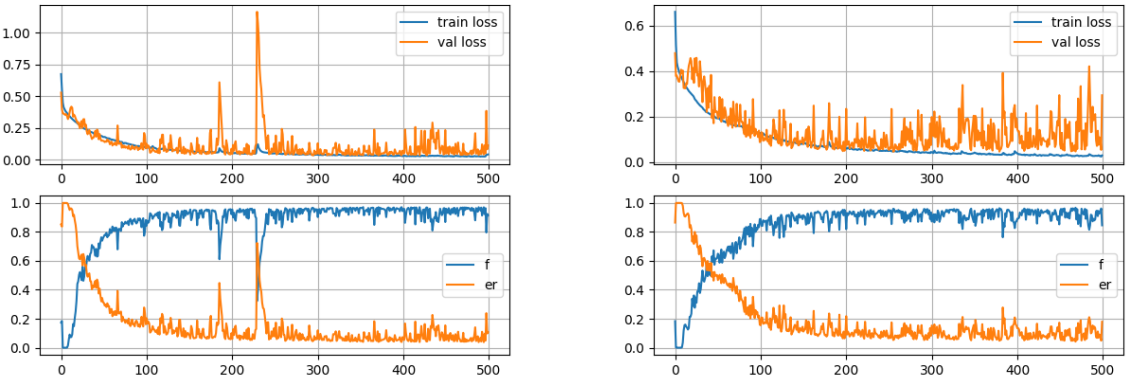
Debido a los buenos resultados obtenidos de las métricas medidas durante el entrenamiento, en un principio se intuyó que podría existir algún tipo de sobreajuste respecto al conjunto de evaluación, pero al evaluar las predicciones de cada modelo sobre sus conjuntos de evaluación y comprobar su correcto funcionamiento, se descartó la posibilidad. Los resultados de las métricas obtenidas de la evaluación de los archivos de listas de eventos sobre los archivos de listas de eventos de referencia se muestran en las tablas 4.4 y 4.5.

Al igual que se ha realizado con el resto de sistemas evaluados, los resultados de este sistema se reflejan mediante *2* tablas de métricas: `basadas en segmentos` y `basadas en eventos`, evaluadas



(a) Gráficas del entrenamiento del primer modelo de validación cruzada

(b) Gráficas del entrenamiento del segundo modelo de validación cruzada



(c) Gráficas del entrenamiento del tercer modelo de validación cruzada

(d) Gráficas del entrenamiento del cuarto modelo de validación cruzada

Figura 4.2: Gráficas de cada modelo de pliegue de validación cruzada del `sistema urban` que muestran las *pérdidas de entrenamiento y de validación*, al igual que *la puntuación media (F) y la tasa de error (ER) en segmentos de un segundo evaluadas sobre el conjunto de validación*

Métricas basadas en segmentos	
Longitud evaluada	18997.82 sg
Archivos evaluados	4
Longitud del segmento	1 sg
Métricas generales (micropromedio)	
Puntuación media (F1)	96.51 %
Precisión	96.10 %
Recuperación	96.93 %
Tasa de error (ER)	0.05
Tasa de sustitución	0.02
Tasa de eliminación	0.01
Tasa de inserción	0.02
Sensibilidad	96.93 %
Especificidad	99.29 %
Exactitud equilibrada	98.11 %
Exactitud	98.92 %
Métricas promedio por clase (macropromedio)	
Puntuación media (F1)	96.20 %
Precisión	96.06 %
Recuperación	96.37 %
Tasa de error (ER)	0.08
Tasa de eliminación	0.04
Tasa de inserción	0.04
Sensibilidad	96.37 %
Especificidad	99.26 %
Exactitud equilibrada	97.82 %
Exactitud	98.92 %

Tabla 4.4: Métricas basadas en segmentos del sistema urban

Métricas basadas en eventos	
Longitud evaluada	18997.82 sg
Archivos evaluados	4
Tiempo del collar	200 ms
Desplazamiento (longitud)	50 %
Métricas generales (micropromedio)	
Puntuación media (F1)	85.78 %
Precisión	81.31 %
Recuperación	90.76 %
Tasa de error (ER)	0.29
Tasa de sustitución	0.02
Tasa de eliminación	0.08
Tasa de inserción	0.19
Métricas promedio por clase (macropromedio)	
Puntuación media (F1)	85.54 %
Precisión	81.45 %
Recuperación	90.16 %
Tasa de error (ER)	0.31
Tasa de eliminación	0.10
Tasa de inserción	0.21

Tabla 4.5: Métricas basadas en eventos del sistema urban

utilizando `micro` y `macro` promediado. Como las métricas optimizadas durante el entrenamiento son la puntuación media (F) y la tasa de error (ER) en segmentos de un segundo utilizando micropromediado, para valorar el comportamiento de un sistema y la mejoría de un sistema respecto a otro se va hacer gran hincapié en estas dos métricas, teniendo en cuenta estos valores, comentando el resto de métricas en el caso de obtener algunos valores anómalos o bastante distantes del rendimiento esperado, dando en segundo lugar más importancia al resto de métricas basadas en segmentos utilizando micropromediado.

Como resultados en este sistema se puede observar que se obtuvo una *ER* de *0.05* y una *F* de *96.51 %* basadas en segmentos de *1* segundo utilizando micropromediado, muy parecidos a los resultados obtenidos sobre el conjunto de validación durante el entrenamiento de la red. Para tratarse de un sistema de detección, las métricas obtenidas son bastante buenas y más aun si las comparamos con las obtenidas en el sistema de referencia del apartado 4.3.3, aunque esto al final, tiene su explicación... Este sistema a diferencia del sistema de referencia, como se ha comentado, el número de clases de eventos sonoros utilizados no es muy superior (contando unicamente con *2* clases más), el número de datos utilizado durante su entrenamiento es ampliamente superior y en sus datos de entrenamiento, validación y evaluación solo existe una clase de evento sonoro activa para cada instante de tiempo. Este conjunto de razones provoca la dramática variación entre los resultados de las métricas de un sistema y otro.

Por lo que se puede observar en las métricas basadas en segmentos utilizando macropromedio, los valores de *ER* y de *F* presentan una ligera variación respecto a los valores utilizando micropromedio, lo que da a entender que hay clases que se detectan mejor que otras, hecho inevitable al entrenar un sistema utilizando micropromedios; pero aun así, los resultados son muy similares y bastante buenos. En cuanto a los resultados de las métricas basadas en eventos, tanto para las que utilizan micro o macro promedio, el sistema ofrece unos buenos resultados, aunque algo menos acertados que los obtenidos en las métricas basadas en segmentos.

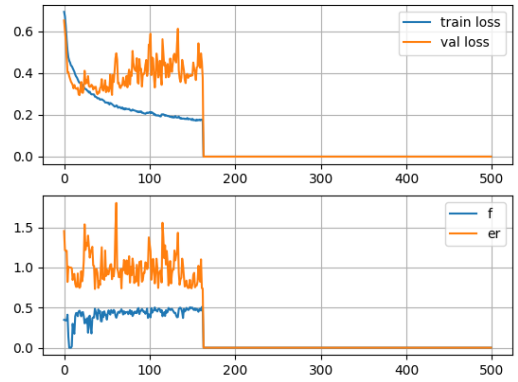
4.6 Estudio del impacto del incremento de capas convolucionales

4.6.1 Descripción y condiciones experimentales

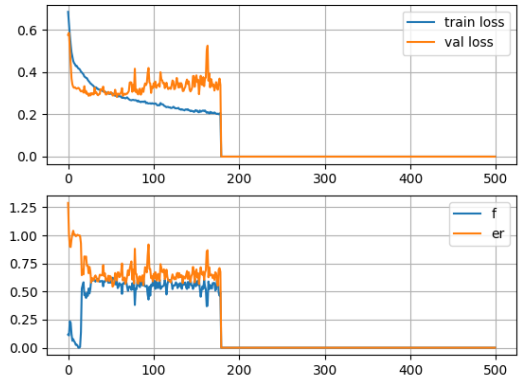
Una de las modificaciones realizadas sobre la arquitectura de la red empleada para el `sistema de referencia` (sistema que se entrena utilizando la base de datos `TUT Sound events 2017`) y el `sistema urban` (sistema que se entrena utilizando los datos de *8* clases sonoras pertenecientes a la base de datos `Urbansound8k`), consiste en añadir una capa convolucional adicional en cada bloque convolucional, situándola antes de la etapa de la normalización por lotes o BatchNormalization y a continuación de la primera capa convolucional.

Este implementación se realizó modificando la función `get_model` de los scripts `sed.py` y `sed_urban.py`, utilizados en el `sistema de referencia` y el `sistema urban`, generando *2* nuevos sistemas a evaluar.

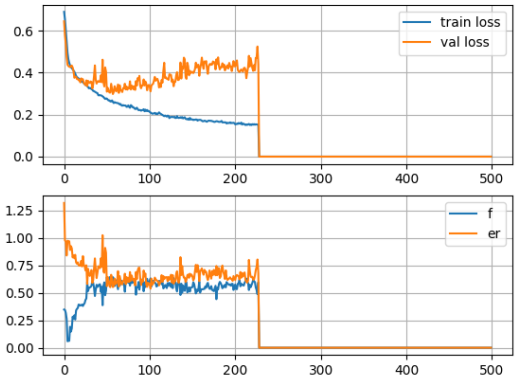
Tras realizar esta modificación en los archivos `sed.py` y `sed_urban.py`, los nuevos sistemas son entrenados para generar nuevos modelos; y estos se evalúan utilizando sus conjuntos de evaluación correspondientes, tal y como se evaluaron sus anteriores modelos sin modificaciones del `sistema de referencia` o `sistema urban`.



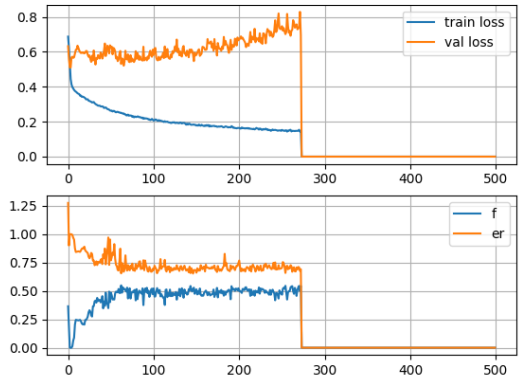
(a) Gráficas del entrenamiento del primer modelo de validación cruzada



(b) Gráficas del entrenamiento del segundo modelo de validación cruzada



(c) Gráficas del entrenamiento del tercer modelo de validación cruzada



(d) Gráficas del entrenamiento del cuarto modelo de validación cruzada

Figura 4.3: Gráficas de cada modelo de pliegue de validación cruzada del *sistema de referencia con una capa convolucional adicional en cada bloque convolucional* que muestran las *pérdidas de entrenamiento y de validación*, al igual que *la puntuación media (F) y la tasa de error (ER) en segmentos de un segundo evaluadas sobre el conjunto de validación*

4.6.2 Resultados del sistema de referencia con una capa convolucional adicional en cada bloque convolucional

Durante el entrenamiento, los resultados obtenidos de `la tasa de error por segmento (ER)` y `la puntuación media (F) o F-score` calculada en segmentos de un segundo de duración utilizando micropromediado sobre el conjunto de validación promediado en cuatro pliegues fueron de una *ER* de *0.6168* y una *F* de *0.5652*; valores bastante parejos a los obtenidos en el sistema de referencia mostrados en el apartado 4.3.3, los cuales, muestran una *ER* de *0.593* y una *F* de *0.5852*. En este caso, se obtiene una peor *ER* pero una mejor *F*.

En la figura 4.3 se pueden apreciar *2* gráficas para cada modelo de pliegue de validación cruzada que muestran las *pérdidas de entrenamiento y de validación*, al igual que *la puntuación media (F) y la tasa de error (ER) en segmentos de un segundo utilizando micropromediado del conjunto de validación* en las diferentes épocas del entrenamiento.

Métricas basadas en segmentos	
Longitud evaluada	6898.28 sg
Archivos evaluados	4
Longitud del segmento	1 sg
Métricas generales (micropromedio)	
Puntuación media (F1)	43.54 %
Precisión	45.72 %
Recuperación	41.56 %
Tasa de error (ER)	0.87
Tasa de sustitución	0.21
Tasa de eliminación	0.38
Tasa de inserción	0.29
Sensibilidad	41.56 %
Especificidad	90.87 %
Exactitud equilibrada	66.21 %
Exactitud	83.17 %
Métricas promedio por clase (macropromedio)	
Puntuación media (F1)	26.85 %
Precisión	43.74 %
Recuperación	28.14 %
Tasa de error (ER)	1.35
Tasa de eliminación	0.72
Tasa de inserción	0.63
Sensibilidad	28.14 %
Especificidad	90.08 %
Exactitud equilibrada	59.11 %
Exactitud	83.17 %

Tabla 4.6: Métricas basadas en segmentos del sistema de referencia con una capa convolucional adicional en cada bloque convolucional

Los resultados obtenidos de las diferentes métricas medidas al evaluar las predicciones generadas por los distintos modelos de este sistema sobre el conjunto de evaluación de la base de datos `TUT Sound events 2017` (que utiliza el sistema de referencia de forma predeterminada) se reflejan en la tabla 4.6.

Como resultados, en este sistema se puede observar que se obtuvo una *ER* de *0.87* y una *F* de *43.54 %* basadas en segmentos de *1* segundo utilizando micropromediado. Estos resultados son muy parecidos a los obtenidos en el sistema de referencia evaluado con el conjunto de evaluación de la base de datos `TUT Sound events 2017`, mostrados en el apartado 4.3.3, en el que se obtuvieron una *ER* de *0.85* y una *F* de *41.95 %* basadas en segmentos de *1* segundo utilizando micropromediado. En el caso de este sistema, la *ER* empeora un poco pero la *F* mejora en las métricas basadas en segmentos de *1* segundo utilizando micropromediado.

Este sistema, al contener una capa convolucional adicional en cada bloque convolucional provoca que la red contenga un mayor número de parámetros que el sistema de referencia, siendo el número de parámetros entrenables de la red resulta ser de *809958* frente a los *366438* parámetros entrenables de la red que utiliza el sistema de referencia, lo que se traduce en un mayor coste computacional, y por tanto, un mayor tiempo de entrenamiento del sistema.

Como se puede observar la tabla 4.6, más allá de los valores de la *ER* y la *F* basadas en segmentos de *1* segundo utilizando micropromediado, los valores del resto de métricas suelen ser mejores en el sistema de referencia (cuyos resultados se pueden observar en el apartado 4.3.3). A pesar de estos hechos, posicionarse a favor de un sistema u otro es complicado debido a que seleccionar un sistema depende de la necesidad de obtener una mejor métrica para una tarea en específico. Si el sistema requiere conseguir una mejor *F* basadas en segmentos de *1* segundo utilizando micropromediado que el sistema de referencia y se esta dispuesto a asumir el aumento del coste computacional para el entrenamiento de la red, este sistema podría llegar a ser una buena elección; sin embargo, si se prioriza el valor de otra métrica diferente (como

Métricas basadas en segmentos	
Longitud evaluada	18997.82 sg
Archivos evaluados	4
Longitud del segmento	1 sg
Métricas generales (micropromedio)	
Puntuación media (F1)	97.41 %
Precisión	97.24 %
Recuperación	97.58 %
Tasa de error (ER)	0.03
Tasa de sustitución	0.02
Tasa de eliminación	0.01
Tasa de inserción	0.01
Sensibilidad	97.58 %
Especificidad	99.50 %
Exactitud equilibrada	98.54 %
Exactitud	99.20 %
Métricas promedio por clase (macropromedio)	
Puntuación media (F1)	97.19 %
Precisión	97.07 %
Recuperación	97.31 %
Tasa de error (ER)	0.06
Tasa de eliminación	0.03
Tasa de inserción	0.03
Sensibilidad	97.31 %
Especificidad	99.49 %
Exactitud equilibrada	98.40 %
Exactitud	99.20 %

Tabla 4.7: Métricas basadas en segmentos del sistema urban con una capa convolucional adicional en cada bloque convolucional

por ejemplo, la *ER* basadas en segmentos de *1* segundo utilizando micropromediado), quizás no merezca la pena realizar esta implementación para este sistema en concreto.

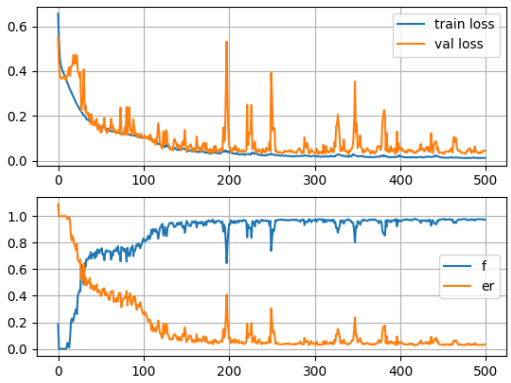
4.6.3 Resultados del sistema urban con una capa convolucional adicional en cada bloque convolucional

Durante el entrenamiento, los resultados obtenidos de `la tasa de error por segmento (ER)` y `la puntuación media (F)` o `F-score` calculada en segmentos de un segundo de duración utilizando micropromediado sobre el conjunto de validación promediado en cuatro pliegues fueron de una *ER* de *0.0283* y una *F* de *0.9776*; valores aún mejores que los obtenidos en el sistema urban mostrados en el apartado 4.5.2, los cuales, muestran una *ER* de *0.042* y una *F* de *0.9679*.

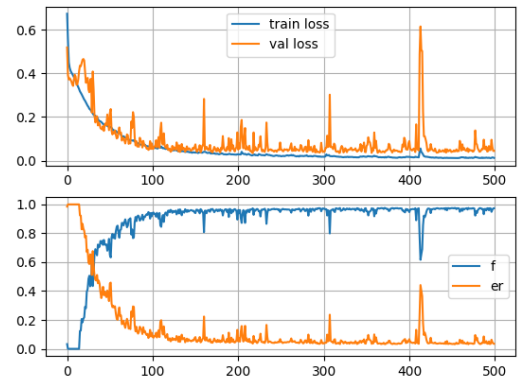
En la figura 4.4 se pueden apreciar *2* gráficas para cada modelo de pliegue de validación cruzada que muestran las *pérdidas de entrenamiento y de validación*, al igual que *la puntuación media (F) y la tasa de error (ER) en segmentos de un segundo utilizando micropromediado del conjunto de validación* en las diferentes épocas del entrenamiento.

Los resultados obtenidos de las diferentes métricas medidas al evaluar las predicciones generadas por los distintos modelos de este sistema sobre el conjunto de evaluación que utiliza el sistema urban se muestran en la tabla4.7.

