

**UNIVERSIDAD EUROPEA
MIGUEL DE CERVANTES**

ESCUELA POLITÉCNICA SUPERIOR

**TITULACIÓN:
MÁSTER UNIVERSITARIO EN GESTIÓN Y ANÁLISIS
DE GRANDES VOLUMENES DE DATOS: BIG DATA**



TRABAJO FIN DE MÁSTER

**Eliminación de ruido espectral
basado en redes neuronales**

AUTOR

Ignazio F.Finazzi

TUTOR

Patricia Jiménez Fernández

VALLADOLID, junio de 2020

Índice de contenidos

| | |
|---|-----------|
| 1. Objetivos del trabajo | 7 |
| 2. Análisis de la situación | 9 |
| 2.1. Del mundo analógico al digital y viceversa | 9 |
| 2.1.1. Muestreo | 10 |
| 2.1.2. Cuantización | 12 |
| 2.2. Proceso tradicional vs redes neuronales | 14 |
| 2.3. Redes neuronales para series temporales | 15 |
| 2.3.1. LSTM | 16 |
| 2.4. Estado del arte en cancelación de ruido con redes neuronales | 20 |
| 2.4.1. Trabajos de investigación y de dominio público | 20 |
| 2.4.2. Algoritmos y programas privativos | 22 |
| 3. Obtención, procesado y almacenamiento de los datos | 24 |
| 3.1. Audio | 24 |
| 3.1.1. Extracción audio de YouTube | 24 |
| 3.1.2. Generación de audio sintético | 25 |
| 3.1.3. Descarga masiva de audiolibros | 26 |
| 3.2. Ruido | 31 |
| 4. Análisis exploratorio de datos | 33 |
| 4.1. Análisis del volumen e integridad de los datos a través de sus metadatos | 33 |
| 4.2. Análisis de los datos de audio | 35 |
| 4.2.1. Dominio de la frecuencia | 35 |
| 4.2.2. Dominio del tiempo | 41 |
| 5. Diseño e implementación de los modelos o técnicas necesarias | 43 |
| 5.1. Preprocesado de datos de audio para el modelo | 44 |
| 5.1.1. Generación de datos de entrenamiento y validación | 46 |

| | | |
|--------------------|---|-----------|
| 5.2. | Modelo de capas LSTM | 47 |
| 5.2.1. | Tipos de arquitectura | 47 |
| 5.2.2. | Arquitectura del modelo | 48 |
| 6. | Análisis de los resultados obtenidos | 53 |
| 6.1. | Comparación con una red mono-capa | 55 |
| 7. | Conclusiones y planes de mejora | 58 |
| 7.1. | Cierre y agradecimientos | 60 |
| A. | Código de la actividad | 62 |
| A.0.1. | Códigos de la gestión de la base de datos | 62 |
| A.0.2. | Códigos del scraper de LibriVox | 68 |
| A.0.3. | Códigos del gestor de audios | 70 |
| A.0.4. | Código del gestor de ruido | 72 |
| A.0.5. | Código de la preparación de los datos para el entrenamiento | 73 |
| A.0.6. | Código del generador de secuencias | 76 |
| A.0.7. | Código del modelo | 77 |
| A.0.8. | Jupyter Notebook del algoritmo de mayor precisión | 79 |
| Referencias | | 85 |

Índice de figuras

| | | |
|-------|---|----|
| 2.1. | Esquema de la conversión analógico a digital | 10 |
| 2.2. | Esquema del muestreo de una señal de 1Hz muestreada a 4 muestras por segundo | 11 |
| 2.3. | Ejemplo de aliasing | 12 |
| 2.4. | Esquema del plegado de una señal con aliasing | 12 |
| 2.5. | Esquema de la cuantización con 2 bits a 10 muestras por segundo, i.e., 4 valores posibles | 13 |
| 2.6. | Esquema de la cuantización con 4 bits a 10 muestras por segundo, i.e., 8 valores posibles | 13 |
| 2.7. | Resolución del Analogical to Digital Converter (ADC) | 14 |
| 2.8. | Comparación de neuronas tradicionales con neuronas recurrentes . . | 15 |
| 2.9. | Celdas Long Short-Term Memorys (LSTMs) | 16 |
| 2.10. | Propagación del estado de una celda LSTM | 17 |
| 2.11. | Puerta de olvido en la celda LSTM | 18 |
| 2.12. | Puerta de entrada en la celda LSTM | 18 |
| 2.13. | Actualización del estado de la celda LSTM | 19 |
| 2.14. | Puerta de olvido en la celda LSTM | 20 |
| 2.15. | Diagramas de flujo de la publicación de RNNoise | 21 |
| 2.16. | Red neuronal de RNNoise | 21 |
| 2.17. | FFTs de la publicación de RNNoise | 22 |
| 3.1. | Esquema de la navegación web | 27 |
| 3.2. | LibriVox página principal | 28 |
| 3.3. | LibriVox analizado con el inspector de Firefox | 29 |
| 3.4. | Diagrama de flujo para el scraper de LibriVox | 31 |
| 4.1. | Espectrograma de la voz humana | 36 |
| 4.2. | Espectrograma del ruido | 37 |
| 4.3. | 10 FFTs continuas con un overlap del 50 % de la pista de audio . . | 38 |
| 4.4. | 10 FFTs continuas con un overlap del 50 % de la pista de ruido . . | 39 |

| | |
|---|----|
| 4.5. Espectrograma de las pistas combinadas | 39 |
| 4.6. Espectrograma de las pistas combinadas | 40 |
| 4.7. Series temporales de las pistas de audio y ruido | 41 |
| 4.8. Serie temporal del audio combinado y comparativa de los histogramas en el dominio del tiempo | 42 |
| 5.1. Esquema de las entradas y salidas de datos del modelo | 44 |
| 5.2. Cadena completa de procesado de señal | 45 |
| 5.3. Arquitecturas de modelo | 47 |
| 5.4. Estructura del archivo HDF5 | 50 |
| 5.5. Estructura de los datos almacenados en el archivo HDF5 | 51 |
| 5.6. Comparación de la señal y su normalización para la primera banda de un audio combinado con ruido | 52 |
| 5.7. Histogramas de todas las bandas de un audio combinado con ruido y su normalización | 52 |
| 6.1. Variación de la tasa de aprendizaje con las épocas | 54 |
| 6.2. Precisión en entrenamiento y validación | 54 |
| 6.3. Pérdidas en entrenamiento y validación | 55 |
| 6.4. Variación de la tasa de aprendizaje con las épocas | 56 |
| 6.5. Precisión en entrenamiento y validación | 57 |

Índice de tablas

| | | |
|------|---|----|
| 4.1. | Columnas de la tabla pista de los audiolibros | 34 |
| 4.2. | Columnas de la tabla con los metadatos de las pistas de ruido | 34 |
| 4.3. | Duración total de todas las pistas | 34 |
| 4.4. | Ejemplo de registros repetidos | 35 |

Capítulo 1

Objetivos del trabajo

El desarrollo del trabajo propuesto consiste en el diseño y prototipado de un sistema de entrenamiento de modelos para la eliminación de ruido en conversaciones habladas.

Con el actual paradigma laboral, donde cada vez más se realiza trabajo no presencial, el número de teleconferencias ha aumentado sustancialmente. Este tipo de comunicaciones no siempre se realiza en el mejor entorno sonoro. Por tanto, muchas veces el audio se ve perturbado con ruido ambiental que los micrófonos no son capaces de filtrar. Existen algunas soluciones comerciales para paliar este problema pero no hay algoritmos con redes neuronales de libre acceso para su uso como librerías en programas de eliminación de ruido. Este trabajo pretende crear un **Framework** para el fácil desarrollo de dichas redes en el cual, el ingeniero de datos únicamente se preocupe del diseño de la red.

Para conseguir este objetivo el siguiente trabajo se ha descompuesto en varias secciones, cada una de ellas ejecutada secuencialmente como el propio trabajo presenta.

En la sección **Análisis de la situación** se encuentra el estudio de la técnica actual y cómo se aborda actualmente el problema. Esto es, una breve introducción al análisis y procesado de audio y el estudio del estado del arte en referencia a la cancelación de ruido.

En la sección **Obtención, procesado y almacenamiento de los datos** se presenta cómo se aborda técnicamente la gestión de datos de audio y cómo se obtuvieron. Para este trabajo, se han probado varias técnicas entre las que se encuentra la autogeneración de los datos pero, finalmente, se descartaron en beneficio del desarrollo de un algoritmo basado en **Web scraping** para la descarga masiva de datos almacenados en fuentes públicas.

En la sección **Análisis exploratorio de datos** se presenta el análisis exhaustivo de los datos obtenidos mediante las técnicas de **Web scraping**.

En la sección **Diseño e implementación de los modelos o técnicas necesarias** se presentan los modelos desarrollados basados en redes neuronales recurrentes para la eliminación de ruido ambiental en audio monocanal.