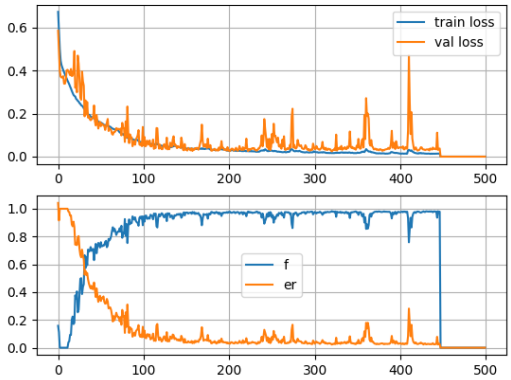
Como resultados, en este sistema se puede observar que se obtuvo una *ER* de *0.03* y una *F* de *97.41 %* basadas en segmentos de *1* segundo utilizando micropromediado. Estos resultados son muy parecidos a los obtenidos en el sistema urban evaluado, mostrados en el apartado 4.5.2, en el que se obtuvieron una *ER* de *0.05* y una *F* de *96.51 %* basadas en segmentos de *1* segundo utilizando micropromediado, pero muestran una pequeña mejora para ambas métricas.



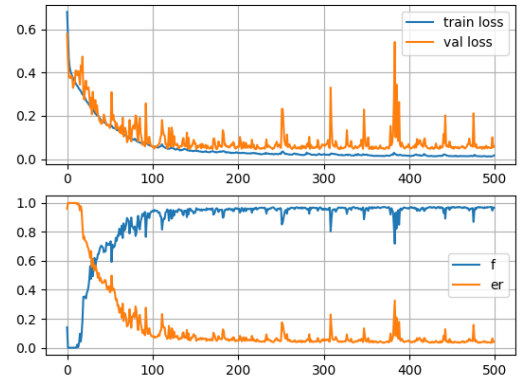
(a) Gráficas del entrenamiento del primer modelo de validación cruzada



(b) Gráficas del entrenamiento del segundo modelo de validación cruzada



(c) Gráficas del entrenamiento del tercer modelo de validación cruzada



(d) Gráficas del entrenamiento del cuarto modelo de validación cruzada

Figura 4.4: Gráficas de cada modelo de pliegue de validación cruzada del *sistema urban con una capa convolucional adicional en cada bloque convolucional* que muestran las *pérdidas de entrenamiento y de validación*, al igual que *la puntuación media (F) y la tasa de error (ER) en segmentos de un segundo evaluadas sobre el conjunto de validación*

Este sistema, al contener una capa convolucional adicional en cada bloque convolucional provoca que la red contenga un mayor número de parámetros que el sistema urban, siendo el número de parámetros entrenables de la red resulta ser de *809256* frente a los *366504* parámetros entrenables de la red que utiliza el sistema urban, lo que se traduce en un mayor coste computacional, y por tanto, un mayor tiempo de entrenamiento del sistema.

Como se puede observar la tabla de resultados 4.7, más allá de los valores de la *ER* y la *F* basadas en segmentos de *1* segundo utilizando micropromediado, todos los valores del resto de métricas son mejores en este nuevo sistema que en el sistema urban (cuyos resultados se pueden observar en el apartado 4.5.2) a excepción de la *tasa de eliminación* en basadas en segmentos de *1* segundo utilizando micropromediado que es de *0.01* para ambos sistemas. Con estos resultados se puede confirmar que para el sistema urban en concreto, el hecho de introducir una capa capa convolucional extra en cada bloque convolucional mejora su rendimiento y es una buena implementación, siempre y cuando se esté dispuesto a asumir el aumento del coste computacional del entrenamiento de la red.

4.7 Estudio del impacto de diferentes tasas de dropout

4.7.1 Descripción y condiciones experimentales

De entre todas las modificaciones realizadas, tanto para el `sistema de referencia`, como para el `sistema urban`, una de ellas consiste en modificar el valor de la `tasa de dropout` de la red neuronal para alterar el número de conexiones activas entre las diferentes capas de la red neuronal durante su entrenamiento. El valor de `la tasa de dropout` utilizado para estos sistemas durante su entrenamiento es de *0.5*, y para este apartado, se generaran *2* nuevos sistemas asociados al `sistema de referencia` y otros *2* asociados al `sistema urban` con un valor de `tasa de dropout` de *0.3* y *0.7* respectivamente.

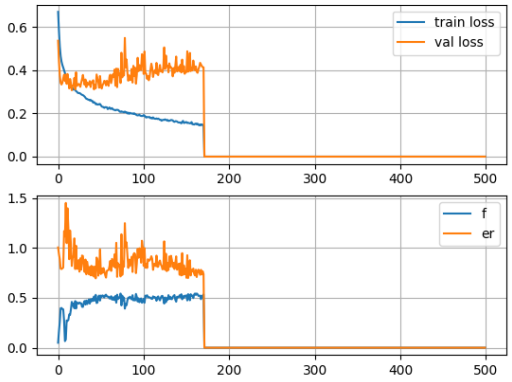
Este parámetro de `tasa de dropout` toma valores en un rango de *0* a *1*, siendo *0.5* el valor que se usa de forma predeterminada, el cual indica que la mitad de las conexiones entre las neuronas de las capas entre las que se aplique la `tasa de dropout` permanecen activadas. Si el valor de este parámetro es más próximos a *0*, se desactivan menos conexiones entre neuronas de una capa y otra; y en el caso de que sea próximo a *1*, se desactivan muchas conexiones entre neuronas de una capa y otra.

Este implementación se realizó modificando la variable `dropout_rate` de los scripts `sed.py` y `sed_urban.py`, utilizados en el `sistema de referencia` y el `sistema urban` respectivamente.

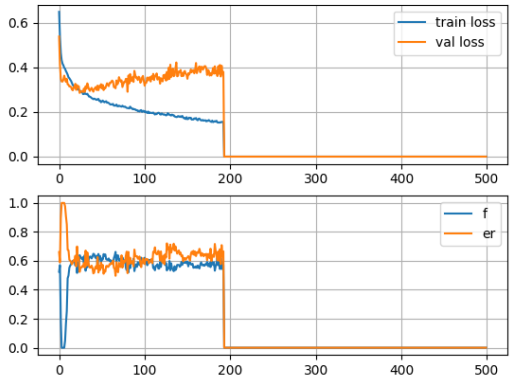
Tras realizar esta modificación en los archivos `sed.py` y `sed_urban.py`, los sistemas son entrenados para generar nuevos modelos; y estos se evalúan utilizando sus conjuntos de evaluación correspondientes, tal y como se evaluaron sus anteriores modelos sin modificaciones del `sistema de referencia` o `sistema urban`.

4.7.2 Resultados del sistema de referencia utilizando diferentes tasas de dropout

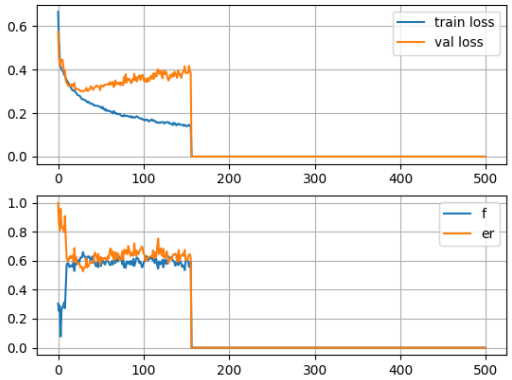
Durante el entrenamiento del sistema de referencia que utiliza una tasa de dropout de *0.3*, los resultados obtenidos de `la tasa de error por segmento (`*ER*`)` y `la puntuación media (`*F*`)` o `F-score` calculada en segmentos de un segundo de duración utilizando micropromediado sobre el conjunto de validación promediado en cuatro pliegues fueron de una *ER* de *0.5866* y una *F* de *0.6023*;



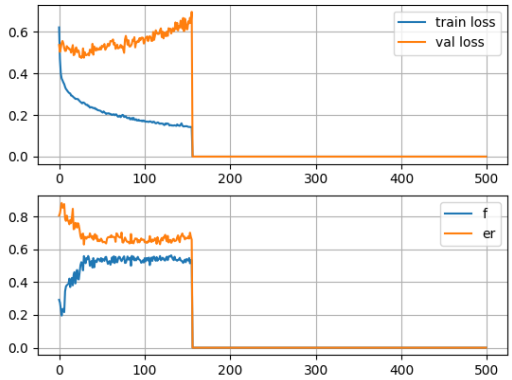
(a) Gráficas del entrenamiento del primer modelo de validación cruzada



(b) Gráficas del entrenamiento del segundo modelo de validación cruzada



(c) Gráficas del entrenamiento del tercer modelo de validación cruzada



(d) Gráficas del entrenamiento del cuarto modelo de validación cruzada

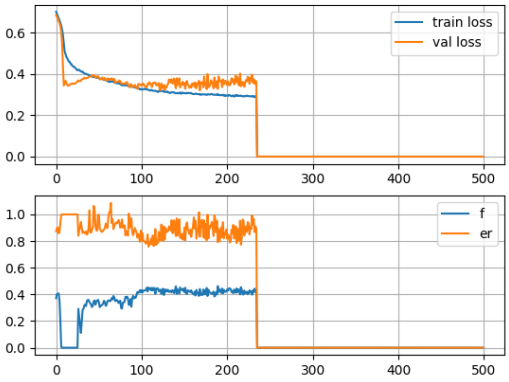
Figura 4.5: Gráficas de cada modelo de pliegue de validación cruzada del *sistema de referencia que utiliza una tasa de dropout de* 0.3 que muestran las *pérdidas de entrenamiento y de validación*, al igual que *la puntuación media (F) y la tasa de error (ER) en segmentos de un segundo evaluadas sobre el conjunto de validación*

mientras que los valores obtenidos para el sistema de referencia que utiliza una tasa de dropout de *0.7* son de *0.6168* y *0.5591* para la *ER* y la *F* respectivamente.

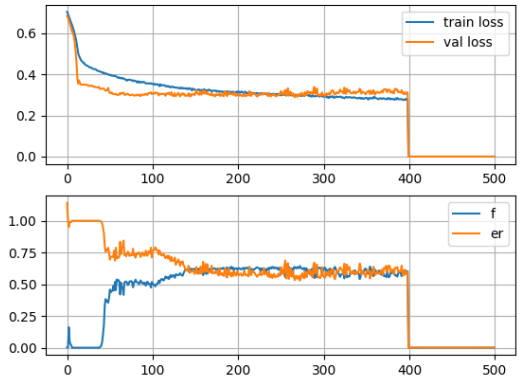
Ambos resultados son bastante parejos a los obtenidos en el sistema de referencia mostrados en el apartado 4.3.3, los cuales, muestran una *ER* de *0.593* y una *F* de *0.5852*. Para el sistema de referencia que utiliza una tasa de dropout de *0.3*, se obtienen unas pequeñas mejoras en los valores de *ER* y de *F*, en cambio, para el sistema de referencia que utiliza una tasa de dropout de *0.7* empeora un poco el valor de la *ER* pero mejora el valor de la *F*.

En la figura 4.5 se pueden apreciar *2* gráficas para cada modelo de pliegue de validación cruzada que muestran las *pérdidas de entrenamiento y de validación*, al igual que *la puntuación media (F) y la tasa de error (ER) en segmentos de un segundo utilizando micropromediado del conjunto de validación* en las diferentes épocas del entrenamiento del sistema de referencia que utiliza una tasa de dropout de *0.3*.

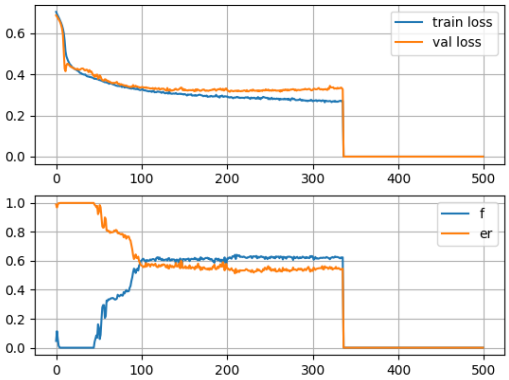
En la figura 4.6 se pueden apreciar *2* gráficas para cada modelo de pliegue de validación cruzada que muestran las *pérdidas de entrenamiento y de validación*, al igual que *la puntuación media (F) y la tasa*



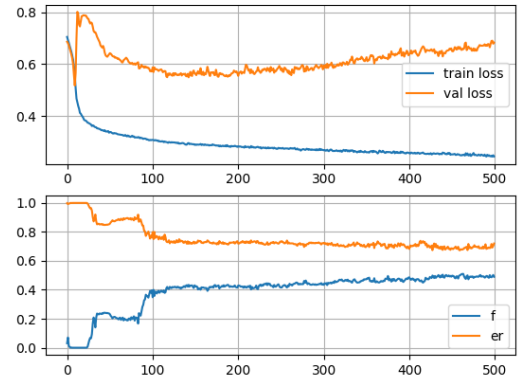
(a) Gráficas del entrenamiento del primer modelo de validación cruzada



(b) Gráficas del entrenamiento del segundo modelo de validación cruzada



(c) Gráficas del entrenamiento del tercer modelo de validación cruzada



(d) Gráficas del entrenamiento del cuarto modelo de validación cruzada

Figura 4.6: Gráficas de cada modelo de pliegue de validación cruzada del *sistema de referencia que utiliza una tasa de dropout de* 0.7 que muestran las *pérdidas de entrenamiento y de validación*, al igual que *la puntuación media (F) y la tasa de error (ER) en segmentos de un segundo evaluadas sobre el conjunto de validación*

de error (ER) en segmentos de un segundo utilizando micropromediado del conjunto de validación en las diferentes épocas del entrenamiento del sistema de referencia que utiliza una tasa de dropout de *0.7*.

Los resultados obtenidos de las diferentes métricas medidas al evaluar las predicciones generadas por los distintos modelos del sistema de referencia que utiliza una `tasa de dropout` de *0.3* y el sistema de referencia que utiliza una `tasa de dropout` de *0.7* sobre el conjunto de evaluación de la base de datos `TUT Sound events 2017` (que utiliza el sistema de referencia de forma predeterminada) se muestran en la tabla 4.8.

Para el sistema de referencia que utiliza una tasa de dropout de *0.3* se puede observar que se obtuvo una *ER* de *0.87* y una *F* de *44.75 %* basadas en segmentos de *1* segundo utilizando micropromediado, mientras que para el sistema de referencia que utiliza una tasa de dropout de *0.7* se obtuvo una *ER* de *0.86* y una *F* de *44.19 %* basadas en segmentos de *1* segundo utilizando micropromediado. Estos resultados se asemejan a los obtenidos en el sistema de referencia evaluado con el conjunto de evaluación de la base de datos `TUT Sound events 2017`, mostrados en el apartado 4.3.3, en el que se obtuvieron una *ER* de *0.85* y una *F* de *41.95 %* basadas en segmentos de *1* segundo utilizando micropromediado.

Con una tasa de dropout de 0.3	
Métricas basadas en segmentos	
Longitud evaluada	6898.28 sg
Archivos evaluados	4
Longitud del segmento	1 sg
Métricas generales (micropromedio)	
Puntuación media (F1)	44.75 %
Precisión	44.86 %
Recuperación	44.64 %
Tasa de error (ER)	0.87
Tasa de sustitución	0.23
Tasa de eliminación	0.32
Tasa de inserción	0.32
Sensibilidad	44.64 %
Especificidad	89.80 %
Exactitud equilibrada	67.22 %
Exactitud	82.72 %
Métricas promedio por clase (macropromedio)	
Puntuación media (F1)	28.18 %
Precisión	35.20 %
Recuperación	31.09 %
Tasa de error (ER)	1.28
Tasa de eliminación	0.69
Tasa de inserción	0.59
Sensibilidad	31.09 %
Especificidad	88.83 %
Exactitud equilibrada	59.96 %
Exactitud	82.72 %

Con una tasa de dropout de 0.7	
Métricas basadas en segmentos	
Longitud evaluada	6898.28 sg
Archivos evaluados	4
Longitud del segmento	1 sg
Métricas generales (micropromedio)	
Puntuación media (F1)	44.19 %
Precisión	47.08 %
Recuperación	41.63 %
Tasa de error (ER)	0.86
Tasa de sustitución	0.19
Tasa de eliminación	0.39
Tasa de inserción	0.27
Sensibilidad	41.63 %
Especificidad	91.25 %
Exactitud equilibrada	66.44 %
Exactitud	83.43 %
Métricas promedio por clase (macropromedio)	
Puntuación media (F1)	26.57 %
Precisión	44.78 %
Recuperación	25.70 %
Tasa de error (ER)	1.27
Tasa de eliminación	0.74
Tasa de inserción	0.52
Sensibilidad	25.70 %
Especificidad	90.09 %
Exactitud equilibrada	57.90 %
Exactitud	83.43 %

Tabla 4.8: Métricas basadas en segmentos del sistema de referencia utilizando una tasa de dropout de 0.3 y 0.7

En el caso de estos sistemas (tanto el sistema de referencia que utiliza una tasa de dropout de *0.3* como el que utiliza una tasa de dropout de *0.7*), la *ER* empeora un poco pero la *F* mejora en las métricas basadas en segmentos de *1* segundo utilizando micropromediado.

Tanto para el sistema de referencia que utiliza una tasa de dropout de *0.3* como para el que utiliza una tasa de dropout de *0.7* no se produce ningún aumento de parámetros entrenables durante el entrenamiento de la red, aunque para el sistema de referencia que utiliza una tasa de dropout de *0.3*, al producirse una mayor cantidad de conexiones entre neuronas respecto al sistema de referencia, el entrenamiento se demora mas; ocurriendo lo contrario para el sistema de refrencia que utiliza una tasa de dropout de *0.7*. Aun así, estas demoras frente al tiempo total de entrenamiento suponen un pequeño porcentaje de tiempo por lo que no es algo realmente a tener en cuenta.

Debido a que ambos sistemas no muestran unas mejorías significativas, más allá de en la *F* basada en segmentos de *1* segundo utilizando micropromediado, que resulta ser mejor aún en el sistema de referencia que utiliza una tasa de dropout de *0.3* en comparación con el sistema de referencia que utiliza una tasa de dropout de *0.7*, para este caso es difícil posicionarse sobre si realmente existe una mejora de rendimiento para uno de los *2* sistemas nuevamente entrenados en comparación el rendimiento ofrecido por el sistema de referencia, cuyas métricas se puede observar en el apartado 4.3.3.

4.7.3 Resultados del sistema urban utilizando diferentes tasas de dropout

Durante el entrenamiento del sistema urban que utiliza una tasa de dropout de *0.3*, los resultados obtenidos de `la tasa de error por segmento (ER)` y `la puntuación media (F) o F-score` calculada en segmentos de un segundo de duración utilizando micropromediado sobre el conjunto de validación promediado en cuatro pliegues fueron de una *ER* de *0.0318* y una *F* de *0.9751*; mientras que los valores obtenidos para el sistema urban que utiliza una tasa de dropout de *0.7* son de *0.1853* y *0.8711* para la *ER* y la *F* respectivamente.

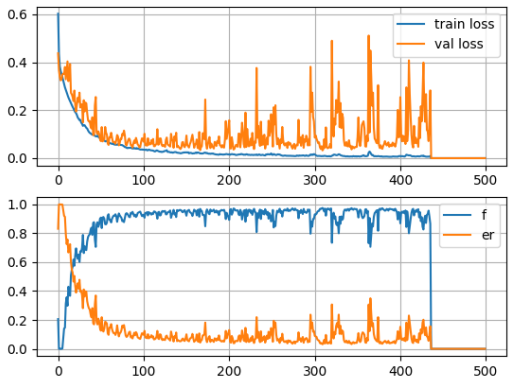
Para el caso de los resultados obtenidos en el sistema urban que utiliza una tasa de dropout de *0.3*, ambos resultados son bastante parejos y se obtienen unas pequeñas mejoras en comparación con los obtenidos en el sistema urban mostrados en el apartado 4.5.2, los cuales, muestran una *ER* de *0.042* y una *F* de *0.9679*. En cambio, para el sistema urban que utiliza una tasa de dropout de *0.7* ambas métricas empeoran significativamente.

En la figura 4.7 se pueden apreciar *2* gráficas para cada modelo de pliegue de validación cruzada que muestran las *pérdidas de entrenamiento y de validación*, al igual que *la puntuación media (F) y la tasa de error (ER) en segmentos de un segundo utilizando micropromediado del conjunto de validación* en las diferentes épocas del entrenamiento del sistema urban que utiliza una tasa de dropout de *0.3*.

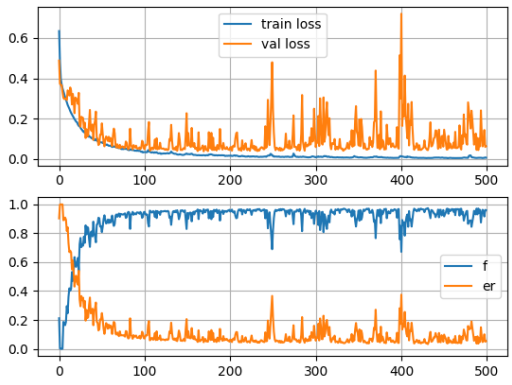
En la figura 4.8 se pueden apreciar *2* gráficas para cada modelo de pliegue de validación cruzada que muestran las *pérdidas de entrenamiento y de validación*, al igual que *la puntuación media (F) y la tasa de error (ER) en segmentos de un segundo utilizando micropromediado del conjunto de validación* en las diferentes épocas del entrenamiento del sistema urban que utiliza una tasa de dropout de *0.7*.

Los resultados obtenidos de las diferentes métricas medidas al evaluar las predicciones generadas por los distintos modelos del sistema urban que utiliza una `tasa de dropout` de *0.3* y el sistema urban que utiliza una `tasa de dropout` de *0.7* sobre el conjunto de evaluación que utiliza el sistema urban se muetsran en la tabla 4.9.

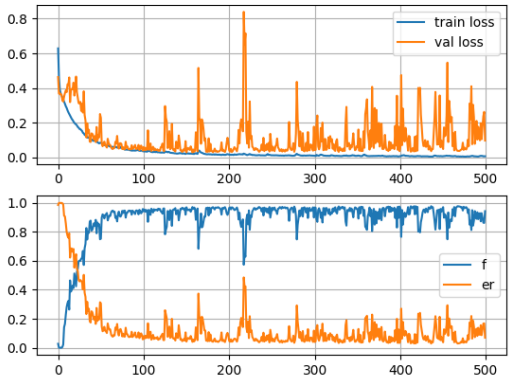
Para el sistema urban que utiliza una tasa de dropout de *0.3* se puede observar que se obtuvo una *ER* de *0.04* y una *F* de *97.09 %* basadas en segmentos de *1* segundo utilizando micropromediado, mientras que para el sistema urban que utiliza una tasa de dropout de *0.7* se obtuvo una *ER* de *0.19* y una *F*



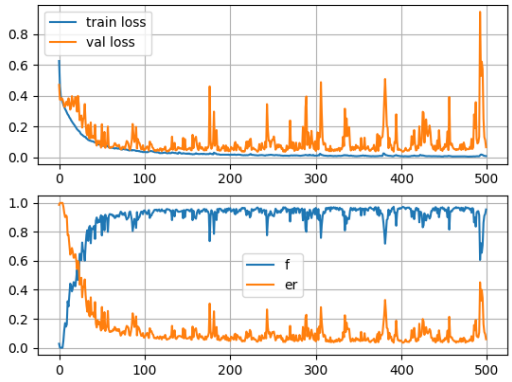
(a) Gráficas del entrenamiento del primer modelo de validación cruzada



(b) Gráficas del entrenamiento del segundo modelo de validación cruzada

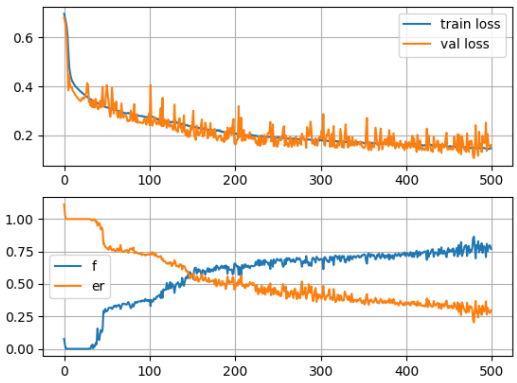


(c) Gráficas del entrenamiento del tercer modelo de validación cruzada

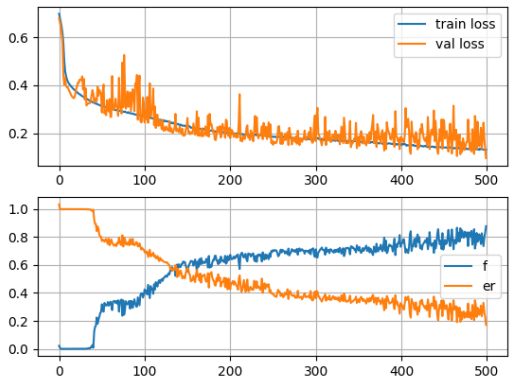


(d) Gráficas del entrenamiento del cuarto modelo de validación cruzada

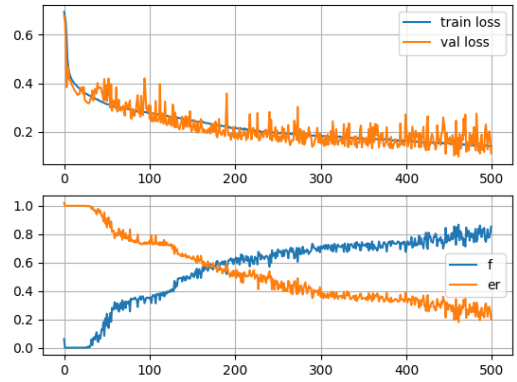
Figura 4.7: Gráficas de cada modelo de pliegue de validación cruzada del *sistema urban que utiliza una tasa de dropout de* 0.3 que muestran las *pérdidas de entrenamiento y de validación*, al igual que *la puntuación media (F) y la tasa de error (ER) en segmentos de un segundo evaluadas sobre el conjunto de validación*



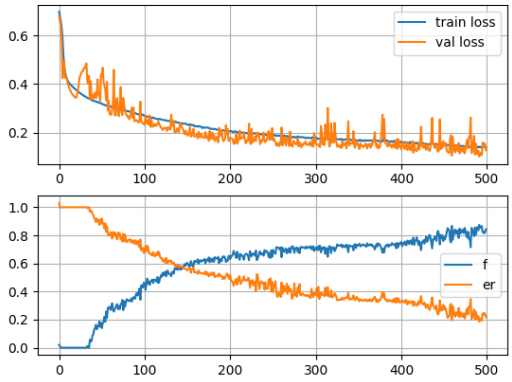
(a) Gráficas del entrenamiento del primer modelo de validación cruzada



(b) Gráficas del entrenamiento del segundo modelo de validación cruzada



(c) Gráficas del entrenamiento del tercer modelo de validación cruzada



(d) Gráficas del entrenamiento del cuarto modelo de validación cruzada

Figura 4.8: Gráficas de cada modelo de pliegue de validación cruzada del *sistema urban que utiliza una tasa de dropout de* 0.7 que muestran las *pérdidas de entrenamiento y de validación*, al igual que *la puntuación media (F) y la tasa de error (ER) en segmentos de un segundo evaluadas sobre el conjunto de validación*

Con una tasa de dropout de 0.3	
Métricas basadas en segmentos	
Longitud evaluada	18997.82 sg
Archivos evaluados	4
Longitud del segmento	1 sg
Métricas generales (micropromedio)	
Puntuación media (F1)	97.09 %
Precisión	96.73 %
Recuperación	97.45 %
Tasa de error (ER)	0.04
Tasa de sustitución	0.02
Tasa de eliminación	0.01
Tasa de inserción	0.01
Sensibilidad	97.45 %
Especificidad	99.40 %
Exactitud equilibrada	98.42 %
Exactitud	99.10 %
Métricas promedio por clase (macropromedio)	
Puntuación media (F1)	96.74 %
Precisión	96.33 %
Recuperación	97.15 %
Tasa de error (ER)	0.07
Tasa de eliminación	0.03
Tasa de inserción	0.04
Sensibilidad	97.15 %
Especificidad	99.39 %
Exactitud equilibrada	98.27 %
Exactitud	99.10 %

Con una tasa de dropout de 0.7	
Métricas basadas en segmentos	
Longitud evaluada	18997.82 sg
Archivos evaluados	4
Longitud del segmento	1 sg
Métricas generales (micropromedio)	
Puntuación media (F1)	86.57 %
Precisión	89.37 %
Recuperación	83.94 %
Tasa de error (ER)	0.19
Tasa de sustitución	0.07
Tasa de eliminación	0.09
Tasa de inserción	0.03
Sensibilidad	83.94 %
Especificidad	98.19 %
Exactitud equilibrada	91.06 %
Exactitud	96.00 %
Métricas promedio por clase (macropromedio)	
Puntuación media (F1)	90.54 %
Precisión	89.55 %
Recuperación	68.68 %
Tasa de error (ER)	0.39
Tasa de eliminación	0.31
Tasa de inserción	0.08
Sensibilidad	68.68 %
Especificidad	98.10 %
Exactitud equilibrada	83.39 %
Exactitud	96.00 %

Tabla 4.9: Métricas basadas en segmentos del sistema urban utilizando una tasa de dropout de 0.3 y 0.7

de *86.57 %* basadas en segmentos de *1* segundo utilizando micropromediado. Resultados muy parecidos a los obtenidos en sus conjuntos de validación durante su entrenamiento.

Tanto para el sistema urban que utiliza una tasa de dropout de *0.3* como para el que utiliza una tasa de dropout de *0.7* no se produce ningún aumento de parámetros entrenables durante el entrenamiento de la red, aunque para el sistema urban que utiliza una tasa de dropout de *0.3*, al producirse una mayor cantidad de conexiones entre neuronas respecto al sistema urban, el entrenamiento se demora mas; ocurriendo lo contrario para el sistema de refrencia que utiliza una tasa de dropout de *0.7*. Aun así, estas demoras frente al tiempo total de entrenamiento suponen un pequeño porcentaje de tiempo por lo que no es algo realmente a tener en cuenta.

En el caso de los resultados obtenidos en el sistema urban que utiliza una tasa de dropout de *0.3*, casi todos los valores de sus métricas son mejores que los obtenidos en el sistema urban (que utiliza una tasa de dropout de *0.5* de forma predeterminada) mostradas en el apartado 4.5.2, por lo que, se repiten las mismas conclusiones dadas para el sistema urban que utiliza una capa convolucional adicional en cada bloque convolucional, pudiendo afirmar que esta implementación realmente mejora el rendimiento del rendimiento urban. Todo lo contrario ocurre con los valores de las métricas correspondientes al sistema urban que utiliza una tasa de dropout de *0.7* frente a los resultados obtenidos en el sistema urban, por lo que en este caso, se puede afirmar que esta implementación empeora el rendimiento del sistema urban.

4.8 Estudio del impacto del tamaño del kernel en las capas convolucionales

4.8.1 Descripción y condiciones experimentales

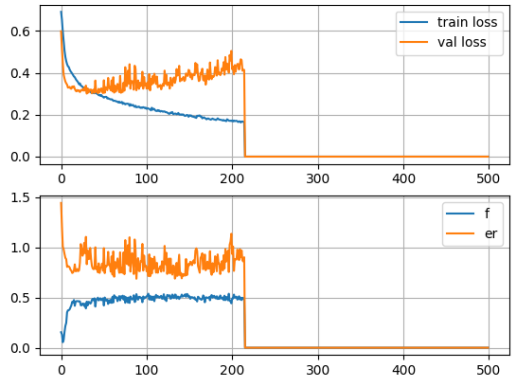
En ese apartado se modifican las dimensiones de las `matrices de kernel` (tambien conocidos como filtros receptivos) utilizadas en las capas convolucionales, tanto en la arquitectura del `sistema de referencia`, como en la del `sistema urban`. Para ello, se generaran *2* nuevos sistemas asociados al `sistema de referencia` y otros *2* asociados al `sistema urban` que utilizan unas `matrices de kernel` en sus capas convolucionales de dimensiones *5x5* y *7x7* respectivamente, en lugar de las `matrices de kernel` de dimensión *3x3* utilizadas en estos sistemas durante su entrenamiento.

Este implementación se realizó modificando la variable `kernel_size` de cada capa convolucional definidas dentro de la función `get_model` de los scripts `sed.py` y `sed_urban.py`, utilizados en el `sistema de referencia` y el `sistema urban` respectivamente.

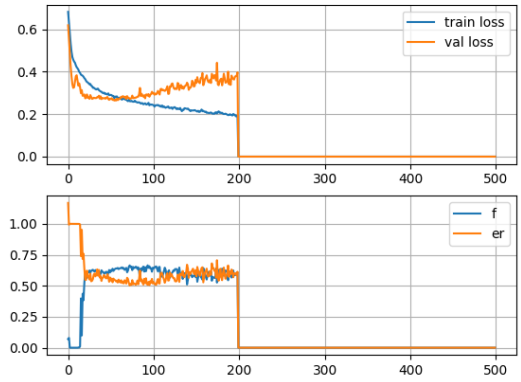
Tras realizar esta modificación en los archivos `sed.py` y `sed_urban.py`, los sistemas son entrenados para generar nuevos modelos; y estos se evalúan utilizando sus conjuntos de evaluación correspondientes, tal y como se evaluaron sus anteriores modelos sin modificaciones del `sistema de referencia` o `sistema urban`.

4.8.2 Resultados del sistema de referencia utilizando diferentes dimensiones de las matrices de Kernel en sus capas convolucionales

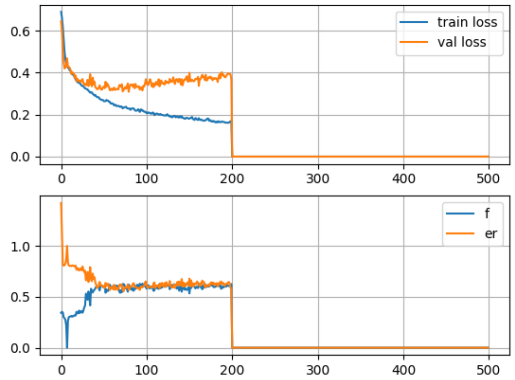
Durante el entrenamiento del sistema de referencia que utiliza matrices de Kernel en sus capas convolucionales con dimensiones de *5x5*, los resultados obtenidos de `la tasa de error por segmento (ER)` y `la puntuación media (F) o F-score` calculada en segmentos de un segundo de duración utilizando micropromediado sobre el conjunto de validación promediado en cuatro pliegues fueron de una



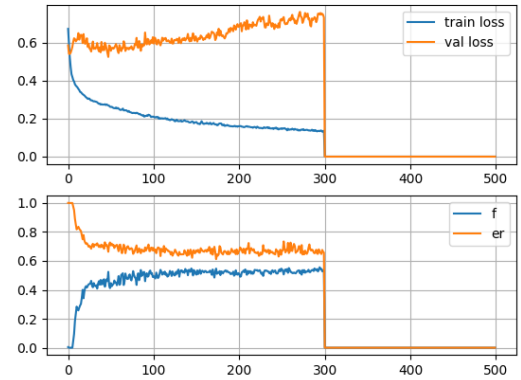
(a) Gráficas del entrenamiento del primer modelo de validación cruzada



(b) Gráficas del entrenamiento del segundo modelo de validación cruzada



(c) Gráficas del entrenamiento del tercer modelo de validación cruzada



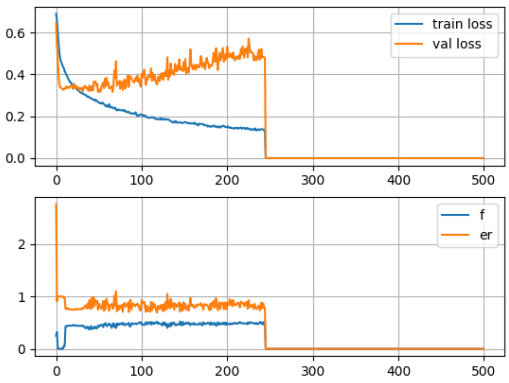
(d) Gráficas del entrenamiento del cuarto modelo de validación cruzada

Figura 4.9: Gráficas de cada modelo de pliegue de validación cruzada del *sistema de referencia que utiliza matrices de Kernel en sus capas convolucionales con dimensiones de* 5x5 que muestran las *pérdidas de entrenamiento y de validación*, al igual que *la puntuación media (F) y la tasa de error (ER) en segmentos de un segundo evaluadas sobre el conjunto de validación*

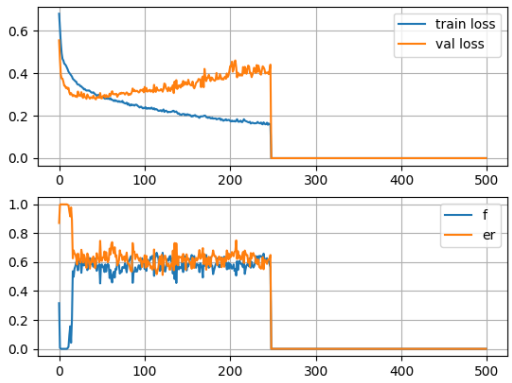
ER de *0.5889* y una *F* de *0.5928*; mientras que los valores obtenidos para el sistema de referencia que utiliza matrices de Kernel en sus capas convolucionales con dimensiones de *7x7* son de *0.5989* y *0.5875* para la *ER* y la *F* respectivamente. Ambos resultados son bastante similares a los obtenidos en el sistema de referencia mostrados en el apartado 4.3.3, los cuales, muestran una *ER* de *0.593* y una *F* de *0.5852*.