Finalmente, en las secciones **Análisis de los resultados obtenidos y Conclusiones y planes de mejora** se presentan los resultados obtenidos y las conclusiones extraídas del desarrollo del trabajo.

Capítulo 2

Análisis de la situación

El estudio de la eliminación de ruido en audios o conversaciones es un problema que se empezó a estudiar muchos años atrás. Desde la aparición de la era digital el análisis y procesado de audio ha crecido mucho. Hoy en día cuando un cantante graba en estudio un disco, estos audios se procesan digitalmente para ecualizarlos, filtrarlos o añadir efectos en el mundo digital.

Existen numerosos programas de edición y análisis de audio, desde *Adobe Audition* del toolset de Adobe hasta *Audacity* un programa gratuito y open-source o, incluso, Matlab® o Python para entornos de desarrollo o experimentación.

El procesado de audio en la actualidad consiste en, como cualquier otro análisis de señal digital, muestrear la señal, aplicarle algoritmos matemáticos a las muestras obtenidas y pasar al mundo analógico de nuevo la señal para así reproducirla en unos altavoces.

Este capítulo presenta el análisis del problema y el estado de arte en la solución del mismo. Se encuentra dividido en varias secciones donde se verá una breve introducción a la digitalización, una comparativa del análisis tradicional de Digital Signal Processing (DSP), una introducción al estudio de series temporales con redes recurrentes y el estado del arte en la reducción de ruido con redes neuronales.

2.1 Del mundo analógico al digital y viceversa

Las señales en el mundo **analógico** son **continuas en tiempo y en valor** i.e., que para cada instante de tiempo hay un valor concreto y definido para la señal. Por el contrario una **señal digital** es **discreta en tiempo y valor** i.e., que sólo hay valores para determinados instantes de tiempo y además esos valores se encuentran determinados por el rango dinámico y la precisión del proceso de digitalización.

En primer lugar, lo que hace falta para digitalizar una señal es un sensor que transforme una magnitud física continua en una magnitud eléctrica, i.e., tensión, intensidad o resistencia. Hay infinidad de sensores que se encargan de esto en función de la magnitud física que se quiera medir. Un ejemplo sería medir temperatura con un termistor que es una resistencia variable en función de la temperatura. El sensor establece una relación entre la magnitud física objetivo y la magnitud eléctrica, de este modo, digitalizando la eléctrica se digitaliza la magnitud objetivo.

El proceso de digitalización de una señal analógica se divide en tres fases. Muestreo, cuantización y codificación. En la figura 2.1 se muestran las fases de la conversión de una señal analógica $x_a(t)$ a una señal continua en valores y discreta en tiempo $x(n)$, una señal discreta en valores y tiempo $x_q(n)$ y una señal digital representada por la salida binaria. A continuación se estudiarán brevemente el muestreo y la cuantización dado el alto efecto que tienen sobre el procesamiento digital de señales. La codificación está relacionado con cómo se almacenan los bits y no es objeto de este trabajo, por ello, no se explica aquí.

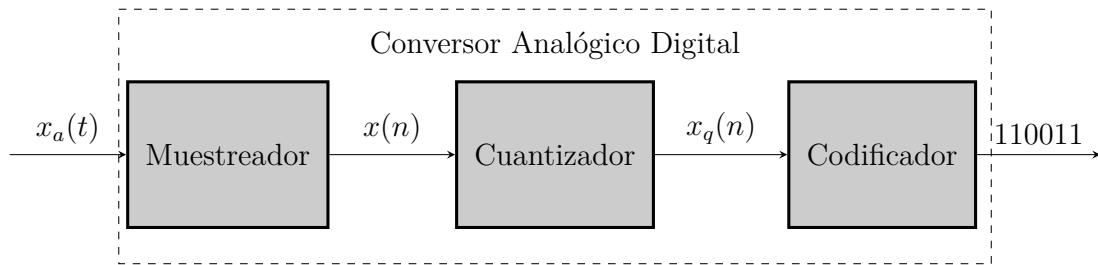


Figura 2.1: Esquema de la conversión analógico a digital

2.1.1 Muestreo

Este proceso consiste en obtener una señal discreta en tiempo a partir de una señal continua en tiempo. Se rige por el parámetro llamado **Tasa de muestreo [sampling rate]**. Este parámetro mide el número de muestras que se van a tomar por unidad de tiempo. Es decir, define la discretización en tiempo. A mayor tasa de muestreo la señal digital podrá reproducir los cambios más rápidos de la señal. En otras palabras, podrá captar frecuencias más altas en la señal analógica. Este punto se verá con mayor detalle en **Teorema de muestreo de Nyquist-Shannon** dado que es fundamental en el análisis de audio.

La figura 2.2 muestra el ejemplo de muestreo de un seno de 1 hercio muestreado a $4\frac{\text{muestras}}{\text{segundo}}$, esto implica $4\frac{\text{muestras}}{\text{periodo}}$.

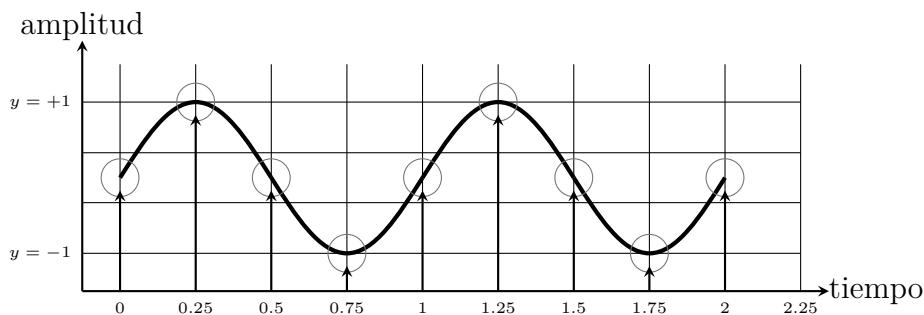


Figura 2.2: Esquema del muestreo de una señal de 1Hz muestreada a 4 muestras por segundo

Teorema de muestreo de Nyquist-Shannon

Si la frecuencia más alta contenida en una señal analógica $x_a(t)$ es $F_{max} = B$ y la señal se muestrea a una tasa $F_s > 2F_{max} \equiv 2B$, entonces $x_a(t)$ se puede recuperar totalmente a partir de sus muestras mediante la siguiente función de interpolación

$$g(t) = \frac{\sin 2\pi Bt}{2\pi Bt} \quad (2.1)$$

Así, $x_a(t)$ se puede expresar como:

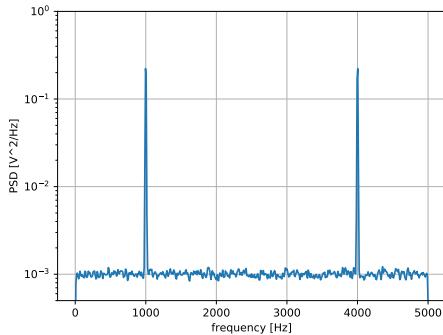
$$x_a(t) = \sum_{n=-\infty}^{\infty} x_a\left(\frac{n}{F_s}\right) g\left(t - \frac{n}{F_s}\right)$$

donde $x_a\left(\frac{n}{F_s}\right) = x_a(nT) \equiv x(n)$ son las muestras de $x_a(t)$

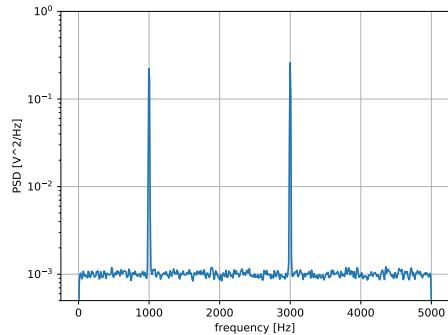
Esto es, que para poder recuperar una señal sin pérdida de información, ésta debe ser muestreada, al menos, con el doble de la frecuencia máxima que la señal contenga. Si no se cumpliera esto, aparecería el fenómeno del *aliasing* que consiste en la superposición de componentes frecuenciales debido a un submuestreo.

En la figura 2.3 se puede ver el efecto de la superposición debida al aliasing. En la primera imagen se pueden ver dos tonos a 1kHz y 4kHz respectivamente, muestreados a 10kHz. En cambio, en la segunda se ven dos tonos a 1kHz y 7kHz muestreados a 10kHz. El tono de mayor frecuencia está submuestreado por tanto los 7kHz se pliegan respecto de un eje en 5kHz apareciendo en 3kHz. De haber habido información en 3kHz ésta se hubiera superpuesto destruyendo lo que en dicha frecuencia hubiera. En la figura 2.4 se muestra un esquema de lo que acaba de ocurrir, donde $\frac{f_s}{2}$ representa la mitad de la frecuencia de muestreo.

Si la información que se encontrara más allá de la frecuencia de muestreo no fuera



(a) Densidad espectral de potencia para la suma dos senos de 1kHz y 4kHz muestreados a 10ksps



(b) Densidad espectral de potencia para la suma dos senos de 1kHz y 7kHz submuestreados a 10ksps

Figura 2.3: Ejemplo de aliasing

de interés pero no se quisiera perjudicar el ancho de banda real de la señal, se debe filtrar con corte en $\frac{f_s}{2}$ para que al plegarse, el nivel de potencia sea despreciable.

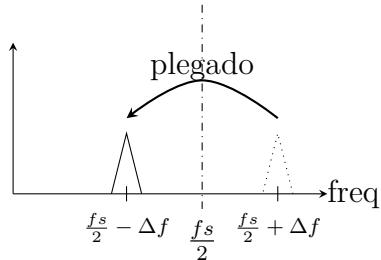


Figura 2.4: Esquema del plegado de una señal con aliasing

2.1.2 Cuantización

La cuantización consiste en obtener una señal discreta en amplitud y tiempo a partir de una señal continua en amplitud y discreta en el tiempo. Para ello, los valores de cada instante de tiempo se aproximan a los valores discretos definidos por los parámetros de la cuantización. Estos parámetros son:

- ▷ **Rango dinámico [dynamic range]**. Este parámetro mide los valores máximos y mínimos hasta los cuales la señal se cuantiza. Es decir, marca los umbrales de la señal. Si los valores de la señal exceden estos límites, la señal se corta apareciendo el llamado efecto *clipping*. A mayor rango dinámico pa-

ra el mismo número de bits mayor rango de valores podrá obtener la señal pero con menor precisión; i.e., los saltos entre estos serán mayores.

- ▷ **Precisión en bits de la digitalización [ADC number of bits].** Este valor representa el número de escalones en los cuales se divide la escalera de cuantización. Es decir, el número de los posibles valores que puede tomar la señal digital. A mayor número de bits para el mismo rango dinámico, la precisión de la cuantización será mejor.

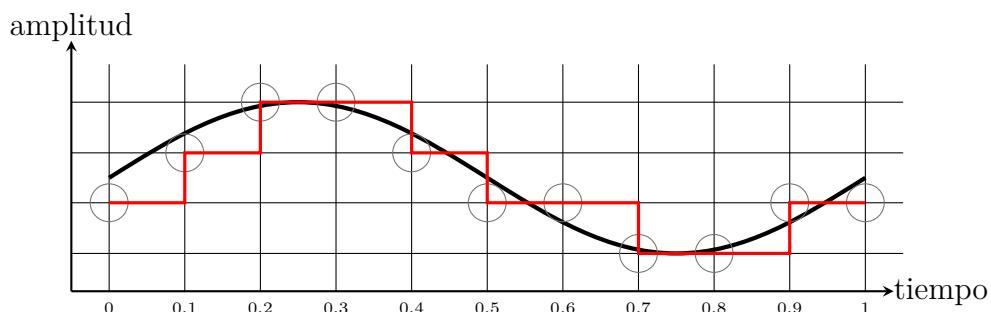


Figura 2.5: Esquema de la cuantización con 2 bits a 10 muestras por segundo, i.e., 4 valores posibles

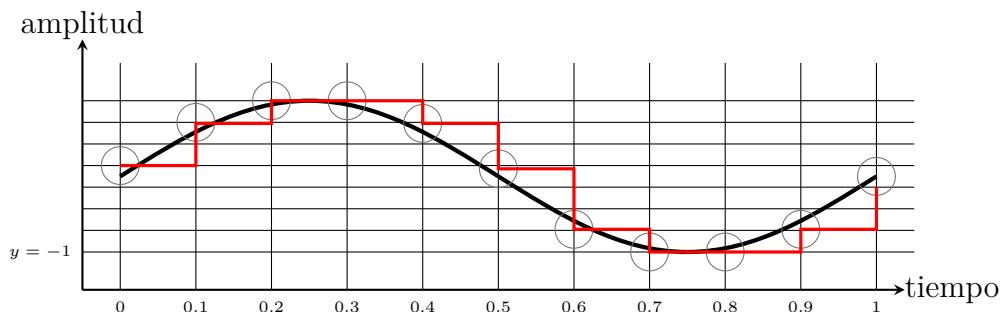


Figura 2.6: Esquema de la cuantización con 4 bits a 10 muestras por segundo, i.e., 8 valores posibles

En las figuras 2.5 y 2.6 se pueden ver ejemplos de una cuantización con 2 y 4 bits respectivamente.

En la figura 2.7 se muestra la escalera de cuantización con la resolución del ADC

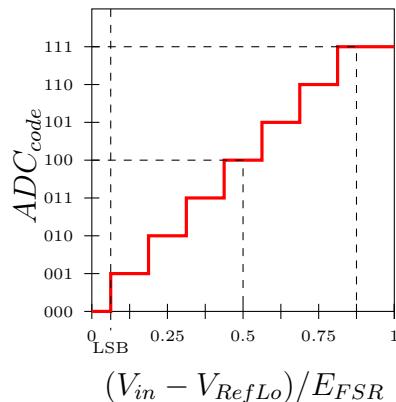


Figura 2.7: Resolución del ADC

calculada como se muestra en 1

$$Q = \frac{E_{FSR}}{2^M} \text{ donde,} \quad (2.2)$$

$$E_{FSR} = V_{RefHi} - V_{RefLow}$$

siendo V_{RefHi} el voltaje máximo de referencia

V_{RefLow} el voltaje mínimo de referencia y

M el número de bits de precisión

Ecuación 1: Ecuación del cálculo de la escalera de cuantización.

2.2 Proceso tradicional vs redes neuronales

El proceso tradicional consiste en procesar la señal digital con técnicas de DSP. Estas técnicas pueden ser de muchos tipos y aplicadas tanto en el dominio del tiempo como en el dominio de la frecuencia. Este tipo de procesado es muy efectivo pero tiene el inconveniente de que si no está lo bastante adaptado para el problema en cuestión, destruye información válida en lugar de limpiar el ruido. Este proceso de afinado, *fine tuning*, es una tarea tediosa y no siempre aplica del mismo modo dado que pequeñas variaciones en el entorno hacen que haya que tunear de nuevo la cadena de procesado de señal.

En contrapartida, están las redes neuronales que, siendo entrenadas con un gran abanico de señales y de ruidos pueden aprender a discernir patrones para limpiar así las señales del ruido. Estos algoritmos son computacionalmente costosos en comparación al DSP tradicional. Es por ello que algunos autores han realizado sistemas híbridos en los cuales el procesado se hace con DSP y el tuneo fino con

redes neuronales. De este modo, se obtienen buenas características de ambos, pero sigue existiendo la limitación del procesado tradicional de DSP que se limita a la cadena de procesado programada, i.e., las etapas de procesado son estáticas en diseño por tanto, su morfología no se adapta según las variaciones del entorno.

2.3 Redes neuronales para series temporales

Una serie temporal es una sucesión de valores que representan una magnitud variable con el tiempo. En el caso del sonido es una onda de presión. En las series temporales, las muestras están relacionadas con su predecesoras y afectarán a sus sucesoras. Las redes neuronales tradicionales no tienen en cuenta los estados anteriores. En el ejemplo de un clasificador de imágenes, la imagen anterior no está relacionada con la siguiente, es decir, la imagen i puede ser un perro y la $i + 1$ un gato pero no existe ninguna relación entre la secuencia de imágenes; hay que aclarar que el caso de un vídeo es diferente porque los fotogramas si guardan relación entre ellos.

Esto demuestra que hace falta algún tipo de neurona o entidad que tenga memoria y sea capaz de relacionar los estados anteriores con los posteriores. Este hecho dio lugar a las **redes neuronales recurrentes**. Es importante remarcar que, el hecho de que tenga memoria implica que, internamente, tiene un estado y para las mismas entradas las salidas pueden ser diferentes debido a que este estado cambie. En la imagen 2.8 se puede ver una esquema de cómo las neuronas recurrentes son afectadas por su estado o salida anterior.

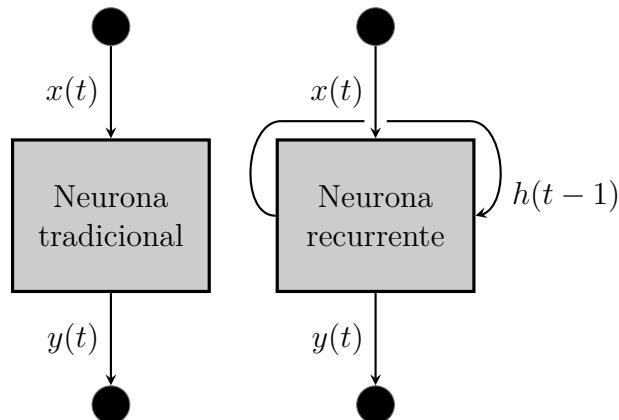


Figura 2.8: Comparación de neuronas tradicionales con neuronas recurrentes

Principalmente existen dos tipos de neuronas recurrentes, las Gated Recurrent Units (GRUs) y las Long Short-Term Memory (LSTM). A continuación se detalla

una explicación de las neuronas LSTM dado que son las usadas por este trabajo.