En la figura 4.9 se pueden apreciar *2* gráficas para cada modelo de pliegue de validación cruzada que muestran las *pérdidas de entrenamiento y de validación*, al igual que *la puntuación media (F) y la tasa de error (ER) en segmentos de un segundo utilizando micropromediado del conjunto de validación* en las diferentes épocas del entrenamiento del sistema de referencia que utiliza matrices de Kernel en sus capas convolucionales con dimensiones de *5x5*; mientras que estos valores para el sistema de referencia que utiliza matrices de Kernel en sus capas convolucionales con dimensiones de *7x7* se pueden observar en la figura 4.10.

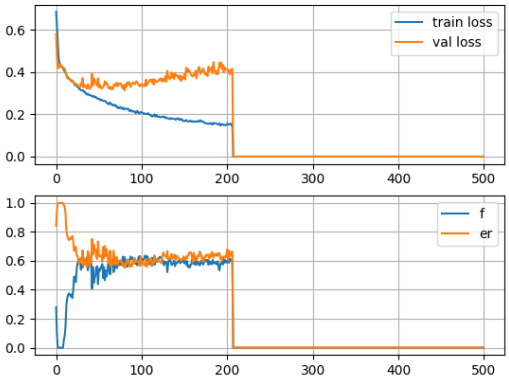
Los resultados obtenidos de las diferentes métricas medidas al evaluar las predicciones generadas por los distintos modelos del sistema de referencia que utiliza matrices de Kernel en sus capas convolucionales con dimensiones de *5x5* y el sistema de referencia que utiliza matrices de Kernel en sus capas convolucio-



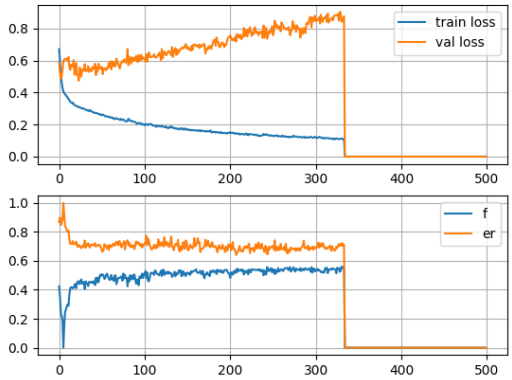
(a) Gráficas del entrenamiento del primer modelo de validación cruzada



(b) Gráficas del entrenamiento del segundo modelo de validación cruzada



(c) Gráficas del entrenamiento del tercer modelo de validación cruzada



(d) Gráficas del entrenamiento del cuarto modelo de validación cruzada

Figura 4.10: Gráficas de cada modelo de pliegue de validación cruzada del *sistema de referencia que utiliza matrices de Kernel en sus capas convolucionales con dimensiones de* 7x7 que muestran las *pérdidas de entrenamiento y de validación*, al igual que *la puntuación media (F) y la tasa de error (ER) en segmentos de un segundo evaluadas sobre el conjunto de validación*

Con matrices de Kernel de dimensión 5x5	
Métricas basadas en segmentos	
Longitud evaluada	6898.28 sg
Archivos evaluados	4
Longitud del segmento	1 sg
Métricas generales (micropromedio)	
Puntuación media (F1)	45.60 %
Precisión	46.08 %
Recuperación	45.14 %
Tasa de error (ER)	0.85
Tasa de sustitución	0.23
Tasa de eliminación	0.32
Tasa de inserción	0.30
Sensibilidad	45.14 %
Especificidad	90.22 %
Exactitud equilibrada	67.68 %
Exactitud	83.17 %
Métricas promedio por clase (macropromedio)	
Puntuación media (F1)	28.84 %
Precisión	40.41 %
Recuperación	31.29 %
Tasa de error (ER)	1.33
Tasa de eliminación	0.69
Tasa de inserción	0.64
Sensibilidad	31.29 %
Especificidad	89.30 %
Exactitud equilibrada	60.29 %
Exactitud	83.17 %

Con matrices de Kernel de dimensión 7x7	
Métricas basadas en segmentos	
Longitud evaluada	6898.28 sg
Archivos evaluados	4
Longitud del segmento	1 sg
Métricas generales (micropromedio)	
Puntuación media (F1)	42.26 %
Precisión	46.69 %
Recuperación	38.59 %
Tasa de error (ER)	0.83
Tasa de sustitución	0.22
Tasa de eliminación	0.39
Tasa de inserción	0.22
Sensibilidad	38.59 %
Especificidad	91.79 %
Exactitud equilibrada	65.19 %
Exactitud	83.43 %
Métricas promedio por clase (macropromedio)	
Puntuación media (F1)	25.71 %
Precisión	44.26 %
Recuperación	27.34 %
Tasa de error (ER)	1.27
Tasa de eliminación	0.73
Tasa de inserción	0.54
Sensibilidad	27.34 %
Especificidad	91.18 %
Exactitud equilibrada	59.26 %
Exactitud	83.43 %

Tabla 4.10: Métricas basadas en segmentos del sistema de referencia utilizando matrices de Kernel en sus capas convolucionales de dimensiones *5x5* y *7x7*

nales con dimensiones de *7x7* sobre el conjunto de evaluación de la base de datos `TUT Sound events 2017` (que utiliza el sistema de referencia de forma predeterminada) se muestran en la tabla 4.10.

Para el sistema de referencia que utiliza matrices de Kernel en sus capas convolucionales con dimensiones de *5x5* se puede observar que se obtuvo una *ER* de *0.85* y una *F* de *45.6 %* basadas en segmentos de *1* segundo utilizando micropromediado, mientras que para el sistema de referencia que utiliza matrices de Kernel en sus capas convolucionales con dimensiones de *7x7* se obtuvo una *ER* de *0.83* y una *F* de *42.27 %* basadas en segmentos de *1* segundo utilizando micropromediado.

Ambos sistemas poseen una mejora en la *F* basada en segmentos de *1* segundo utilizando micropromediado en comparación con el sistema de referencia evaluado con el conjunto de evaluación de la base de datos `TUT Sound events 2017`, mostrados en el apartado 4.3.3. Por otro lado, solo el sistema de referencia que utiliza matrices de Kernel en sus capas convolucionales con dimensiones de *7x7* mejora la *ER* basada en segmentos de *1* segundo utilizando micropromediado, ya que, en el caso del sistema de referencia que utiliza matrices de Kernel en sus capas convolucionales con dimensiones de *5x5* su valor es el mismo que el obtenido para el sistema de referencia evaluado con el conjunto de evaluación de la base de datos `TUT Sound events 2017`, en el que se obtuvieron una *ER* de *0.85* y una *F* de *41.95 %* basadas en segmentos de *1* segundo utilizando micropromediado.

Estos sistema, al contener entrenarse utilizando mayores dimensiones de las matrices de kernel en sus capas convolucionales provoca que la red contenga un mayor número de parámetros que el sistema de referencia, siendo el número de parámetros entrenables de la red resulta ser de *892774* para el sistema de referencia que utiliza matrices de Kernel en sus capas convolucionales con dimensiones de *5x5* y

de *1682278* para el sistema de referencia que utiliza matrices de Kernel en sus capas convolucionales con dimensiones de *7x7*, frente a los *366438* parámetros entrenables de la red que utiliza el sistema de referencia, lo que se traduce en un mayor coste computacional, y por tanto, en un mayor tiempo de entrenamiento del sistema.

Si el objetivo a considerar es optimizar la métrica *F* basada en segmentos de *1* segundo utilizando micropromediado, ambos sistemas ofrecen una mejora, siendo la más notable la del sistema de referencia que utiliza matrices de Kernel en sus capas convolucionales con dimensiones de *5x5*. Por otro lado, Si el objetivo a considerar es optimizar la métrica *ER* basada en segmentos de *1* segundo utilizando micropromediado, en este caso la mejor solución sería seleccionar el sistema de referencia que utiliza matrices de Kernel en sus capas convolucionales con dimensiones de *7x7* en lugar del sistema de referencia que utiliza matrices de Kernel en sus capas convolucionales con dimensiones de *5x5*. Al margen de los resultados obtenidos en las métricas basadas en segmentos de *1* segundo utilizando micropromediado, para el resto de métricas generalmente ambos sistemas ofrecen peores resultados pero no muy alejados de los obtenidos en el sistema de referencia evaluado con el conjunto de evaluación de la base de datos `TUT Sound events 2017`.

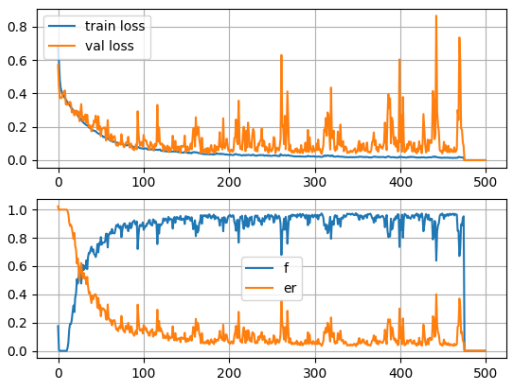
Para estos dos sistemas entrenados utilizando dimensiones diferentes de las matrices de kernel en capa convolucional de la red, tal y como ocurre con la mayoría de sistemas evaluados basados en el sistema de referencia que se evaluan utilizando el conjunto de evaluación la base de datos `TUT Sound events 2017` y unicamente contienen modificaciones en la arquitectura de la red, es muy difícil posicionarse sobre si realmente estos sistemas son más óptimos. Como se ha comentado en otras ocasiones, depende de la utilidad que se le asigne y en el caso de que alguno de estos sistemas resultase ser el más adecuado, si existe disposición de asumir la elevación de los costes computacionales durante el entrenamiento de la red debido al aumento de parámetros.

4.8.3 Resultados del sistema urban utilizando diferentes dimensiones de las matrices de Kernel en sus capas convolucionales

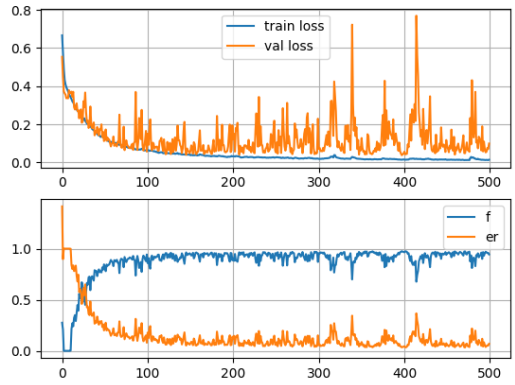
Durante el entrenamiento del sistema urban que utiliza matrices de Kernel en sus capas convolucionales con dimensiones de *5x5*, los resultados obtenidos de `la tasa de error por segmento (`*ER*`)` y `la puntuación media (`*F*`) o F-score` calculada en segmentos de un segundo de duración utilizando micropromediado sobre el conjunto de validación promediado en cuatro pliegues fueron de una *ER* de *0.0336* y una *F* de *0.9736*; mientras que los valores obtenidos para el sistema urban que utiliza matrices de Kernel en sus capas convolucionales con dimensiones de *7x7* son de *0.0348* y *0.973* para la *ER* y la *F* respectivamente. Para estos *2* sistemas, ambas métricas poseen una leve mejora respecto a las obtenidas en el sistema urban, que posee una una *ER* de *0.042* y una *F* de *0.9679*, mostradas en el apartado 4.5.2.

En la figura 4.11 se pueden apreciar *2* gráficas para cada modelo de pliegue de validación cruzada que muestran las *pérdidas de entrenamiento y de validación*, al igual que *la puntuación media (F) y la tasa de error (ER) en segmentos de un segundo utilizando micropromediado del conjunto de validación* en las diferentes épocas del entrenamiento del sistema urban que utiliza matrices de Kernel en sus capas convolucionales con dimensiones de *5x5*; mientras que estos valores para el sistema urban que utiliza matrices de Kernel en sus capas convolucionales con dimensiones de *7x7* se pueden observar en la figura 4.12.

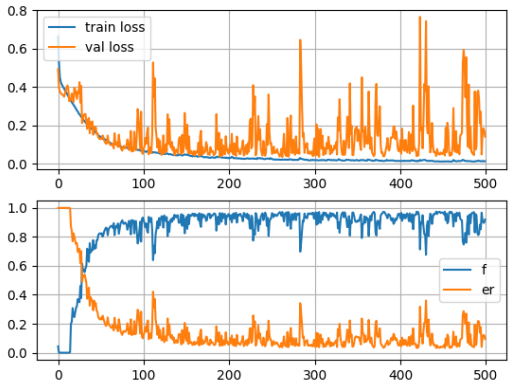
Los resultados obtenidos de las diferentes métricas medidas al evaluar las predicciones generadas por los distintos modelos del sistema urban que utiliza matrices de Kernel en sus capas convolucionales con dimensiones de *5x5* y el sistema urban que utiliza matrices de Kernel en sus capas convolucionales con



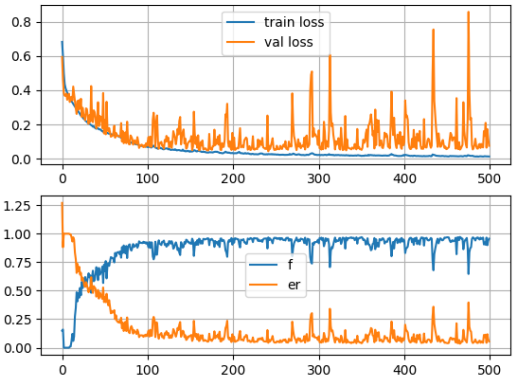
(a) Gráficas del entrenamiento del primer modelo de validación cruzada



(b) Gráficas del entrenamiento del segundo modelo de validación cruzada

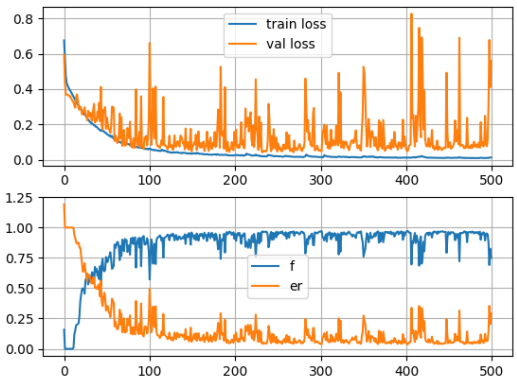


(c) Gráficas del entrenamiento del tercer modelo de validación cruzada

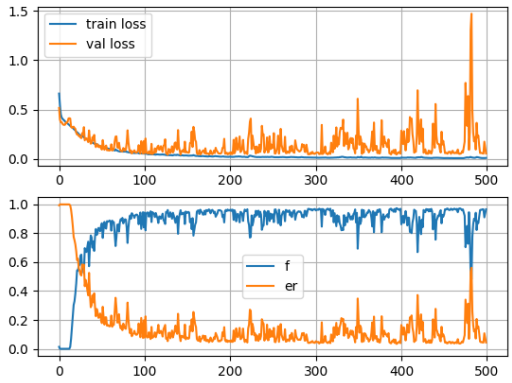


(d) Gráficas del entrenamiento del cuarto modelo de validación cruzada

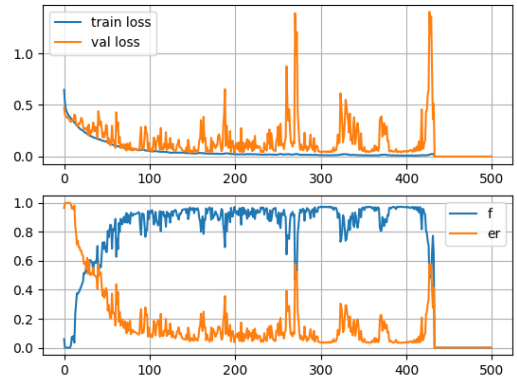
Figura 4.11: Gráficas de cada modelo de pliegue de validación cruzada del *sistema urban que utiliza matrices de Kernel en sus capas convolucionales con dimensiones de* 5x5 que muestran las *pérdidas de entrenamiento y de validación*, al igual que *la puntuación media (F) y la tasa de error (ER) en segmentos de un segundo evaluadas sobre el conjunto de validación*



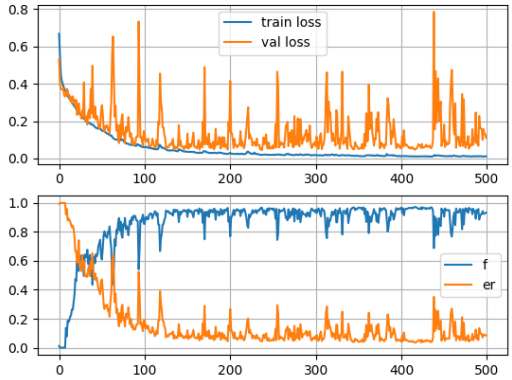
(a) Gráficas del entrenamiento del primer modelo de validación cruzada



(b) Gráficas del entrenamiento del segundo modelo de validación cruzada



(c) Gráficas del entrenamiento del tercer modelo de validación cruzada



(d) Gráficas del entrenamiento del cuarto modelo de validación cruzada

Figura 4.12: Gráficas de cada modelo de pliegue de validación cruzada del *sistema urban que utiliza matrices de Kernel en sus capas convolucionales con dimensiones de* 7x7 que muestran las *pérdidas de entrenamiento y de validación*, al igual que *la puntuación media (F) y la tasa de error (ER) en segmentos de un segundo evaluadas sobre el conjunto de validación*

Con matrices de Kernel de dimensión 5x5	
Métricas basadas en segmentos	
Longitud evaluada	18997.82 sg
Archivos evaluados	4
Longitud del segmento	1 sg
Métricas generales (micropromedio)	
Puntuación media (F1)	97.15 %
Precisión	96.92 %
Recuperación	97.39 %
Tasa de error (ER)	0.04
Tasa de sustitución	0.02
Tasa de eliminación	0.01
Tasa de inserción	0.01
Sensibilidad	97.39 %
Especificidad	99.44 %
Exactitud equilibrada	98.41 %
Exactitud	99.12 %
Métricas promedio por clase (macropromedio)	
Puntuación media (F1)	96.93 %
Precisión	96.71 %
Recuperación	97.15 %
Tasa de error (ER)	0.06
Tasa de eliminación	0.03
Tasa de inserción	0.03
Sensibilidad	97.15 %
Especificidad	99.42 %
Exactitud equilibrada	98.29 %
Exactitud	99.12 %

Con matrices de Kernel de dimensión 7x7	
Métricas basadas en segmentos	
Longitud evaluada	18997.82 sg
Archivos evaluados	4
Longitud del segmento	1 sg
Métricas generales (micropromedio)	
Puntuación media (F1)	96.96 %
Precisión	96.72 %
Recuperación	97.19 %
Tasa de error (ER)	0.04
Tasa de sustitución	0.02
Tasa de eliminación	0.01
Tasa de inserción	0.01
Sensibilidad	97.19 %
Especificidad	99.40 %
Exactitud equilibrada	98.30 %
Exactitud	99.06 %
Métricas promedio por clase (macropromedio)	
Puntuación media (F1)	96.59 %
Precisión	96.46 %
Recuperación	96.72 %
Tasa de error (ER)	0.07
Tasa de eliminación	0.03
Tasa de inserción	0.04
Sensibilidad	96.72 %
Especificidad	99.39 %
Exactitud equilibrada	98.05 %
Exactitud	99.06 %

Tabla 4.11: Métricas basadas en segmentos del sistema urban utilizando matrices de Kernel en sus capas convolucionales de dimensiones *5x5* y *7x7*

dimensiones de *7x7* sobre el conjunto de evaluación que utiliza el sistema urban se muestran en la tabla 4.11.