2.3.1 LSTM

Las celdas LSTM fueron propuestas por Sepp Hochreiter y Jürgen Schmidhuber en 1997 (Hochreiter y Schmidhuber, 1997). Estas redes se crearon para evitar el problema de las dependencias a largo plazo. Esto es, tener en cuenta estados que ocurrieron muchas iteraciones atrás en el tiempo. De esta manera las redes compuestas por celdas LSTM son capaces de 'recordar' información por largos períodos de tiempo(Olah, 2017).

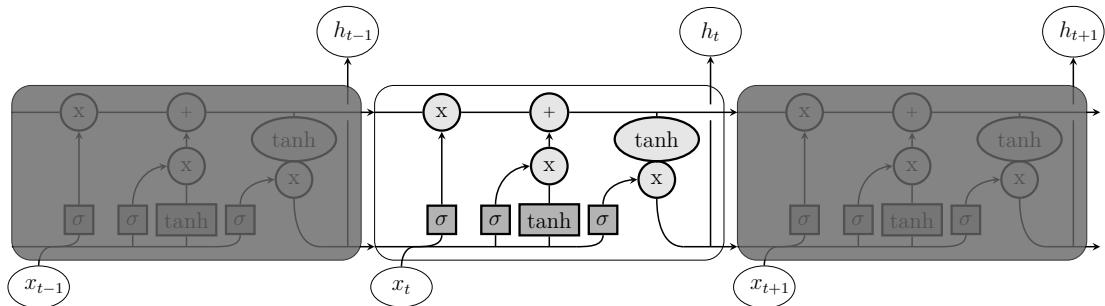


Figura 2.9: Celdas LSTMs

En la figura 2.9 se pueden ver tres módulos LSTMs conectados. El diagrama contiene varias entidades:

- ▷ x_t representan las entradas en los diferentes instantes de tiempo; estas entradas son los vectores de entrada, dado que cada línea contiene un vector completo.
- ▷ h_t representan las salidas.
- ▷ Las operaciones en círculo o elipse son operaciones elemento a elemento de un vector respecto del otro.
- ▷ Los rectángulos representan capas de redes neuronales.
- ▷ Las líneas que se juntan representan concatenaciones de vectores.
- ▷ Las líneas que divergen representan copias del vector por cada línea.

La línea superior de la celda se propaga a lo largo de todas las celdas de la cadena como se muestra en la figura 2.10.

Las celdas tienen la capacidad de eliminar e introducir información mediante la red de neuronas sigmoide seguida de una multiplicación elemento a elemento. La red

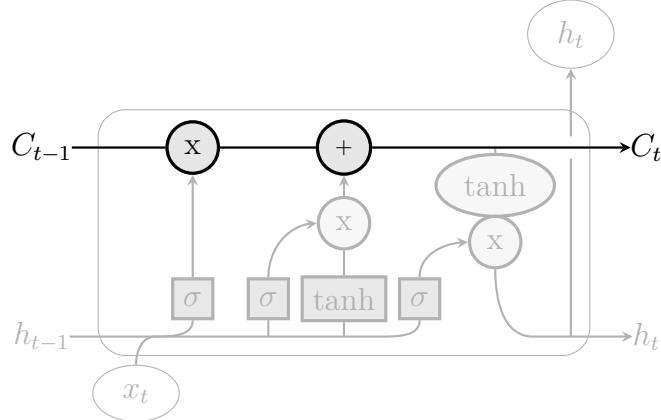


Figura 2.10: Propagación del estado de una celda LSTM

de neuronas sigmoide da un peso entre cero y uno en función de la importancia de cada componente. A este conjunto de elementos se les llama puertas porque tienen la capacidad de dejar pasar información o no. La salida del sigmoide próxima a cero no deja pasar información, por el contrario, próxima a uno si la deja pasar.

Ponderando el estado anterior

A la entrada de la celda lo primero que aparece es la llamada capa de la puerta de olvido, en inglés, *forget gate layer*. Esta puerta tiene la función de discernir cuánto del estado anterior permanece y cuánto se desecha. Como en todas las puertas, la salida es entre 0 y 1 y después se multiplica por el estado anterior C_{t-1} . En la figura 2.11 se pueden ver los bloques implicados en esta operación donde la salida del sigmoide tiene el valor:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \quad (2.3)$$

Ponderando las nuevas entradas

El siguiente paso consiste en discernir cuanta información de la actual a la entrada se propaga al estado actual de la celda. Este paso está dividido en dos etapas.

La primera etapa recibe el nombre de capa de puerta de entrada, en inglés *input gate layer*. Esta puerta tiene la función de decidir cuáles de los valores del vector concatenado de entrada actual y salida anterior van a ser actualizados en el estado

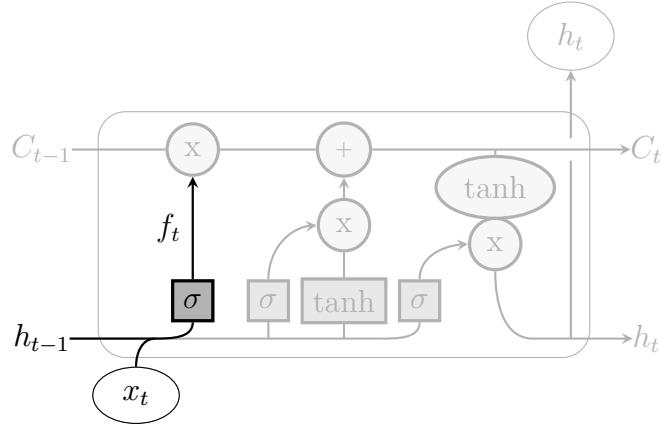


Figura 2.11: Puerta de olvido en la celda LSTM

actual de la celda en la figura 2.12 y en la siguiente ecuación viene representado por i_t .

La segunda etapa consiste en una operación elemento a elemento de tanh en la cual se van a calcular los nuevos valores para cada elemento del vector. En la figura 2.12 y en la siguiente ecuación viene representado por \tilde{C}_t .

$$\begin{aligned} i_t &= \sigma(W_f \cdot [h_{t-1}, x_t] + b_i) \\ \tilde{C}_t &= \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \end{aligned} \quad (2.4)$$

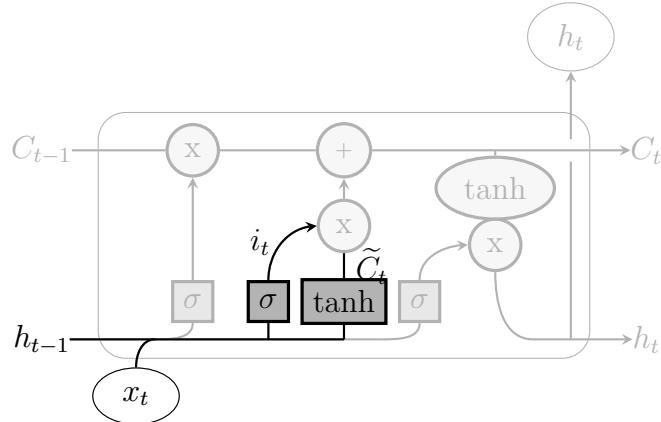


Figura 2.12: Puerta de entrada en la celda LSTM

Actualizando el estado de la celda

A continuación hay que actualizar el estado de la celda. Para ello se multiplica el estado anterior por f_t para olvidar la información que ya no es relevante y se suma la información nueva ponderada por i_t como muestran la figura 2.13 y la siguiente ecuación.

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t \quad (2.5)$$

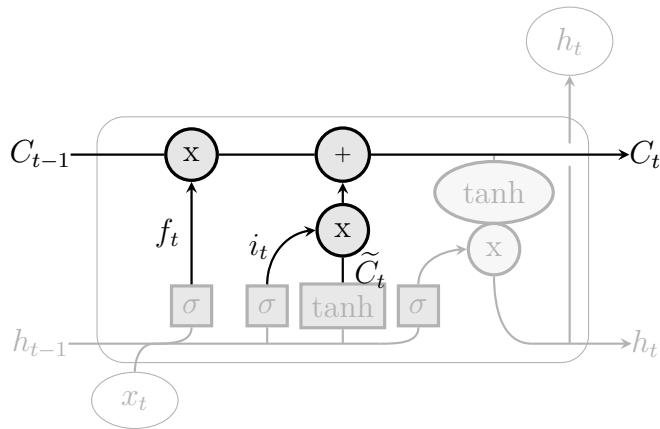


Figura 2.13: Actualización del estado de la celda LSTM

Generando la salida

Finalmente queda generar la salida de la celda. La salida se va a basar en el estado actual de la celda pero no directamente el estado. En primer lugar está la última puerta de la celda que es la encargada de discernir que valores del vector saldrán (al igual que en todas las puertas, es una multiplicación elemento a elemento con un valor entre 0 y 1). Esta puerta dejará pasar los valores del vector de estado de la celda a los que previamente se les ha calculado la tangente hiperbólica cuyos valores de salida son -1 o 1. La siguiente ecuación y la figura 2.14 muestran dichos cálculos.

$$\begin{aligned} o_t &= \sigma (W_o \cdot [h_{t-1}, x_t] + b_o) \\ h_t &= o_t \cdot \tanh (C_t) \end{aligned} \quad (2.6)$$

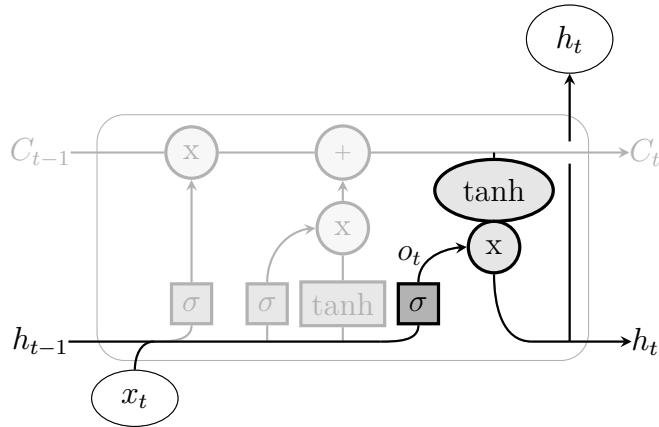


Figura 2.14: Puerta de olvido en la celda LSTM

2.4 Estado del arte en cancelación de ruido con redes neuronales

La cancelación de ruido basado en redes neuronales profundas es algo que se está estudiando en la actualidad y que ha crecido mucho en los últimos años. Esta sección va a discernir entre dos bloques, las investigaciones y desarrollos abiertos y los software privativos de algunas compañías.

2.4.1 Trabajos de investigación y de dominio público

Los trabajos de dominio público y las publicaciones científicas suelen ser la base de los desarrollos e implementaciones comerciales que hacen las empresas posteriormente. En este campo, existen varios algoritmos con resultados muy buenos que han sido la base de estudio para este trabajo.

El primero de ellos y, sobre el que más se basa este trabajo es una red llamada **RNNoise**(Valin, 2018) desarrollada en 2018 por Jean-Marc Valin. Este algoritmo presenta un híbrido entre DSP y redes neuronales.

La figura 2.15a presenta la arquitectura del algoritmo presentado. Se puede ver que tiene un detector de actividad vocal encargado de discernir cuando se elimina el ruido y cuando no. La figura 2.15b presenta el diagrama de bloques de la eliminación de ruido en el dominio de la frecuencia. Este trabajo calcula una Fast Fourier Transform (FFT) con muy poca precisión frecuencial debido al enventanado que utiliza. Es por esta razón que la red neuronal sólo se utiliza como asistencia de la cadena de DSP.

En la figura 2.16(Valin, 2018) se puede ver la red neuronal que implementa el

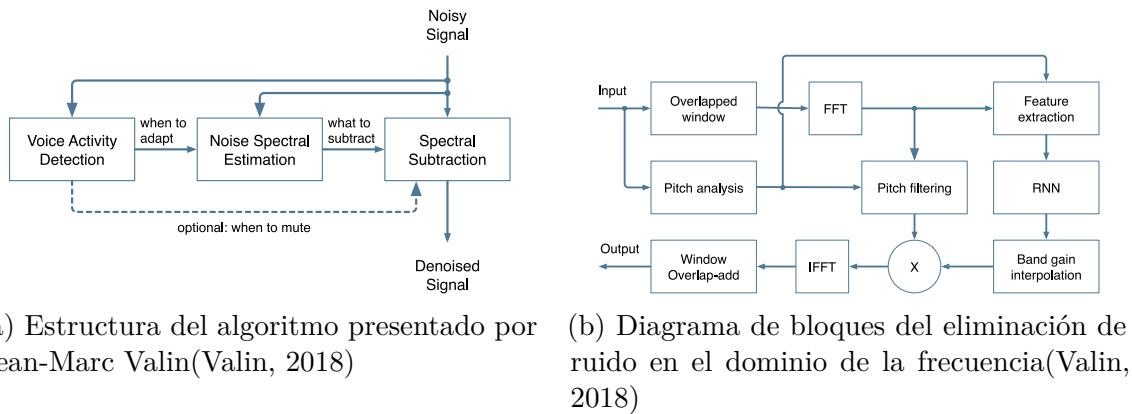


Figura 2.15: Diagramas de flujo de la publicación de RNNNoise

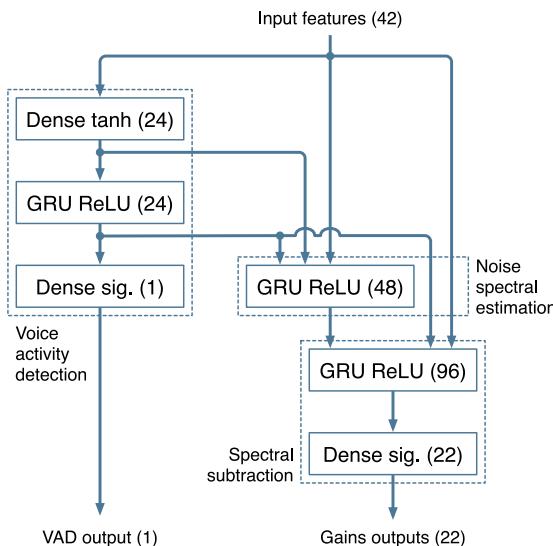
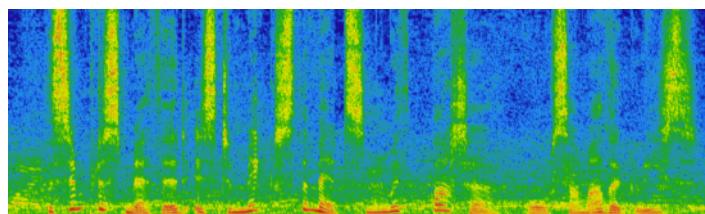


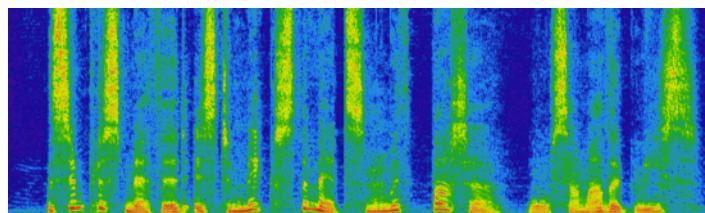
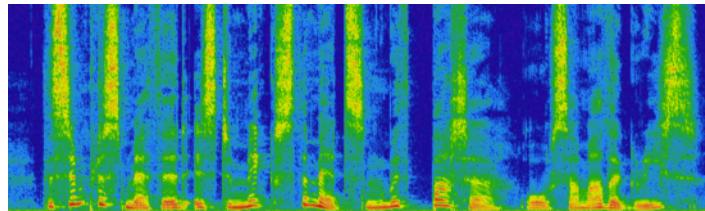
Figura 2.16: Red neuronal de RNNNoise

algoritmo. Es una red formada por celdas GRU, para minimizar el tamaño, dado que es un algoritmo pensado para correr en tiempo real. Es una red formada por tres partes, la detección de actividad vocal, la estimación de ruido espectral y la sustracción del ruido espectral. Como salidas, tiene la detección de voz y las ganancias que se aplican a los filtros de cada una de las 22 bandas en las que se divide el espectro. Se puede destacar que, a partir de un espectro de 22 componentes es imposible recomponer el sonido, es por ello que la salida se usa para tunear unas ganancias, a modo de ecualizador.

En las figuras 2.17a, 2.17b y 2.17c se pueden ver las FFTs con ruido, limpiada por el algoritmo y la original sin ruido, respectivamente. El resultado es bastante bueno



(a) FFT del audio con ruido(Valin, 2018)

(b) FFT con los resultados del audio con la extracción de ruido
del algoritmo RNNNoise(Valin, 2018)

(c) FFT del audio original sin ruido(Valin, 2018)

Figura 2.17: FFTs de la publicación de RNNNoise

dado que el espectrograma sin ruido y el de ruido eliminado son muy parecidos. El código se encuentra disponible en Github(Valin y Richards, 2019) y se distribuye como una Application Programming Interface (API) para su integración en proyectos mayores.