Para el sistema urban que utiliza matrices de Kernel en sus capas convolucionales con dimensiones de *5x5* se puede observar que se obtuvo una *ER* de *0.04* y una *F* de *97.15 %* basadas en segmentos de *1* segundo utilizando micropromediado, mientras que para el sistema urban que utiliza matrices de Kernel en sus capas convolucionales con dimensiones de *7x7* se obtuvo una *ER* de *0.04* y una *F* de *96.96 %* basadas en segmentos de *1* segundo utilizando micropromediado.

Estos sistema, al contener entrenarse utilizando mayores dimensiones de las matrices de kernel en sus capas convolucionales provoca que la red contenga un mayor número de parámetros que el sistema urban, siendo el número de parámetros entrenables de la red resulta ser de *892840* para el sistema urban que utiliza matrices de Kernel en sus capas convolucionales con dimensiones de *5x5* y de *1682344* para el sistema urban que utiliza matrices de Kernel en sus capas convolucionales con dimensiones de *7x7*, frente a los *366504* parámetros entrenables de la red que utiliza el sistema urban, lo que se traduce en un mayor coste computacional, y por tanto, en un mayor tiempo de entrenamiento del sistema.

Para los *2* sistemas entrenados (tanto el sistema urban que utiliza matrices de Kernel en sus capas convolucionales con dimensiones de *5x5*, como el sistema urban que utiliza matrices de Kernel en sus capas convolucionales con dimensiones de *7x7*), se obtienen mejores valores en las métricas de en comparación los obtenidos en el sistema urban, de una *ER* y una *F* basadas en segmentos de *1* segundo utilizando micropromediado de *0.05* y *96.51 %* respectivamente. Para el resto de métricas evaluadas, la gran mayoría de sus valores asociados a estos sistemas generalmente son mejores que las métricas obtenidas en el

sistema urban, por lo que se puede afirmar que aumentar el tamaño de las matrices de kernel en las capas convolucionales tomando dimensiones de *5x5* y *7x7* es una implementación que aporta mejores resultados en las métricas de los sistema urban entrenados utilizando estas características. Por otra parte, el sistema urban que utiliza matrices de Kernel en sus capas convolucionales con dimensiones de *5x5* ofrece aun mejor rendimiento que el sistema urban que utiliza matrices de Kernel en sus capas convolucionales con dimensiones de *7x7* en casi todas las métricas, por lo que, si hubiera que escoger entre realizar una implementación u otra, la mejor opción sería utilizar matrices de kernel en las capas convolucionales con dimensiones de *5x5* en la arquitectura del sistema urban.

4.9 Estudio del impacto de la función de activación en los bloques convolucionales

4.9.1 Uso de la función de activación ELU

Una de las implementaciones realizadas, tanto para el `sistema de referencia`, como para el `sistema urban`, consiste en utilizar la función de activación `ELU` en los bloques convolucionales en lugar de la función de activación ReLU que se usa de forma predeterminada, generando de esta manera, *2* nuevos sistemas.

Este implementación se realizó sustituyendo la función de activación ReLU por la función de activación ELU dentro de cada bloque convolucional definidos dentro de la función `get_model` de los scripts `sed.py` y `sed_urban.py`, utilizados en el `sistema de referencia` y el `sistema urban` respectivamente.

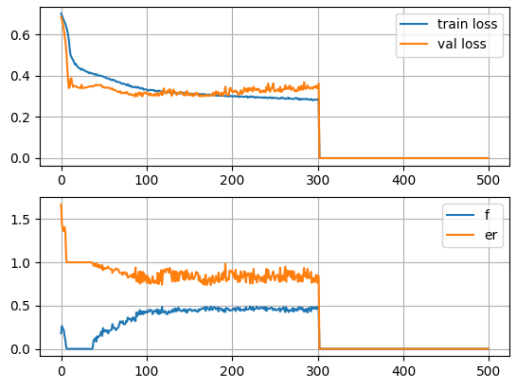
Tras realizar esta modificación en los archivos `sed.py` y `sed_urban.py`, los sistemas son entrenados para generar nuevos modelos; y estos se evalúan utilizando sus conjuntos de evaluación correspondientes, tal y como se evaluaron sus anteriores modelos sin modificaciones del `sistema de referencia` o `sistema urban`.

4.9.1.1 Resultados del sistema de referencia con activaciones ELU en sus bloques convolucionales

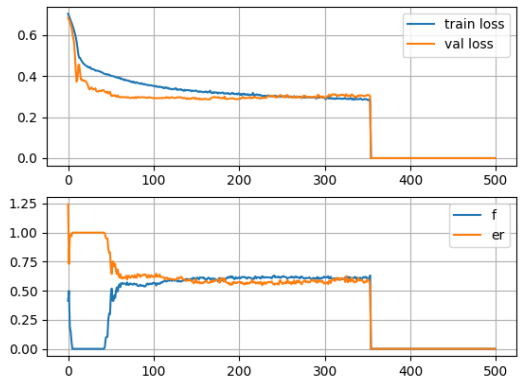
Durante el entrenamiento, los resultados obtenidos de `la tasa de error por segmento (ER)` y `la puntuación media (F)` o `F-score` calculada en segmentos de un segundo de duración utilizando micropromediado sobre el conjunto de validación promediado en cuatro pliegues fueron de una *ER* de *0.6274* y una *F* de *0.55*; valores bastante parejos a los obtenidos en el sistema de referencia mostrados en el apartado 4.3.3, los cuales, muestran una *ER* de *0.593* y una *F* de *0.5852*. En este caso, se obtiene un peor *ER* pero una mejor *F*.

En la figura 4.13 se pueden apreciar *2* gráficas para cada modelo de pliegue de validación cruzada que muestran las *pérdidas de entrenamiento y de validación*, al igual que *la puntuación media (F) y la tasa de error (ER) en segmentos de un segundo utilizando micropromediado del conjunto de validación* en las diferentes épocas del entrenamiento.

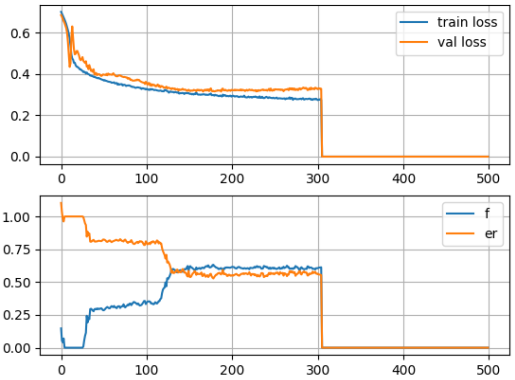
Los resultados obtenidos de las diferentes métricas medidas al evaluar las predicciones generadas por los distintos modelos de este sistema sobre el conjunto de evaluación de la base de datos `TUT Sound events 2017` (que utiliza el sistema de referencia de forma predeterminada) se muestran en la tabla 4.12.



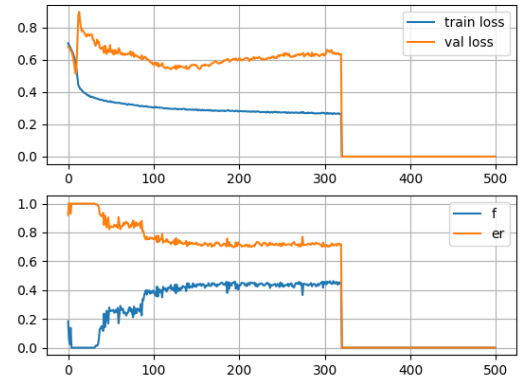
(a) Gráficas del entrenamiento del primer modelo de validación cruzada



(b) Gráficas del entrenamiento del segundo modelo de validación cruzada



(c) Gráficas del entrenamiento del tercer modelo de validación cruzada



(d) Gráficas del entrenamiento del cuarto modelo de validación cruzada

Figura 4.13: Gráficas de cada modelo de pliegue de validación cruzada del *sistema de referencia con activaciones ELU en sus bloques convolucionales* que muestran las *pérdidas de entrenamiento y de validación*, al igual que *la puntuación media (F) y la tasa de error (ER) en segmentos de un segundo evaluadas sobre el conjunto de validación*

Métricas basadas en segmentos	
Longitud evaluada	6898.28 sg
Archivos evaluados	4
Longitud del segmento	1 sg
Métricas generales (micropromedio)	
Puntuación media (F1)	39.61 %
Precisión	43.50 %
Recuperación	36.36 %
Tasa de error (ER)	0.89
Tasa de sustitución	0.22
Tasa de eliminación	0.42
Tasa de inserción	0.25
Sensibilidad	36.36 %
Especificidad	91.23 %
Exactitud equilibrada	63.79 %
Exactitud	82.63 %
Métricas promedio por clase (macropromedio)	
Puntuación media (F1)	23.28 %
Precisión	42.22 %
Recuperación	23.05 %
Tasa de error (ER)	1.34
Tasa de eliminación	0.77
Tasa de inserción	0.57
Sensibilidad	23.05 %
Especificidad	90.30 %
Exactitud equilibrada	56.67 %
Exactitud	82.63 %

Tabla 4.12: Métricas basadas en segmentos del sistema de referencia utilizando activaciones ELU en sus bloques convolucionales

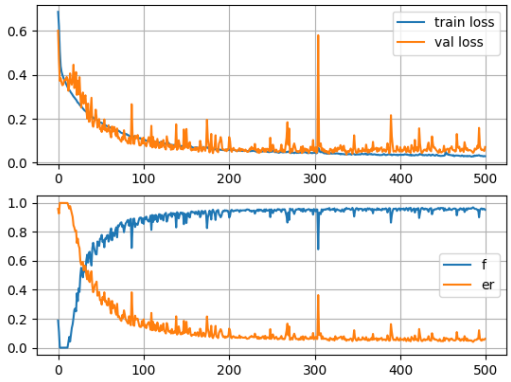
Como resultados, en este sistema se puede observar que se obtuvo una *ER* de *0.89* y una *F* de *39.61 %* basadas en segmentos de *1* segundo utilizando micropromediado. Estos resultados son peores que los obtenidos en el sistema de referencia evaluado con el conjunto de evaluación de la base de datos `TUT Sound events 2017`, mostrados en el apartado 4.3.3, en el que se obtuvieron una *ER* de *0.85* y una *F* de *41.95 %* basadas en segmentos de *1* segundo utilizando micropromediado.

Como este sistema utiliza activaciones ELU en sus bloques convolucionales en lugar de las activaciones ReLU, a pesar de que no haya un aumento de parámetros entrenables en el sistema, el coste computacional es superior debido a los cálculos de esta función y esto lleva a que el entrenamiento del sistema se demore, necesitando un mayor tiempo de entrenamiento.

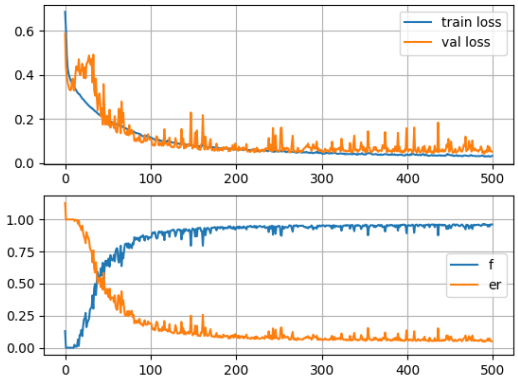
Al igual que ocurre con los valores de la *ER* y la *F* basadas en segmentos de *1* segundo utilizando micropromediado en comparación con las obtenidas en el sistema de referencia evaluado con el conjunto de evaluación de la base de datos `TUT Sound events 2017`, para el resto de métricas evaluadas, el sistema de referencia con activaciones ELU en sus bloques convolucionales presenta en su gran mayoría valores menos acertados. Por este motivo se puede llegar a la conclusión de que sustituir las activaciones ReLU por activaciones ELU en los bloques convolucionales no es una modificación en la arquitectura de la red que beneficie a la hora de conseguir un mejor rendimiento, o al menos, en el sistema de referencia que utiliza los datos de la base de datos `TUT Sound events 2017` para entrenarse.

4.9.1.2 Resultados del sistema urban con activaciones ELU en sus bloques convolucionales

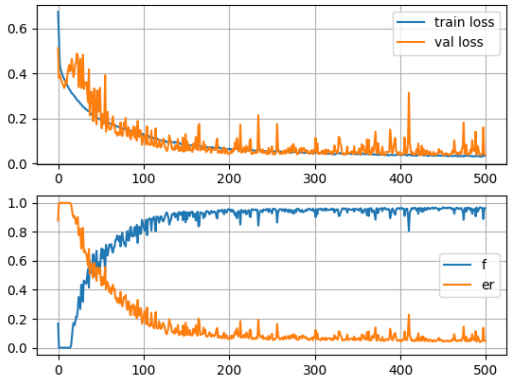
Durante el entrenamiento, los resultados obtenidos de `la tasa de error por segmento (ER)` y `la puntuación media (F) o F-score` calculada en segmentos de un segundo de duración utilizando micropromediado sobre el conjunto de validación promediado en cuatro pliegues fueron de una *ER*



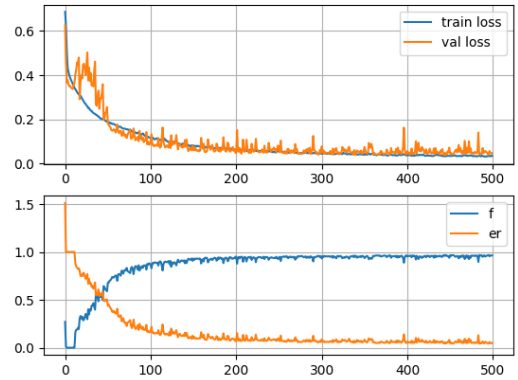
(a) Gráficas del entrenamiento del primer modelo de validación cruzada



(b) Gráficas del entrenamiento del segundo modelo de validación cruzada



(c) Gráficas del entrenamiento del tercer modelo de validación cruzada



(d) Gráficas del entrenamiento del cuarto modelo de validación cruzada

Figura 4.14: Gráficas de cada modelo de pliegue de validación cruzada del *sistema urban con activaciones ELU en sus bloques convolucionales* que muestran las *pérdidas de entrenamiento y de validación*, al igual que *la puntuación media (F) y la tasa de error (ER) en segmentos de un segundo evaluadas sobre el conjunto de validación*

de *0.0418* y una *F* de *0.9684*; valores muy similares a los obtenidos en el sistema urban mostrados en el apartado 4.5.2, los cuales, muestran una *ER* de *0.042* y una *F* de *0.9679*.

En la figura 4.14 se pueden apreciar *2* gráficas para cada modelo de pliegue de validación cruzada que muestran las *pérdidas de entrenamiento y de validación*, al igual que *la puntuación media (F) y la tasa de error (ER) en segmentos de un segundo utilizando micropromediado del conjunto de validación* en las diferentes épocas del entrenamiento.

Los resultados obtenidos de las diferentes métricas medidas al evaluar las predicciones generadas por los distintos modelos de este sistema sobre el conjunto de evaluación que utiliza el sistema urban se muestran en la tabla 4.13.

Como resultados, en este sistema se puede observar que se obtuvo una *ER* de *0.04* y una *F* de *96.85 %* basadas en segmentos de *1* segundo utilizando micropromediado. Estos resultados son muy parecidos a los obtenidos en el sistema urban evaluado, mostrados en el apartado 4.5.2, en el que se obtuvieron una

Métricas basadas en segmentos	
Longitud evaluada	18997.82 sg
Archivos evaluados	4
Longitud del segmento	1 sg
Métricas generales (micropromedio)	
Puntuación media (F1)	96.85 %
Precisión	96.39 %
Recuperación	97.32 %
Tasa de error (ER)	0.04
Tasa de sustitución	0.02
Tasa de eliminación	0.01
Tasa de inserción	0.02
Sensibilidad	97.32 %
Especificidad	99.34 %
Exactitud equilibrada	98.33 %
Exactitud	99.03 %
Métricas promedio por clase (macropromedio)	
Puntuación media (F1)	96.52 %
Precisión	96.21 %
Recuperación	96.83 %
Tasa de error (ER)	0.07
Tasa de eliminación	0.03
Tasa de inserción	0.04
Sensibilidad	96.83 %
Especificidad	99.32 %
Exactitud equilibrada	98.08 %
Exactitud	99.03 %

Tabla 4.13: Métricas basadas en segmentos del sistema urban utilizando activaciones ELU en sus bloques convolucionales

ER de *0.05* y una *F* de *96.51 %* basadas en segmentos de *1* segundo utilizando micropromediado, pero muestran una pequeña mejora para ambas métricas.

Como este sistema utiliza activaciones ELU en sus bloques convolucionales en lugar de las activaciones ReLU, a pesar de que no haya un aumento de parámetros entrenables en el sistema, el coste computacional es superior debido a los cálculos de esta función y esto lleva a que el entrenamiento del sistema se demore, necesitando un mayor tiempo de entrenamiento.

Como se puede observar las tablas de resultados 4.13, más allá de los valores de la *ER* y la *F* basadas en segmentos de *1* segundo utilizando micropromediado, casi todos los valores del resto de métricas son mejores en este nuevo sistema que en el sistema urban (cuyos resultados se pueden observar en el apartado 4.5.2) a excepción de alguna metricas que poseen los mismos valores en ambos sistemas. Con estos resultados se puede confirmar que para el sistema urban en concreto, el hecho de sustituir las activaciones ReLU por activaciones ELU en los bloques convolucionales mejora su rendimiento y es una buena implementación, siempre y cuando se esté dispuesto a asumir el aumento del coste computacional del entrenamiento de la red.

4.9.2 Uso de la función de activación PReLU

La última modificación que se realizó, tanto para el `sistema de referencia`, como para el `sistema urban`, consiste en utilizar la función de activación `PReLU` en los bloques convolucionales en lugar de la función de activación ReLU que se usa de forma predeterminada, generando de esta manera, *2* nuevos sistemas.

Este implementación se realizó sustituyendo la función de activación ReLU por la función de activación PReLU dentro de cada bloque convolucional definidos dentro de la función `get_model` de los

Métricas basadas en segmentos	
Longitud evaluada	6898.28 sg
Archivos evaluados	4
Longitud del segmento	1 sg
Métricas generales (micropromedio)	
Puntuación media (F1)	42.24 %
Precisión	43.76 %
Recuperación	40.83 %
Tasa de error (ER)	0.85
Tasa de sustitución	0.26
Tasa de eliminación	0.33
Tasa de inserción	0.26
Sensibilidad	40.83 %
Especificidad	90.19 %
Exactitud equilibrada	65.51 %
Exactitud	82.41 %
Métricas promedio por clase (macropromedio)	
Puntuación media (F1)	27.91 %
Precisión	44.76 %
Recuperación	30.81 %
Tasa de error (ER)	1.26
Tasa de eliminación	0.69
Tasa de inserción	0.57
Sensibilidad	30.81 %
Especificidad	89.51 %
Exactitud equilibrada	60.16 %
Exactitud	82.41 %

Tabla 4.14: Métricas basadas en segmentos del sistema de referencia utilizando activaciones PReLU en sus bloques convolucionales

scripts `sed.py` y `sed_urban.py`, utilizados en el `sistema de referencia` y el `sistema urban` respectivamente.

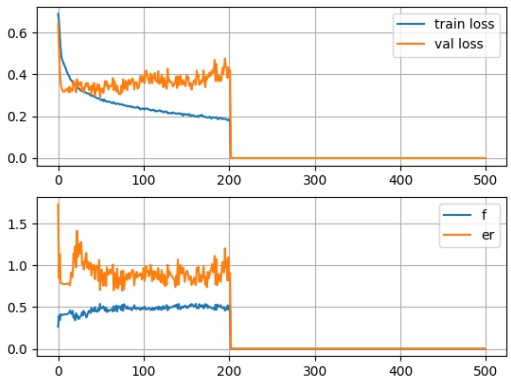
Tras realizar esta modificación en los archivos `sed.py` y `sed_urban.py`, los sistemas son entrenados para generar nuevos modelos; y estos se evalúan utilizando sus conjuntos de evaluación correspondientes, tal y como se evaluaron sus anteriores modelos sin modificaciones del `sistema de referencia` o `sistema urban`.

4.9.2.1 Resultados del sistema de referencia con activaciones PReLU en sus bloques convolucionales

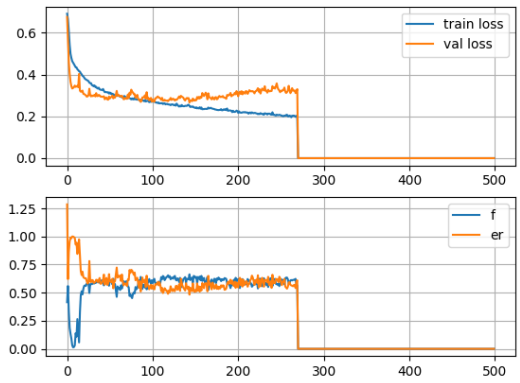
Durante el entrenamiento, los resultados obtenidos de `la tasa de error por segmento (`*ER*`)` y `la puntuación media (`*F*`) o F-score` calculada en segmentos de un segundo de duración utilizando micropromediado sobre el conjunto de validación promediado en cuatro pliegues fueron de una *ER* de *0.5894* y una *F* de *0.5907*; valores bastante parejos a los obtenidos en el sistema de referencia mostrados en el apartado 4.3.3, los cuales, muestran una *ER* de *0.593* y una *F* de *0.5852*.

En la figura 4.15 se pueden apreciar *2* gráficas para cada modelo de pliegue de validación cruzada que muestran las *pérdidas de entrenamiento y de validación*, al igual que *la puntuación media (F) y la tasa de error (ER) en segmentos de un segundo utilizando micropromediado del conjunto de validación* en las diferentes épocas del entrenamiento.

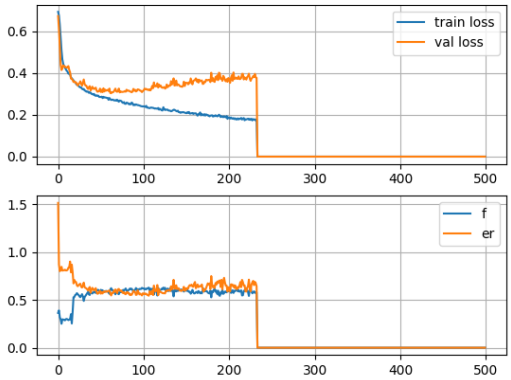
Los resultados obtenidos de las diferentes métricas medidas al evaluar las predicciones generadas por los distintos modelos de este sistema sobre el conjunto de evaluación de la base de datos `TUT Sound events 2017` (que utiliza el sistema de referencia de forma predeterminada) se muestran en la tabla 4.14.



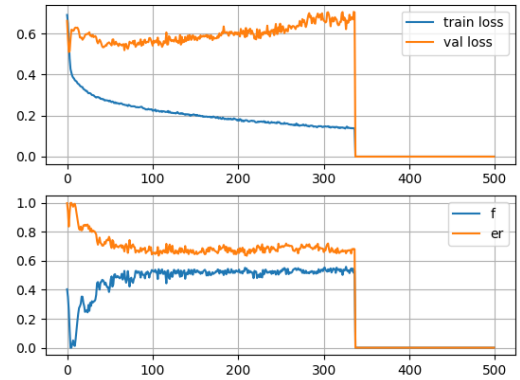
(a) Gráficas del entrenamiento del primer modelo de validación cruzada



(b) Gráficas del entrenamiento del segundo modelo de validación cruzada



(c) Gráficas del entrenamiento del tercer modelo de validación cruzada



(d) Gráficas del entrenamiento del cuarto modelo de validación cruzada

Figura 4.15: Gráficas de cada modelo de pliegue de validación cruzada del *sistema de referencia con activaciones PReLU en sus bloques convolucionales* que muestran las *pérdidas de entrenamiento y de validación*, al igual que *la puntuación media (F) y la tasa de error (ER) en segmentos de un segundo evaluadas sobre el conjunto de validación*

Como resultados, en este sistema se puede observar que se obtuvo una *ER* de *0.85* y una *F* de *42.24 %* basadas en segmentos de *1* segundo utilizando micropromediado. Estos resultados son muy similares a los obtenidos en el sistema de referencia evaluado con el conjunto de evaluación de la base de datos `TUT Sound events 2017`, mostrados en el apartado 4.3.3, en el que se obtuvieron una *ER* de *0.85* y una *F* de *41.95 %* basadas en segmentos de *1* segundo utilizando micropromediado.

Como este sistema utiliza activaciones PReLU en sus bloques convolucionales en lugar de las activaciones ReLU, se aumenta el número de parámetros entrenables de *366438* a *2070374* en el sistema. En comparación a cuando se utilizan las activaciones ReLU, el coste computacional es superior debido a los cálculos de esta función y esto lleva a que el entrenamiento del sistema se demore, necesitando un mayor tiempo de entrenamiento.

A pesar de que este sistema posea un valor de *F* basada en segmentos de *1* segundo utilizando micropromediado un poco mejor que el obtenido en el sistema de referencia evaluado con el conjunto de evaluación de la base de datos `TUT Sound events 2017`, el resto metricas poseen valores similares y es difícil determinar que sistema posee un mejor rendimiento en base a los resultados obtenidos. Como se ha comentado en oras ocasiones, dependiendo de la métrica a la que se le asigne más relevancia, a la hora de decidir entre implementar la modificación de sustituir las activaciones ReLU en los bloques convolucionalesde la red por activaciones PReLU o no hacerlo puede ser más sencillo. Para este caso, si se le asigna una prioridad absoluta a las métricas de *ER* y *F* basadas en segmentos de *1* segundo utilizando micropromediado sobre el resto de ellas, esta implementación puede llegar a ser una buena elección debido a su mejora en la *F*, pero tiene el inconveniente de la gran cantidad de parámetros que posee el sistema durante su entrenamiento.

4.9.2.2 Resultados del sistema urban con activaciones PReLU en sus bloques convolucionales

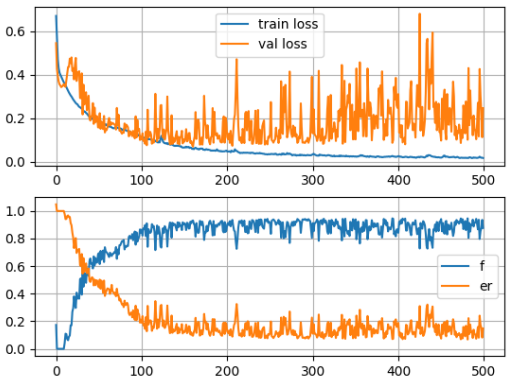
Durante el entrenamiento, los resultados obtenidos de `la tasa de error por segmento (ER)` y `la puntuación media (F) o F-score` calculada en segmentos de un segundo de duración utilizando micropromediado sobre el conjunto de validación promediado en cuatro pliegues fueron de una *ER* de *0.0608* y una *F* de *0.9531*; peores valores que los obtenidos en el sistema urban mostrados en el apartado 4.5.2, los cuales, muestran una *ER* de *0.042* y una *F* de *0.9679*.

En la figura 4.16 se pueden apreciar *2* gráficas para cada modelo de pliegue de validación cruzada que muestran las *pérdidas de entrenamiento y de validación*, al igual que *la puntuación media (F) y la tasa de error (ER) en segmentos de un segundo utilizando micropromediado del conjunto de validación* en las diferentes épocas del entrenamiento.

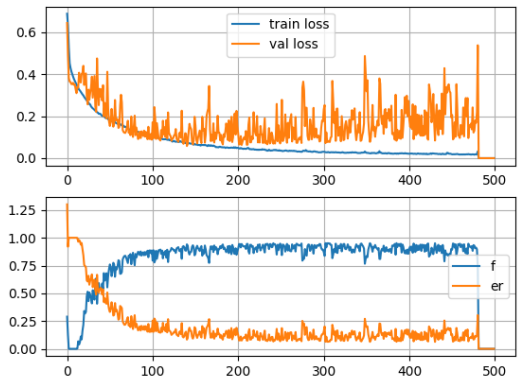
Los resultados obtenidos de las diferentes métricas medidas al evaluar las predicciones generadas por los distintos modelos de este sistema sobre el conjunto de evaluación que utiliza el sistema urban se muestran en la tabla 4.15.

Como resultados, en este sistema se puede observar que se obtuvo una *ER* de *0.06* y una *F* de *95.40 %* basadas en segmentos de *1* segundo utilizando micropromediado. Estos resultados son parecidos a los obtenidos en el sistema urban evaluado, mostrados en el apartado 4.5.2, en el que se obtuvieron una *ER* de *0.05* y una *F* de *96.51 %* basadas en segmentos de *1* segundo utilizando micropromediado, pero son peores en ambas métricas.

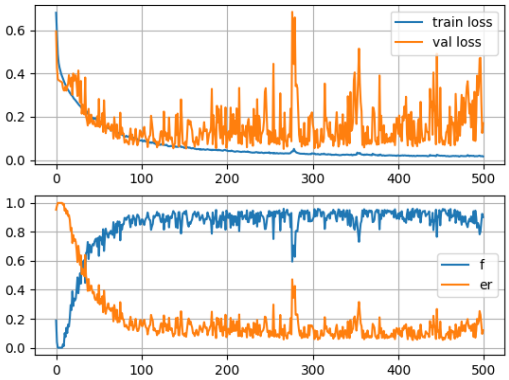
Como este sistema utiliza activaciones PReLU en sus bloques convolucionales en lugar de las activaciones ReLU, se aumenta el número de parámetros entrenables de *366438* a *2070440* en el sistema. En comparación a cuando se utilizan las activaciones ReLU, el coste computacional es superior debido a los



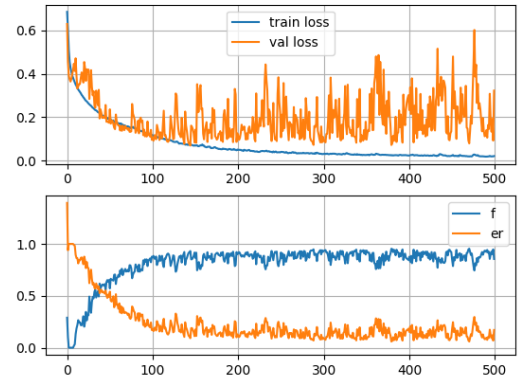
(a) Gráficas del entrenamiento del primer modelo de validación cruzada



(b) Gráficas del entrenamiento del segundo modelo de validación cruzada



(c) Gráficas del entrenamiento del tercer modelo de validación cruzada



(d) Gráficas del entrenamiento del cuarto modelo de validación cruzada

Figura 4.16: Gráficas de cada modelo de pliegue de validación cruzada del *sistema urban con activaciones PReLU en sus bloques convolucionales* que muestran las *pérdidas de entrenamiento y de validación*, al igual que *la puntuación media (F) y la tasa de error (ER) en segmentos de un segundo evaluadas sobre el conjunto de validación*

Métricas basadas en segmentos	
Longitud evaluada	18997.82 sg
Archivos evaluados	4
Longitud del segmento	1 sg
Métricas generales (micropromedio)	
Puntuación media (F1)	95.40 %
Precisión	94.88 %
Recuperación	95.93 %
Tasa de error (ER)	0.06
Tasa de sustitución	0.03
Tasa de eliminación	0.01
Tasa de inserción	0.02
Sensibilidad	95.93 %
Especificidad	99.06 %
Exactitud equilibrada	97.49 %
Exactitud	98.58 %
Métricas promedio por clase (macropromedio)	
Puntuación media (F1)	94.79 %
Precisión	94.76 %
Recuperación	94.90 %
Tasa de error (ER)	0.10
Tasa de eliminación	0.05
Tasa de inserción	0.05
Sensibilidad	94.90 %
Especificidad	99.03 %
Exactitud equilibrada	96.96 %
Exactitud	98.58 %

Tabla 4.15: Métricas basadas en segmentos del sistema urban utilizando activaciones PReLU en sus bloques convolucionales

cálculos de esta función y esto lleva a que el entrenamiento del sistema se demore, necesitando un mayor tiempo de entrenamiento.

Dado que este sistema posee peores valores en la mayoría de sus metricas, incluyendo la *ER* y la *F* basadas en segmentos de *1* segundo utilizando micropromediado, en comparación con el sistema urban y contiene un mayor coste computacional, se puede afirmar que el hecho de utilizar activaciones PReLU en sus bloques convolucionales en la arquitectura de la red que utiliza el sistema urban en lugar de las activaciones ReLU no ofrece mejores resultados, y por lo tanto, esta implementación no es beneficiosa.

4.10 Resumen de resultados relevantes

Como se ha comentado a lo largo de este capítulo, evaluar el rendimiento de un sistema depende de la relevancia que se le asigne a un conjunto de métricas que se tengan en consideración. En este apartado se pretende resumir los resultados obtenidos para todos los estudios realizados considerando únicamente algunas de las las métricas basadas en segmentos utilizando micropromediado, debido a que el sistema de referencia es entrenado optimizando los valores de la `tasa de error` y la `puntuación media` medidas en esas condiciones.

Estos resultados se encuentran en la tabla 4.16, en donde se muestra el valor de `puntuación media`, `precisión`, `recuperación` y `tasa de error`; indicadas como `F1`, `P`, `R` y `ER` respectivamente. Se ha decidido incluir las métricas de `precisión`, `recuperación` ademas de la `puntuación media` y `la tasa de error`, ya que estas tasas miden una relación directa entre el número de falsos positivos (por parte de la precisión) y falsos negativos (por parte de la recuperación) que se produce sobre el número total de eventos.

Como se puede observar en la tabla 4.16:

		Capas convolucionales		Tasa de dropout			Dimensiones del Kernel			Activación		
		Base	Adicional	0.3	**0.5**	0.7	**3x3**	5x5	7x7	ELU	**ReLU**	PreLU
Sistema de referencia	F1	**42.0 %**	43.5 %	44.8 %	**42.0 %**	44.2 %	**42.0 %**	45.6 %	42.3 %	39.6 %	**42.0 %**	42.2 %
	P	**44.6 %**	45.7 %	44.9 %	**44.6 %**	47.1 %	**44.6 %**	46.1 %	46.7 %	43.5 %	**44.6 %**	43.8 %
	R	**39.6 %**	41.6 %	44.6 %	**39.6 %**	41.6 %	**39.6 %**	45.1 %	38.6 %	36.44 %	**39.6 %**	40.8 %
	ER	**0.85**	0.87	0.87	**0.85**	0.86	**0.85**	0.85	0.83	0.89	**0.85**	0.85
Sistema urban	F1	**96.5 %**	97.4 %	97.1 %	**96.5 %**	86.6 %	**96.5 %**	97.2 %	97.0 %	96.9 %	**96.5 %**	95.4 %
	P	**96.1 %**	97.2 %	96.7 %	**96.1 %**	89.4 %	**96.1 %**	96.9 %	96.7 %	96.4 %	**96.1 %**	94.9 %
	R	**96.9 %**	97.6 %	97.5 %	**96.9 %**	83.9 %	**96.9 %**	97.4 %	97.2 %	97.3 %	**96.9 %**	95.9 %
	ER	**0.05**	0.03	0.04	**0.05**	0.19	**0.05**	0.04	0.04	0.04	**0.05**	0.06

Tabla 4.16: Métricas basadas en segmentos utilizando micropromediado obtenidas en las diferentes implementaciones

- Para el estudio del impacto al incrementar una capa convolucional adicional en cada bloque convolucional de la red neuronal utilizada, tanto en el sistema de referencia como el sistema urban se presenta una mejoría en todas las métricas adjuntas, a excepción de la *ER* que empeora su valor en esta implementación cuando se le aplica al sistema de referencia.