Por otro lado existen proyectos de código abierto en la plataforma Github que, si bien no están tan detallados y no tienen un publicación asociada, si cuentan con el código abierto. Dos ejemplos de ello son (Bukkapatnam, 2020) y (Yuuyoru, 2018). El primero de ellos usa una red basada en LSTM y el segundo se basa en las celdas GRU. Ambos están muy relacionados porque el segundo sale a partir del primero y ambos usan la base de datos abierta MIR-1K.

2.4.2 Algoritmos y programas privativos

Son varias la compañías que han visto un nicho de mercado en este campo. Esta sección va a tratar dos propuestas diferentes que llevan dos enfoques comerciales

distintos.

La primera de ellas es un software de NVIDIA llamado **RTX Voice**(NVIDIA, 2020). Este software es de uso gratuito y tiene desarrollado unos plugins para funcionar con algunas aplicaciones de chat por voz. El enfoque comercial de este programa no es otro que promocionar sus tarjetas RTX dado que para correr el software hace falta una tarjeta gráfica de NVIDIA. Los resultados de este algoritmo son muy impresionantes siendo capaces de eliminar completamente sonidos esporádicos como sonido de teclado, sonido de palmadas y los ruidos habituales de servidores y bullicio.

Krisp(Krisp, 2020) es otra empresa que desarrolla algoritmos de cancelación de ruido. El enfoque comercial que le dan a su producto es venderlo como servicio donde existen dos tipos de licencias, **Free** con límite de $\frac{120\text{min}}{\text{semana}}$ o **Pro** sin límite. Sus resultados son muy buenos y trabajan con la tarjeta de audio y no como un plugin sobre una aplicación. Es un algoritmo ligero que no precisa de una gráfica para procesarse.

Ambas soluciones dan unos resultados prácticamente perfectos y ambas **trabajan en tiempo real**. Como productos comerciales el problema radica en que no dan información en qué red utilizan ni bajo qué condiciones la entranan.

Capítulo 3

Obtención, procesado y almacenamiento de los datos

En este capítulo se abarca cómo se han obtenido los datos, cómo han sido almacenados y preprocesados para su explotación.

Los datos de conversaciones limpias de ruido no son algo fácil de conseguir. Son archivos que ocupan mucho disco y no se encontraron repositorios con la cantidad de información necesaria. Por otro lado está la obtención de ruidos, esta tarea si es más sencilla dado que existe gran diversidad de formas de descargarlos para sumarlos con los audios libres de ruido.

3.1 Audio

En primer lugar cabe destacar que se tuvieron en cuenta varias posibilidades durante dicha fase. A continuación se describen haciendo hincapié en la opción elegida.

3.1.1 *Extracción audio de YouTube*

La primera opción que se tuvo en cuenta fue la extracción del audio de videos alojados en **YouTube** de entrevistas y programas radiofónicos. Este tipo de fuentes conlleva el problema de que no se sabe a priori cuánto libres de ruidos están los audios. Usualmente, se introducen efectos o el ruido del público puede interferir el entrenamiento. Para evitar este problema habría que analizar todos los audios descargados, tarea tediosa y que requiere mucho tiempo.

3.1.2 Generación de audio sintético

La segunda opción que se barajó fue la generación de audio sintético. La generación de datos sintéticos es una técnica muy usada en el entrenamiento de modelos de Machine Learning (ML). Existen dos grupos en la generación de datos sintéticos:

- ▷ **A partir de datos existentes.** Estas técnicas son muy usadas en el análisis de imágenes. Es muy típico ampliar el Dataset de imágenes para clasificadores aplicando transformaciones a la imágenes. Por ejemplo, si se está diseñando un clasificador de tipos de imágenes, una forma de ampliar el Dataset sería generar imágenes espejo de las originales. Esto consiste simplemente ordenar las columnas de la matriz de forma opuesta.
- ▷ **Desde cero.** Estas técnicas requieren usualmente de una red neuronal o algoritmo previamente entrenada que sea capaz de generar dichos datos de manera que sean prácticamente igual a unos reales.

En el caso del audio no se puede partir de un audio para modificarlo y expandir el Dataset. La única forma sería aplicar una ganancia en cuyo caso sería igual a subir o bajar el volumen, pero no es técnicamente generar datos nuevos. Por esta razón se eligió la segunda forma de generar datos sintéticos.

En el caso del audio, los sintetizadores existen hace mucho tiempo. Los dispositivos Global Positioning System (GPS) de los automóviles tienen un sintetizador que reproduce una voz hablada. Estos sintetizadores son algo realmente complicado de hacer y típicamente tienen voces metálicas que no reproducen fielmente una voz humana. De hecho éste es un gran campo dentro de las redes neuronales profundas. Estos tipos de algoritmo se denominan *text to speech*.

Para este trabajo se desplegó el algoritmo *Real Time Voice Cloning*. Este algoritmo es un proyecto de código abierto que consiste en una red neuronal para la clonación de voz(Jia y cols., 2019). El algoritmo se entrena con una voz y después es capaz de reproducir audio con dicha voz a partir de texto.

Finalmente, tras un breve entrenamiento de esta red, se descartó debido a que los resultados generaban una voz metálica. Esto es debido al corto entrenamiento al que fue sometida la red. Cabe destacar que es un algoritmo costoso que, corriendo en gráfica dedicada tarda en generar el audio pero es un algoritmo con gran futuro y fuertemente apoyado por la comunidad open-source.

3.1.3 Descarga masiva de audiolibros

Los audiolibros son una fuente de audio en la cual no deben existir ruidos externos que afecten a la grabación, adicionalmente, son grabaciones reales de personas y se tiene mucha información adicional como el texto que se está leyendo. Por estas razones son fuentes muy válidas para entrenar redes neuronales relacionadas con el análisis de la voz humana.

Como contrapartida está que la mayoría de los audiolibros son de pago y no están disponibles para una descarga masiva. Afortunadamente, existen diferentes sitios web que almacenan audios libros de dominio público, una de las más conocidas es Librivox. Esta web posee audiolibros en varios idiomas perfectamente clasificados y con gran cantidad de información sobre los mismos. Adicionalmente, cuenta con una API que consiste en un servicio web que recibe peticiones GET y devuelve en formato HyperText Markup Language (HTML) o JavaScript Object Notation (JSON) la información solicitada. Este servicio web fue probado pero los resultados no fueron los esperados dado que cuando se aplicaban los filtros, la información devuelta por el servicio no cumplía las restricciones impuestas. Como consecuencia se descartó este método de obtención de la información y las Uniform Resource Locators (URLs) de descarga de los libros. En su lugar, se planteó el desarrollo de un algoritmo de Web scraping.

Los algoritmos de Web scraping son altamente utilizados para la extracción de información de manera masiva y automatizada de los sitios web. Cuando se hace una petición a un sitio web éste devuelve el código de la página web que está almacenado en el servidor que proporciona el servicio de almacenamiento web, en inglés, *hosting* y el navegador web lo interpreta mostrando su contenido. La figura 3.1 presenta un esquema resumido de esta comunicación. Este código puede ser analizado por un algoritmo para extraer información, salvo que el código de la página haya sido ofuscado, dado que es un código altamente estructurado. Este tipo de técnicas están en la frontera de la ética dado que también pueden ser usadas para hackeo web.

Las técnicas de Web scraping están compuestas por varios pasos:

- ▷ Exploración visual del sitio web.
- ▷ Análisis de la URL. En muchas ocasiones, parte de la información que el sitio web devuelve viene filtrada mediante una serie de filtros que se definen en la URL.
- ▷ Análisis del sitio web mediante herramientas de desarrollador. Consiste analizar el sitio web para encontrar dónde está la información que se quiere

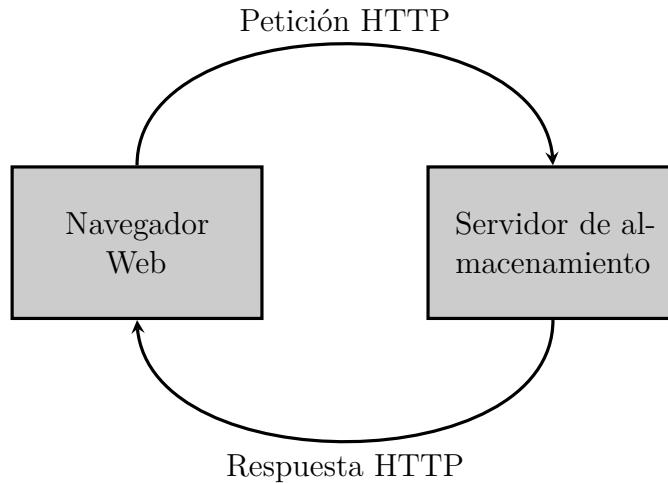


Figura 3.1: Esquema de la navegación web

extraer y cómo viene en el código de la página. Para esto se usa un navegador web y se analiza el código mediante el inspector del navegador. Esta herramienta resalta cada una de las partes del código de la página web que se corresponden con las partes visuales de la misma.

- ▷ Extracción y almacenamiento de la información.

Existen dos tipos principales de Web scraping en función del tipo de web a la que se le quiera extraer la información y del tipo de seguridad que lleve implementada:

- ▷ Sitios web estáticos. Extracción con peticiones librerías de peticiones web y respuestas en código HTML
- ▷ Sitio web dinámicos. Extracción avanzado mediante un driver de navegador (Firefox, Chrome...) y respuestas de código JavaScript.

El primero de los tipos consiste en hacer peticiones a la URL objetivo y extraer la información devuelta en la petición. Este tipo de algoritmos no permiten hacer una navegación interactiva por la página y no permite interactuar con JavaScript, algo cada vez más presente en las webs actuales. Como punto a favor, estos tipos de algoritmo son muy rápidos y fáciles de desarrollar. En contra partida, si se hace un número grande de peticiones al servidor éste puede bloquear la IP de entrada de la petición para protegerse de ataques.

El segundo tipo consiste en la emulación de la navegación humana llamando al navegador. Estas técnicas son mucho más avanzadas y permiten obtener información de cualquier página, llenar formularios y hacer uso de la página como si

de un humano se tratara. Aporta muchas más posibilidades que el otro tipo de Web scraping y es mucho más robusto a bloqueos por parte del servidor, esto se puede mejorar incluso, poniendo esperas aleatorias entre peticiones. Sin embargo, ejecuta un navegador y, por consiguiente, es mucho más lenta la extracción de la información.

Exploración visual

Para el caso concreto del trabajo, la web objetivo se puede ver que contiene diferentes formas de analizar el catálogo. Esto no son más que filtros que se le pueden aplicar a la base de datos donde la página almacena la información. En la figura 3.2 se puede ver la estructura de la página.

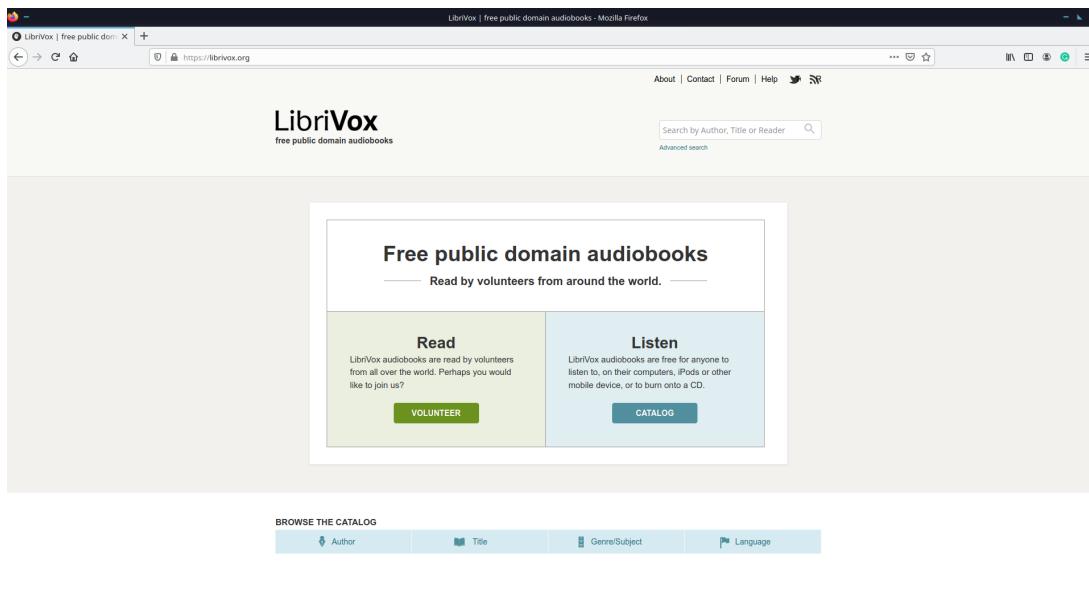


Figura 3.2: LibriVox página principal

Análisis de la URL

A continuación se selecciona el filtro del lenguaje puesto que el trabajo se va a entrar para audiolibros en castellano. El navegador se actualiza y muestra el siguiente contenido en la barra de dirección.

```
https://librivox.org/search?primary_key=5&search_category=language&search_page=1&search_form=get_results
```

De esta información se pueden extraer que existen varios tipos de información separados por el carácter & organizados como **atributo=valor**. A continuación se analizan cada uno de los filtros:

- ▷ **primary_key** es el valor del campo principal de búsqueda, en este caso como se busca por la categoría **language** el valor 5 representa el **español**.
- ▷ **search_category** este campo representa la categoría mediante la cual se están haciendo la búsqueda. Otros posibles valores serían **author**, **title**, entre otros.
- ▷ **search_page** este campo representa la página actual de búsqueda. Por consiguiente para cambiar de página dentro del mismo idioma se debe ir incrementando este número

Análisis del sitio web con herramientas de desarrollador

El siguiente paso consiste en analizar cómo están la información en la página web. En este caso, la lista de resultados la devuelve un JavaScript que tarda unos segundos en cargar. Después de la carga se puede empezar a analizar el contenido como muestra la figura 3.3

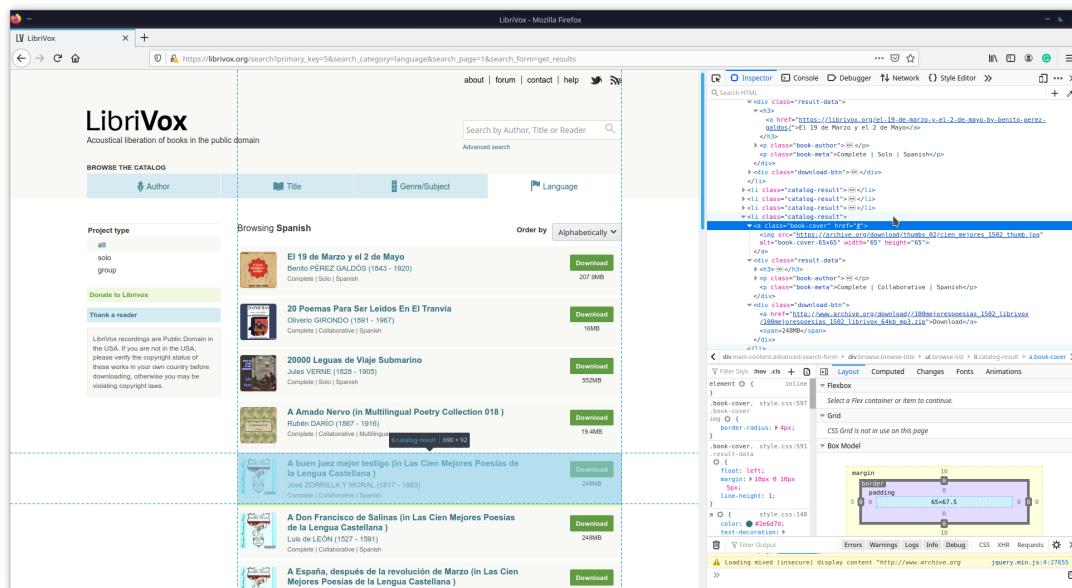


Figura 3.3: LibriVox analizado con el inspector de Firefox

```
<li class="catalog-result">
  <a href="#" class="book-cover">
    
  </a>
  <div class="result-data">
    <h3>
      <a href="https://librivox.org/las-cien-mejores-poemas-de-la-lengua-castellana-by-marcelino-menendez-y-pelayo/">
        A buen juez mejor testigo (in Las Cien Mejores Poesías de la Lengua Castellana )
      </a></h3>
      <p class="book-author"> <a href="https://librivox.org/author/11155">José ZORRILLA Y MORAL (1817 - 1893)</a> </p>
      <p class="book-meta"> Complete | Collaborative | Spanish</p>
    </div>
    <div class="download-btn">
      <a href="http://www.archive.org/download//100mejorespoesias_1502_librivox/100mejorespoesias_1502_librivox_64kb_mp3.zip">
        Download
      </a>
      <span>248MB</span>
    </div>
  </li>
```

Listing 3.1: Ejemplo de la información contenida para uno de los libros

Los datos de los libros que devuelve la búsqueda para dicha página se encuentran dentro de los bloques `<li class="catalog-result">`. El ejemplo del contenido de uno de ellos se encuentra en el listado de código 3.1. Se puede ver que hay mucha información sobre el libro, inclusive, la URL de descarga del mismo.