- Para el estudio del impacto al modificar el valor de la tasa de dropout en la arquitectura de la red neuronal utilizada, el sistema de referencia presenta mejores valores en las métricas *F1, P y R* cuando se utiliza una tasa de dropout de 0.7 y un mejor valor de *ER* para una tasa de dropout de 0,5 (valor predeterminado). En cuanto al sistema urban, se puede observar que todos los valores de las métricas seleccionadas son mas efectivos cuando se utiliza una tasa de dropout de 0.3.

- Para el estudio del impacto de modificar las dimensiones de las matrices de kernel en cada bloque convolucional, se obtienen una mejora en los valores de *F1, P y R* en el sistema de referencia que utiliza dimensiones de las matrices de kernel de *5x5* sin mejorar el valor de *ER* respecto a la arquitectura de la red neuronal utilizada sin modificaciones, mientras que se obtiene un mejor valor *ER* al utilizar dimensiones de *7x7*, empeorando el resto de métricas. En cambio, en lo que al sistema urban se refiere, se obtiene un mejor rendimiento en todas las métricas cuando se utilizan matrices de kernel de dimensión *5x5*.

- Para el estudio del impacto de utilizar diferentes activaciones en los bloques convolucionales de la red neuronal utilizada, se obtienen una mejoría en las métricas de *F1 y R* cuando se utilizan activaciones PReLU en el sistema de referencia, conservando el mismo valor de *ER* obtenido al utilizar activaciones ReLU de forma predeterminada en la arquitectura de la red neuronal utilizada. Por otro lado, para el sistema urban se obtienen mejoras en todas las métricas al utilizar activaciones ELU en los bloques convolucionales de la red neuronal.

Capítulo 5

Conclusiones y líneas futuras

En este capítulo se realiza un breve resumen sobre el trabajo realizado, las conclusiones obtenidas y se realizan varias propuestas orientadas a futuras líneas de investigación derivadas de este trabajo.

5.1 Conclusiones

Este trabajo de fin de grado ha consistido en el testeo y evaluación de un sistema de detección y clasificación de eventos sonoros al que se le han realizado diferentes implementaciones.

Como punto de partida se desarrolló un capítulo correspondiente a un estudio teórico donde se introducen los conceptos básicos necesarios utilizados a lo largo de este trabajo, en donde se dio una gran relevancia a las redes neuronales artificiales; una de las técnicas mas en auge dentro del campo del "Machine Learning" utilizando aprendizaje supervisado que están orientadas a la generación de modelos capaces de realizar predicciones sobre un conjunto de datos.

A continuación, durante una etapa de búsqueda de información, se recopilaron diferentes `bases de datos` de audio y `sistema de detección y clasificación de eventos sonoros`. En este apartado se incluyeron aquellas bases de datos que tuvieran cierta diversidad de clases de eventos sonoros con una cantidad suficiente de datos de audio para cada clase (a ser posible: sonidos cotidianos, urbanos o relacionados con la seguridad) o en su defecto, que tuvieran menos clases de eventos sonoros pero un gran número de datos de audio para cada una de ellas. En cuanto a los sistemas, se incluyeron aquellos sistemas eficientes relacionados con el reconocimiento y clasificación de diferentes eventos sonoros, de los cuales, sus autores hubieran publicado su código fuente de manera pública y fuera posible su uso para este trabajo.

De este conjunto de sistemas se seleccionó uno de ellos como el sistema de referencia a utilizar en este trabajo, del cual se comprobó su correcto funcionamiento y se evaluó con una "nueva base de datos" formada por archivos de audio de diferentes clases de eventos sonoros que el sistema de referencia es capaz de identificar, procedentes de algunas de las bases de datos recopiladas disponibles. Dado que esta "nueva" base de datos no contenía todos las clases de eventos sonoros que el sistema de referencia es capaz de detectar, las métricas obtenidas, a pesar de ser similares a las obtenidas con el conjunto de evaluación de la base de datos con la que se entrenaba el sistema, resultaron ser sesgadas y poco concluyentes como para poder afirmar que este sistema contiene la misma eficacia a la hora detectar eventos en otras bases de datos externas en lugar de los eventos pertenecientes al conjunto de evaluación de la base de datos con la que se entrena.

Este sistema de referencia también se entrenó y se evaluó utilizando los datos de *8* de las *10* clases de eventos sonoros que posee la base de datos "`UrbanSound8k`", apodando a este sistema como sistema urban, obteniendo una mejoría bastante significativa en los valores de las métricas, en comparación con el sistema de referencia entrenado con la base "`TUT Sound events 2017`" (base de datos que utiliza por defecto, en la que se distinguen *6* clases de eventos sonoros diferentes). Esto es atribuido a que el sistema posee un mayor número de datos para cada clase de eventos sonoros en sus conjuntos de entrenamiento y validación, y por otra parte, en sus distintos conjuntos de datos generados a partir de la base de datos "`UrbanSound8k`" (conjuntos de entrenamiento, validación y evaluación) no conciben la posibilidad de contener varias clases de eventos sonoros activas en un mismo instante de tiempo.

Para comprobar como afecta la alteración de ciertos términos en la arquitectura de la red neuronal empleada respecto al rendimiento final del sistema de referencia entrenado con la base "`TUT Sound events 2017`" y el sistema urban, se realizaron ciertas modificaciones. Estas modificaciones consistieron en: añadir una capa convolucional adicional en cada bloque convolucional, modificar el valor de la tasa de dropout en las conexiones de las neuronas entre capas, utilizar matrices de kernel de capa convolucional con diferentes dimensiones y sustituir las activaciones "ReLUs" de los bloques convolucionales por activaciones "PReLUs" y "ELUs". Cada modificación se aplicó de manera individual, por lo que el sistema de referencia evaluado con la base de datos que utiliza por defecto (TUT Sound events 2017) y el sistema urban se entrenaron nuevamente con cada modificación para posteriormente evaluar sus métricas y determinar (en el caso de que lo haya) si existe un beneficio en el rendimiento de un sistema u otro.

En cuanto a los resultados obtenidos, para las diferentes modificaciones de la arquitectura de la red neuronal, en el sistema de referencia entrenado con la base de datos que utiliza por defecto (TUT Sound events 2017), no se produjeron mejoras destacables en la mayoría de las métricas evaluadas, aunque para algunas modificaciones se obtuvieron mejoras puntuales en algunas métricas determinadas. Para estos sistemas en específico se ha determinado que dependiendo de la importancia que se le asigne a cada una de las métricas evaluadas, se podría llegar a considerar que el sistema puede o no presentar una mejora con las implementaciones realizadas.

El caso contrario ocurrió a la hora de evaluar el sistema urban realizando estas mismas modificaciones en la arquitectura de la red utilizada. Si bien, determinar si existe una mejoría con la implementación de algunas modificaciones es una tarea comprometida, tal y como ocurría con algunas modificaciones realizadas con el sistema de referencia; como regla general, cuando se consideró que una modificación mejoraba o empeoraba el rendimiento del sistema, era debido a que la gran mayoría de sus métricas presentaban una mejora o viceversa respecto a los resultados obtenidos en el sistema urban sin realizar modificaciones en la arquitectura de su red neuronal.

5.2 Líneas futuras

Para concluir, en este último apartado se plantea una serie de lineas futuras derivadas de este TFG:

- Estudio de un sistema de detección y clasificación de eventos sonoros entrenado para optimizar la tasa de error (ER) y la puntuación F o F-score basadas en segmentos utilizando macro-promediado, para evitar de esta manera el desequilibrio de rendimiento entre las diferentes clases. En esta línea futura se podría entrenar un sistema con estas características y comparar los resultados de sus métricas con el mismo sistema entrenado optimizando la tasa de error (ER) y la puntuación F o F-score basadas en segmentos utilizando micro-promediado (como el sistema de referencia utilizado en este trabajo).

- Estudio de compare el rendimiento de un sistema de detección y clasificación de eventos sonoros utilizando diferentes técnicas de pre-procesado de audio. Como puede ser para un sistema que utilice una CRNN el hecho de utilizar: espectrogramas, coeficientes cepstrales de frecuencia Mel, espectrogramas de Mel o espectrogramas de log-Mel. Se podría realizar una comparativa de las métricas obtenidas para cada una de las técnicas empleadas remuestreando los diferentes archivos de audio con los que se trabaja para varias frecuencias de muestreo, entre otras modificaciones.

- Estudio sobre un sistema de detección y clasificación de eventos sonoros más exhaustivo, que involucre una mayor implicación de modificación de parámetros pertenecientes a la arquitectura de su red, en el que se realice una evaluación del sistema aplicando todas las modificaciones que involucren un mayor rendimiento del sistema de forma conjunta.

- Estudio detallado sobre un sistema de detección de eventos multimedia orientados a la detección de anomalías o situaciones peligrosas, donde el audio suele usarse como fuente complementaria de información.

- Empleo de transformadores, introducidos en [159], como técnica novedosa de aprendizaje profundo aplicada en sistemas de detección y clasificación de eventos sonoros, tal y como se ha aplicado en [160], utilizado en un sistema de detección y separación de eventos de sonido en entornos domésticos. Los transformadores poseen una arquitectura de red neuronal específica diseñada para manipular datos secuenciales, principalmente en el campo del procesamiento de lenguaje natural (como puede ser para trabajos orientados a la traducción, predicción de texto, sumarización, etc) o bien en el campo de las series de tiempo o la previsión, entre otros. Esta técnica se encuentra completamente en auge y son las encargadas de desbancar a las redes RNN, debido a que las RNN tienen una memoria limitada a corto plazo.

Bibliografía

[1] E. Franco and R. Ramos, "Aprendizaje de máquina y aprendizaje profundo en biotecnología: aplicaciones, impactos y desafíos," *Ciencia, Ambiente y Clima*, vol. 2, pp. 7–26, 12 2019.

[2] S. Kahl, H. Hussein, E. Fabian, J. Schloßhauer, E. Thangaraju, D. Kowerko, and M. Eibl, "Acoustic event classification using convolutional neural networks," *INFORMATIK 2017*, 2017.

[3] S. Adavanne, A. Politis, J. Nikunen, and T. Virtanen, "Sound event localization and detection of overlapping sources using convolutional recurrent neural networks," *IEEE Journal of Selected Topics in Signal Processing*, vol. 13, no. 1, 2018.

[4] Y. Cao, T. Iqbal, Q. Kong, M. Galindo, W. Wang, and M. D. Plumbley, "Two-stage sound event localization and detection using intensity vector and generalized cross-correlation," *Tech. report of Detection and Classification of Acoustic Scenes and Events 2019 (DCASE) Challange*, 2019.

[5] Y. Cohen, D. A. Nicholson, and T. J. Gardner, "Tweetynet: A neural network that enables high-throughput, automated annotation of birdsong," *bioRxiv*, 2020.

[6] Y. Xu, Q. Kong, W. Wang, and M. D. Plumbley, "Large-scale weakly supervised audio classification using gated convolutional neural network," 2017.

[7] S. Adavanne and T. Virtanen, "A report on sound event detection with different binaural features," 2017.

[8] S. Aarthi and S. Chitrakala, "Scene understanding - a survey," in *2017 International Conference on Computer, Communication and Signal Processing (ICCCSP)*. IEEE, 2017, pp. 1–4.

[9] T. Virtanen, M. D. Plumbley, and D. Ellis, *Computational analysis of sound scenes and events*. Springer, 2018.

[10] S. Gharib, H. Derrar, D. Niizumi, T. Senttula, J. Tommola, T. Heittola, T. Virtanen, and H. Huttunen, "Acoustic scene classification: A competition review," in *2018 IEEE 28th International Workshop on Machine Learning for Signal Processing (MLSP)*. IEEE, 2018, pp. 1–6.

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

[12] D. Stowell, D. Giannoulis, E. Benetos, M. Lagrange, and M. D. Plumbley, "Detection and classification of acoustic scenes and events," *IEEE Transactions on Multimedia*, vol. 17, no. 10, pp. 1733–1746, 2015.

[13] E. B. D. Wang, G. J. Brown, and C. Darwin, "Computational auditory scene analysis: Principles, algorithms and applications," *Acoustical Society of America Journal*, vol. 124, p. 13, 2008.

[14] J. Torres and T. Torres, *DEEP LEARNING Introducción Práctica con Keras*, ser. Watch This Space Series. Independently Published, 2018. [Online]. Available: https://books.google.es/books?id=E696uQEACAAJ

[15] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, 1943.

[16] S. Haykin, *Neural Networks and Learning Machines*, ser. Neural networks and learning machines. Prentice Hall, 2009, no. v. 10. [Online]. Available: https://books.google.es/books?id=K7P36lKzI_QC

[17] J. L. McClelland, D. E. Rumelhart, P. R. Group *et al.*, *Parallel distributed processing*. MIT press Cambridge, MA, 1986, vol. 2.

[18] D. J. Matich, "Redes neuronales: Conceptos básicos y aplicaciones," *Universidad Tecnológica Nacional, México*, vol. 41, 2001.

[19] P. Larranaga, I. n. Inza, and A. Moujahid, "Redes neuronales: Conceptos básicos y aplicaciones," 07 2021.

[20] "Cómo elegir una función de activación para el aprendizaje profundo," https://topbigdata.es/como-elegir-una-funcion-de-activacion-para-el-aprendizaje-profundo/ [Último acceso 20/julio/2021].

[21] "Una introducción práctica a sigmoid, tanh, relu, leaky relu, prelu, elu y selu," https://ichi.pro/es/7-funciones-de-activacion-populares-que-debes-conocer-en-deep-learning-y-como-usarlas-con-keras-y-tensorflow-2- [Último acceso 21/julio/2021].

[22] "Machine learning combat 11- entrenamiento de la red neuronal profunda," https://programmerclick.com/article/9603762818/ [Último acceso 20/julio/2021].

[23] Y. LeCun, "Funciones de activación y de costo," https://atcold.github.io/pytorch-Deep-Learning/es/week11/11-1/ [Último acceso 20/julio/2021].

[24] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (elus)," *arXiv preprint arXiv:1511.07289*, 2015.

[25] S. Polamuri, "Difference between softmax function and sigmoid function," https://dataaspirant.com/difference-between-softmax-function-and-sigmoid-function/ [Último acceso 24/julio/2021].

[26] "Mejora de una red neuronal artificial con regularización y optimización," https://ichi.pro/es/mejora-de-una-red-neuronal-artificial-con-regularizacion-y-optimizacion-79834580974110/ [Último acceso 26/julio/2021].

[27] "Explicación sobre el tamaño de lote," https://blog.csdn.net/qq_20259459/article/details/53943413/ [Último acceso 5/agosto/2021].

[28] "¿Qué es el tamaño del lote en la red neuronal?" https://qastack.mx/stats/153531/what-is-batch-size-in-neural-network/ [Último acceso 5/agosto/2021].

[29] J. Brownlee, "A gentle introduction to batch normalization for deep neural networks," %https://machinelearningmastery.com/batch-normalization-for-training-of-deep-neural-networks/ [Último acceso 5/agosto/2021].

[30] S. Sra, S. Nowozin, and S. J. Wright, *Optimization for machine learning*. Mit Press, 2012.

[31] A. V. Srinivasan, "Stochastic gradient descent," https://towardsdatascience.com/stochastic-gradient-descent-clearly-explained-53d239905d31/ [Último acceso 25/julio/2021].

[32] I. G. Gavilán, "Catálogo de componentes de redes neuronales (y iv): optimizadores," https://ignaciogavilan.com/catalogo-de-componentes-de-redes-neuronales-y-iv-optimizadores/ [Último acceso 25/julio/2021].

[33] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization." *Journal of machine learning research*, vol. 12, no. 7, 2011.

[34] M. D. Zeiler, "Adadelta: an adaptive learning rate method," *arXiv preprint arXiv:1212.5701*, 2012.

[35] T. Tieleman and G. Hinton, "Root mean square propagation. divide the gradient by a running average of its recent magnitude. coursera: Neural networks for machine learning," 2012.

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

[37] A. Vannieuwenhuyze, *Inteligencia artificial fácil - Machine Learning y Deep Learning prácticos*. Eni, 2020.

[38] "Aprendizaje automatico y las metricas de regresión," https://sitiobigdata.com/2018/08/27/machine-learning-metricas-regresion-mse/ [Último acceso 24/julio/2021].

[39] C. Olah, "Neural networks, manifolds, and topology," http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/ [Último acceso 24/julio/2021].

[40] D. Godoy, "Understanding binary cross-entropy / log loss: a visual explanation," https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a/ [Último acceso 24/julio/2021].

[41] "Función de pérdida de entropía cruzada," https://ichi.pro/es/funcion-de-perdida-de-entropia-cruzada-267783942726718/ [Último acceso 24/julio/2021].

[42] F. Ramírez, "Las matemáticas del machine learning: Funciones de activación," https://empresas.blogthinkbig.com/las-matematicas-del-machine-learning-funciones-de-activacion/ [Último acceso 24/julio/2021].

[43] Á. Gonzalo, "¿Qué es el sobreajuste u overfitting y por qué debemos evitarlo?" https://machinelearningparatodos.com/que-es-el-sobreajuste-u-overfitting-y-por-que-debemos-evitarlo/ [Último acceso 26/julio/2021].

[44] J. M. Heras, "Regularización lasso l1, ridge l2 y elasticnet," https://www.iartificial.net/regularizacion-lasso-l1-ridge-l2-y-elasticnet/ [Último acceso 26/julio/2021].

[45] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[46] "Understanding lstm networks," http://colah.github.io/posts/2015-08-Understanding-LSTMs/ [Último acceso 29/julio/2021].

[47] "Intro to recurrent neural networks, lstm and gru," https://www.kaggle.com/thebrownviking20/intro-to-recurrent-neural-networks-lstm-gru/ [Último acceso 29/julio/2021].

[48] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[49] M. A. Nielsen, *Neural networks and deep learning*. Determination press San Francisco, CA, 2015, vol. 25.

[50] "Convolución," https://datascience.eu/es/matematica-y-estadistica/convolucion/ [Último acceso 31/julio/2021].

[51] "¿Cómo funcionan las convolutional neural networks? visión por ordenador," https://www.aprendemachinelearning.com/como-funcionan-las-convolutional-neural-networks-vision-por-ordenador/ [Último acceso 31/julio/2021].

[52] "Convolutional neural networks (cnns / convnets)," https://cs231n.github.io/convolutional-networks/ [Último acceso 29/julio/2021].

[53] J. Vieco, "Red convolucional con pytorch," https://cleverpy.com/red-convolucional-pytorch/ [Último acceso 31/julio/2021].

[54] "Red neuronal convolucional, que se divide principalmente en cuatro bloques principales," https://programmerclick.com/article/9763840912/ [Último acceso 31/julio/2021].

[55] C. C. Chatterjee, "An approach towards convolutional recurrent neural networks," https://towardsdatascience.com/an-approach-towards-convolutional-recurrent-neural-networks-a2e6ce722b19/ [Último acceso 6/agosto/2021].

[56] "Muestreo digital," https://es.wikipedia.org/wiki/Muestreo_digital/ [Último acceso 31/julio/2021].

[57] "Cuantificación digital," https://es.wikipedia.org/wiki/Cuantificaci%C3%B3n_digital/ [Último acceso 31/julio/2021].

[58] "Conceptos básicos de sonido digital," http://formacion.intef.es/pluginfile.php/42889/mod_imscp/content/1/conceptos_bsicos_del_sonido_digital.html/ [Último acceso 31/julio/2021].

[59] "Greenfacts, basado en eu-osha: Decibelio," https://ec.europa.eu/health/scientific_committees/opinions_layman/es/perdida-audicion-reproductores-musica-mp3/glosario/def/decibelio.htm/ [Último acceso 31/julio/2021].

[60] "Cuartoinformática: Conceptos básicos de sonido digital," http://cuartoinformatica.tecnojulio.com/audio-digital/ [Último acceso 31/julio/2021].

[61] F. Aguirre Martín *et al.*, "Desarrollo y análisis de clasificadores de señales de audio," Tesis de máster, Escuela Politécnica Superior de Gandía, Universidad Politécnica de Valencia, 2017.

[62] G. A. M. Mascorro and G. A. Torres, "Reconocimiento de voz basado en mfcc, sbc y espectrogramas," *Ingenius*, no. 10, pp. 12–20, 2013.

[63] "Espectrograma," https://es.wikipedia.org/wiki/Espectrograma/ [Último acceso 31/julio/2021].

[64] C. R. Llorente, "Diseño, implementación y evaluación de técnicas de identificación de emociones a través de la voz," Trabajo de fin de grado, ETSIT, UPM, 2007.

[65] "Getting to know the mel spectrogram," https://towardsdatascience.com/getting-to-know-the-mel-spectrogram-31bca3e2d9d0/ [Último acceso 30/agosto/2021].

[66] S. Davis and P. Mermelstein, "Ieee transactions on acoustics, speech, and signal processing, 28 (4)," *IEEE Transactions on Acoustics, Speech, and Signal Processing, 28 (4)*, pp. 357–366, 1980.

[67] "Mel frequency cepstral coefficients," https://es.wikipedia.org/wiki/MFCC/ [Último acceso 26/julio/2021].

[68] D. B. Laynez, "Sistemas de verificación automática de locutor," Trabajo de fin de grado, ETSI, US, 2012.

[69] K. Venkataramanan and H. R. Rajamohan, "Emotion recognition from speech," 2019.

[70] G. van Rossum, "Python," https://www.python.org/, 1991.

[71] "¿Qué es python?" https://elpythonista.com/python/ [Último acceso 22/agosto/2021].

[72] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[73] "Tensorflow para principiantes," https://www.apsl.net/blog/2017/12/05/tensor-flow-para-principiantes-i/ [Último acceso 22/agosto/2021].

[74] F. Chollet *et al.*, "Keras," https://keras.io, 2015.

[75] "Piperlab: Keras," https://piperlab.es/glosario-de-big-data/keras/ [Último acceso 22/agosto/2021].

[76] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[77] T. Oliphant *et al.*, "Numpy," https://numpy.org/, 2006.

[78] J. Hunter *et al.*, "Matplotlib," https://matplotlib.org/, 2012.

[79] B. McFee, C. Raffel, D. Liang, D. Ellis, M. McVicar, E. Battenberg, and O. Nieto, "Librosa: Audio and music signal analysis in python," https://librosa.org/, 2015.

[80] "Informatik 2017, ws34 deep learning in heterogenen datenbeständen," https://informatik2017.de/ws34-dlhd/ [Último acceso 31/julio/2021].

[81] U. o. M. Montreal Institute for Learning Algorithms (MILA), "Theano," http://www.deeplearning.net/software/theano/, 2007.

[82] E. Battenberg, S. Dieleman, D. Nouri, E. Olson, A. van den Oord, C. Raffel, J. Schlüter, and S. K. Sønderby, "Lasagne," https://lasagne.readthedocs.io/, 2014.

[83] I. Intel Corporation, Willow Garage, "Opencv," https://opencv.org/, 2000.

[84] "Acoustic event detection using recurrent neural networks," https://github.com/misskaseyann/acoustic-event-detection/ [Último acceso 31/julio/2021].

[85] "Urbansound8k dataset," https://urbansounddataset.weebly.com/urbansound8k.html/ [Último acceso 8/agosto/2021].

[86] "Freesound," https://www.freesound.org/ [Último acceso 31/julio/2021].

[87] W. McKinney *et al.*, "Pandas," https://pandas.pydata.org/, 2008.

[88] M. L. Waskom *et al.*, "Seaborn: statistical data visualization," https://seaborn.pydata.org/, 2008.

[89] S. Adavanne, A. Politis, and T. Virtanen, "Sound event localization, detection, and tracking of multiple overlapping stationary and moving sources using convolutional recurrent neural network," https://github.com/sharathadavanne/seld-net/ [Último acceso 7/agosto/2021].

[90] ——, "Localization, detection and tracking of multiple moving sound sources with a convolutional recurrent neural network," *arXiv preprint arXiv:1904.12769*, 2019.

[91] "Detection and classification of acoustic scenes and events (dcase)," http://dcase.community/ [Último acceso 2/agosto/2021].

[92] "Challenge dcase 2019 task 3: Sound event localization and detection," http://dcase.community/challenge2019/task-sound-event-localization-and-detection/ [Último acceso 2/agosto/2021].

[93] "Challenge dcase 2020 task 3: Sound event localization and detection," http://dcase.community/challenge2020/task-sound-event-localization-and-detection/ [Último acceso 2/agosto/2021].

[94] S. Adavanne, A. Politis, and T. Virtanen, "Tau spatial sound events 2019 - ambisonic and microphone array, development datasets," https://zenodo.org/record/2580091#.YQ7Gg4gzbIU/ [Último acceso 7/agosto/2021].

[95] ——, "Tau spatial sound events 2019 - ambisonic and microphone array, evaluation datasets," https://zenodo.org/record/3377088#.YQ7HbIgzbIU/ [Último acceso 7/agosto/2021].

[96] M. Smales, "Sound classification using deep learning," https://mikesmales.medium.com/sound-classification-using-deep-learning-8bc2aa1990b7/ [Último acceso 7/agosto/2021].

[97] "Udacity," https://www.udacity.com/course/machine-learning-engineer-nanodegree--nd009t/ [Último acceso 31/julio/2021].

[98] K. Drossos, S. I. Mimilakis, S. Gharib, Y. Li, and T. Virtanen, "Sound event detection with depthwise separable and dilated convolutions," 2020.

[99] "Tut-sed synthetic 2016: Synthetic dataset for sound event detection research," https://webpages.tuni.fi/arg/paper/taslp2017-crnn-sed/tut-sed-synthetic-2016/ [Último acceso 8/agosto/2021].

[100] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[101] E. Cakir, G. Parascandolo, T. Heittola, H. Huttunen, and T. Virtanen, "Convolutional recurrent neural networks for polyphonic sound event detection," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 25, no. 6, pp. 1291–1303, Jun 2017. [Online]. Available: http://dx.doi.org/10.1109/TASLP.2017.2690575

[102] A. Gorin, N. Makhazhanov, and N. Shmyrev, "Dcase 2016 sound event detection system based on convolutional neural network," *IEEE AASP Challenge: Detection and Classification of Acoustic Scenes and Events*, 2016.

[103] "Challenge dcase 2016 task 3: Sound event detection in real life audio," http://dcase.community/challenge2016/task-sound-event-detection-in-real-life-audio/ [Último acceso 2/agosto/2021].

[104] A. Mesaros, T. Heittola, and T. Virtanen, "TUT database for acoustic scene classification and sound event detection," in *24th European Signal Processing Conference 2016 (EUSIPCO 2016)*, Budapest, Hungary, 2016.

[105] "Tut sound events 2016, development dataset," https://zenodo.org/record/45759#.YRZkZogzbIU/ [Último acceso 13/agosto/2021].

[106] "Tut sound events 2016, evaluation dataset," https://zenodo.org/record/996424#.YRZknIgzbIU/ [Último acceso 13/agosto/2021].

[107] "Acoustic event detection of general sounds," https://www.spsc.tugraz.at/student-projects/Pernkopf12.html/ [Último acceso 7/agosto/2021].

[108] "Challenge dcase 2017 task 4: Large-scale weakly supervised sound event detection for smart cars," http://dcase.community/challenge2017/task-large-scale-sound-event-detection/ [Último acceso 2/agosto/2021].

[109] "Audioset ontology," http://research.google.com/audioset/ontology/index.html/ [Último acceso 8/julio/2021].

[110] M. Dorfer and G. Widmer, "Training general-purpose audio tagging networks with noisy labels and iterative self-verification," in *Proceedings of the Detection and Classification of Acoustic Scenes and Events 2018 Workshop (DCASE2018)*, 2018, pp. 178–182.

[111] "Challenge dcase 2018 task 2: General-purpose audio tagging of freesound content with audioset labels," http://dcase.community/challenge2018/task-general-purpose-audio-tagging/ [Último acceso 11/agosto/2021].

[112] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015.

[113] T. Iqbal, Q. Kong, M. D. Plumbley, and W. Wang, "General-purpose audio tagging from noisy labels using convolutional neural networks," in *Proceedings of the Detection and Classification of Acoustic Scenes and Events 2018 Workshop (DCASE2018*. Tampere University of Technology, 2018, pp. 212–216.

[114] Y. N. Dauphin, A. Fan, M. Auli, and D. Grangier, "Language modeling with gated convolutional networks," in *International conference on machine learning*. PMLR, 2017, pp. 933–941.

[115] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz, "mixup: Beyond empirical risk minimization," 2018.

[116] I.-Y. Jeong and H. Lim, "Audio tagging system for dcase 2018: focusing on label noise, data augmentation and its efficient learning," *Tech. Rep., DCASE Challenge*, 2018.

[117] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 2261–2269.

[118] J. Hu, L. Shen, S. Albanie, G. Sun, and E. Wu, "Squeeze-and-excitation networks," 2019.

[119] "Challenge dcase 2017 task 3: Sound event detection in real life audio," http://dcase.community/challenge2017/task-sound-event-detection-in-real-life-audio/ [Último acceso 2/agosto/2021].

[120] "Tut sound events 2017, development dataset," https://zenodo.org/record/814831#.YRgBtogzbIU/ [Último acceso 13/agosto/2021].

[121] "Tut sound events 2017, evaluation dataset," https://zenodo.org/record/1040179#.YRgBt4gzbIU/ [Último acceso 13/agosto/2021].

[122] "Esc-50: Dataset for environmental sound classification," https://github.com/karolpiczak/ESC-50/ [Último acceso 8/agosto/2021].

[123] K. J. Piczak, "ESC: Dataset for Environmental Sound Classification," in *Proceedings of the 23rd Annual ACM Conference on Multimedia*. ACM Press, 2015, pp. 1015–1018. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2733373.2806390

[124] "Challenge dcase 2016 task 2: Sound event detection in synthetic audio," http://dcase.community/challenge2016/task-sound-event-detection-in-synthetic-audio/ [Último acceso 10/agosto/2021].

[125] "Challenge dcase 2016: task 2 train/development datasets," https://archive.org/details/dcase2016_task2_train_dev/ [Último acceso 10/agosto/2021].

[126] "Challenge dcase 2016: task 2 test datasets," https://archive.org/details/dcase2016_task2_test_public/ [Último acceso 10/agosto/2021].

[127] "Sound ideas," https://www.sound-ideas.com/ [Último acceso 31/julio/2021].

[128] "Tut acoustic scenes 2016, development dataset," https://zenodo.org/record/45739#.YRZlV4gzbIU/ [Último acceso 13/agosto/2021].

[129] "Tut acoustic scenes 2016, evaluation dataset," https://zenodo.org/record/165995#.YRZlWIgzbIU/ [Último acceso 13/agosto/2021].

[130] "Tut acoustic scenes 2017, development dataset," https://zenodo.org/record/400515#.YRgCTIgzbIU/ [Último acceso 13/agosto/2021].

[131] "Tut acoustic scenes 2017, evaluation dataset," https://zenodo.org/record/1040168#.YRgCTYgzbIU/ [Último acceso 13/agosto/2021].

[132] E. Fonseca, M. Plakal, D. P. Ellis, F. Font, X. Favory, and X. Serra, "Learning sound event classifiers from web audio with noisy labels," in *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2019, pp. 21–25.

[133] "Marsyas: Music analysis, retrieval and synthesis for audio signals," http://marsyas.info/downloads/datasets.html/ [Último acceso 8/agosto/2021].

[134] G. Tzanetakis and P. Cook, "Musical genre classification of audio signals," *IEEE Transactions on speech and audio processing*, vol. 10, no. 5, pp. 293–302, 2002.

[135] K. Drossos, S. Lipping, and T. Virtanen, "Clotho: An audio captioning dataset," 2019.

[136] "Amazon mechanical turk," https://www.mturk.com/ [Último acceso 8/agosto/2021].

[137] D. Stowell and M. D. Plumbley, "An open dataset for research on audio field recording archives: freefield1010," 2013.

[138] "Freesound loops 4k," https://zenodo.org/record/3685832#.YRAnwYgzbIU/ [Último acceso 8/agosto/2021].

[139] F. Font Corbera and X. Serra, "Tempo estimation for music loops and a simple confidence measure," in *Devaney J, Mandel MI, Turnbull D, Tzanetakis G, editors. ISMIR 2016. Proceedings of the 17th International Society for Music Information Retrieval Conference; 2016 Aug 7-11; New York City (NY).[Canada]: ISMIR; 2016. p. 269-75.* International Society for Music Information Retrieval (ISMIR), 2016.

[140] "Freesound one-shot percussive sounds," https://zenodo.org/record/3665275#.YRAttYgzbIV/ [Último acceso 8/agosto/2021].

[141] A. Ramires, P. Chandna, X. Favory, E. Gómez, and X. Serra, "Neural percussive synthesis parameterised by high-level timbral features," in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2020, pp. 786–790.

[142] "Simscenetvb learning," https://zenodo.org/record/3248703#.YRAwu4gzbIW/ [Último acceso 8/agosto/2021].

[143] "Librispeech," http://www.openslr.org/12/ [Último acceso 8/julio/2021].

[144] "simscene," https://bitbucket.org/mlagrange/simscene/src/master/ [Último acceso 31/julio/2021].

[145] "Sound events for surveillance applications," https://zenodo.org/record/3519845#.YRBk6ogzbIX/ [Último acceso 8/agosto/2021].

[146] E. Fonseca, X. Favory, J. Pons, F. Font, and X. Serra, "Fsd50k: an open dataset of human-labeled sound events," *arXiv preprint arXiv:2010.00475*, 2020.

[147] M. Cartwright, J. Cramer, A. E. M. Mendez, Y. Wang, H.-H. Wu, V. Lostanlen, M. Fuentes, G. Dove, C. Mydlarz, J. Salamon, O. Nov, and J. P. Bello, "Sonyc-ust-v2: An urban sound tagging dataset with spatiotemporal context," 2020.

[148] "Zooniverse," https://www.zooniverse.org/ [Último acceso 8/agosto/2021].

[149] "Department of environmental protection - nyc," https://www1.nyc.gov/site/dep/index.page/ [Último acceso 8/agosto/2021].

[150] E. Fonseca, M. Plakal, F. Font, D. P. W. Ellis, X. Favory, J. Pons, and X. Serra, "General-purpose tagging of freesound audio with audioset labels: Task description, dataset, and baseline," 2018.

[151] "Freesound dataset," https://annotator.freesound.org/fsd/ [Último acceso 31/julio/2021].

[152] E. Fonseca, M. Plakal, F. Font, D. P. W. Ellis, and X. Serra, "Audio tagging with noisy labels and minimal supervision," 2020.

[153] "Challenge dcase 2019 task 2: Audio tagging with noisy labels and minimal supervision," http://dcase.community/challenge2019/task-audio-tagging/ [Último acceso 11/agosto/2021].

[154] "Yahoo flickr creative commons 100m," http://projects.dfki.uni-kl.de/yfcc100m/ [Último acceso 11/agosto/2021].

[155] "Pyenv: Instala múltiples versiones de python en tu sistema," https://ubunlog.com/pyenv-instala-multiples-versiones-de-python-en-tu-sistema/ [Último acceso 11/agosto/2021].

[156] A. Mesaros, T. Heittola, and T. Virtanen, "Metrics for polyphonic sound event detection," *Applied Sciences*, vol. 6, no. 6, 2016. [Online]. Available: https://www.mdpi.com/2076-3417/6/6/162

[157] G. Forman and M. Scholz, "Apples-to-apples in cross-validation studies: Pitfalls in classifier performance measurement," *SIGKDD Explor. Newsl.*, vol. 12, no. 1, pp. 49–57, November 2010. [Online]. Available: http://doi.acm.org/10.1145/1882471.1882479

[158] "Challenge dcase 2017 task 3 results," http://dcase.community/challenge2017/task-sound-event-detection-in-real-life-audio-results/ [Último acceso 2/agosto/2021].

[159] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[160] K. Miyazaki, T. Komatsu, T. Hayashi, S. Watanabe, T. Toda, and K. Takeda, "Convolution-augmented transformer for semisupervised sound event detection," in *Proc. Workshop Detection Classification Acoust. Scenes Events (DCASE)*, 2020, pp. 100–104.

Universidad de Alcalá
Escuela Politécnica Superior

ESCUELA POLITECNICA
SUPERIOR

Universidad
de Alcalá