Extracción y almacenamiento de la información

Para la extracción de la información se desarrolló un algoritmo basado en un webdriver con la librería *Selenium* y en el parseador de HTML *BeautifulSoup*. El algoritmo consiste en ir incrementando el valor de página y buscando toda la información de los libros en dicha página después de que la Query haya cargado. Para esto se busca el valor de la última página; si este valor es nulo, se incrementa un segundo el tiempo aleatorio de espera para el parseo de los datos. Si el valor obtenido es distinto de nulo, significa que la página ha cargado, entonces, se parsea la información de todos los libros en la página y se almacena en una base de datos *SQLite*. En la figura 3.4 se puede ver el diagrama de flujo del código para la parte del scraper y en A.2 se puede ver el código que implementa la clase.

Una vez que la información de los libros está almacenada en la base de datos, se puede pasar a descargar los libros desde sus correspondientes URLs. Para ello se hacen las peticiones GET correspondientes para la descarga de los audios. Una vez descargados, se descomprimen y se analizan para extraerles los metadatos que son almacenados en la base de datos. En A.3 se puede ver el código que implementa esta parte del trabajo.

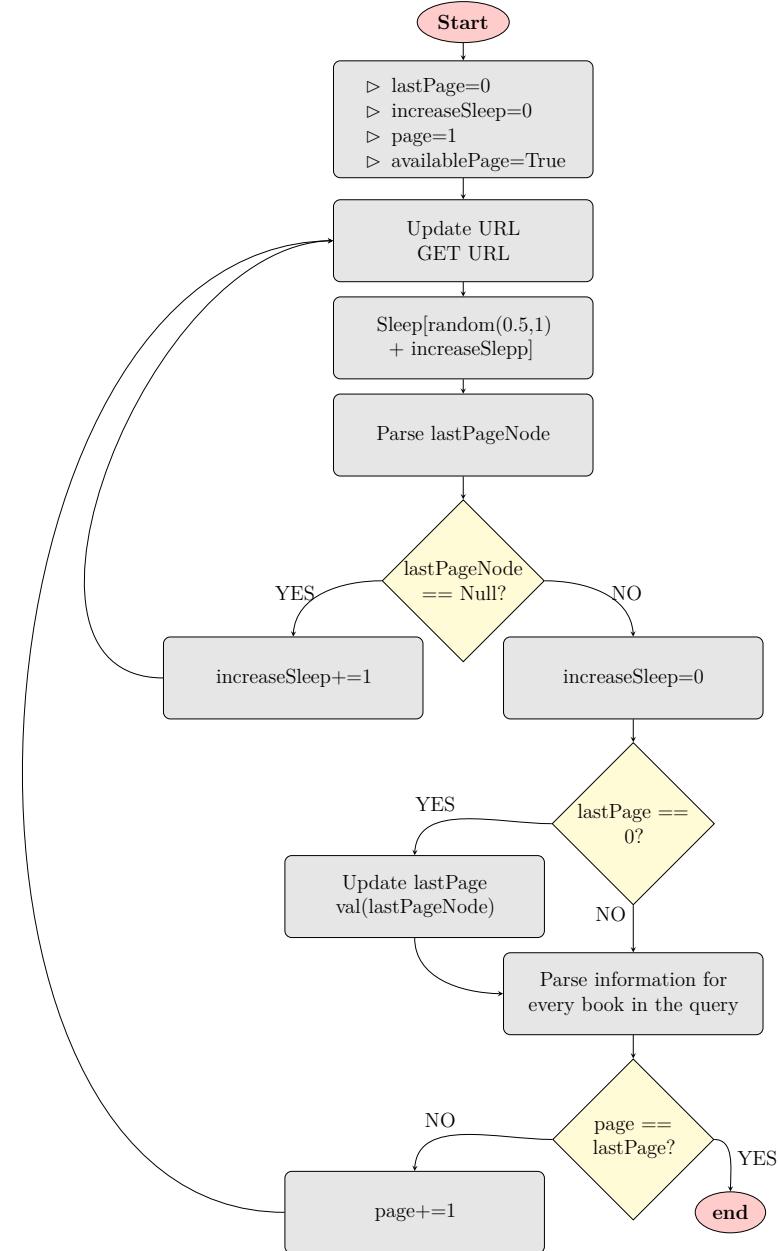


Figura 3.4: Diagrama de flujo para el scraper de LibriVox

3.2 Ruido

Para el ruido se han seleccionado vídeos de YouTube con ruidos urbanos, murmullo, lluvia y ruido característico de un centro de procesado de datos. Estos vídeos se

han descargado mediante la librería *youtube_dl* y se les ha extraído el sonido así como información referente a la tasa de muestreo, duración y el número de canales. Esta información junto con la localización de almacenamiento se ha introducido en la base de datos de SQLite. En A.4 se puede ver el código que implementa esta parte del proyecto.

Capítulo 4

Análisis exploratorio de datos

El análisis exploratorio de los datos se va a dividir en dos grupos. El primero de ellos consiste en el análisis del dataset a partir de los metadatos insertados en la base de datos. El segundo consiste en el estudio y el análisis de los datos de audio descargados y cómo éstos van a ser tratados.

4.1 Análisis del volumen e integridad de los datos a través de sus metadatos

Los metadatos han sido divididos en dos tablas en función del lugar al que pertenecen. La primera tabla hace referencia a las pistas de los audiolibros y la segunda a los ruidos. La tabla 4.1 presenta la columnas que contiene la tabla de las pistas de audio y la 4.2 los metadatos de las pistas de ruido.

A partir de las tablas se pueden sacar datos que den una idea de la dimensión del problema y si los datos son compatibles entre ellos o no. Dos pistas de audio pueden ser no compatibles porque el algoritmo se diseña para una tasa de muestreo estática y, las pistas de audio o ruido, no tienen por qué tener tasas de muestreo que sean múltiplos y divisores comunes de la del modelo. En primer lugar se pueden ver la duración total de las pista de audiolibros y de ruido, para ello simplemente se lanza una query **agregando por el campo duración** con la función **SUM** como muestra la tabla 4.3.

Otra métrica interesante es, como se comentó anteriormente, la compatibilidad de tasas de muestreo. Para ello se **agrega por el campo tasa de muestreo** con la función **DISTINCT**. El resultado se encuentra en la tabla 4.3.

A parte de obtener métricas, se puede analizar la consistencia de los datos. Por ejemplo, se podría analizar si existen algunas pista de audio que se encuentren en

| Nombre | Tipo de dato | Descripción |
|-----------------------|--------------|---|
| id | int | Identificador de la pista de audio |
| book_name_dummy | text | Nombre del libro con caracteres ASCII reducidos |
| book_name | text | Nombre completo del libro |
| book_author | text | Autor del libro |
| book_url | text | URL de descarga del libro |
| book_language | text | Lenguaje del libro |
| book_path | text | Dirección de almacenamiento del libro comprimido |
| book_n_tracks | int | Número de pistas del libro |
| track_name | text | Nombre de la pista de audio |
| track_path | text | Directorio de almacenamiento de la pista de audio |
| track_channels | int | Número de canales de la pista de audio |
| track_sample_rate | int | Tasa de muestreo de la pista de audio |
| track_duration | real | Duración de la pista de audio |
| track_status | text | Estado del archivo (OK → descargado, DELETED → eliminado) |
| track_insert_datetime | int | Fecha y hora de inserción del registro |

Tabla 4.1: Columnas de la tabla pista de los audiolibros

| Nombre | Tipo de dato | Descripción |
|-----------------|--------------|---|
| id | int | Identificador de la pista de ruido |
| name | text | Nombre del ruido que coincide con el nombre del archivo ASCII reducidos |
| url | text | URL de descarga del video que contiene el ruido |
| path | text | Dirección de almacenamiento de la pista de ruido |
| channels | int | Número de canales de la pista de ruido |
| sample_rate | int | Tasa de muestreo de la pista de ruido |
| duration | real | Duración de la pista de ruido |
| status | text | Estado del archivo (OK → descargado, DELETED → eliminado) |
| insert_datetime | int | Fecha y hora de inserción del registro |

Tabla 4.2: Columnas de la tabla con los metadatos de las pistas de ruido

varios libros. Esto se daría si la plataforma de almacenamiento metiera varios libros

| Tipo de pista | Duración [horas] | Tasas de muestreo [<i>muestras</i> / <i>segundo</i>] |
|---------------|---------------------|--|
| Audiolibros | 707.70 ^a | [22050] |
| Ruido | 52.91 | [44100, 48000] |

Tabla 4.3: Duración total de todas las pistas

^aEste dato es previo a la eliminación de pistas repetidas. El dato real es **555.73 horas**

en una misma URL de descarga. En 4.1 se puede ver el código que implementa la query.

```
SELECT track_name, COUNT(*) AS c
FROM audio_books_tracks
GROUP BY track_name
HAVING c > 1
ORDER BY c DESC
```

Listing 4.1: Query para obtener las pistas repetidas en varios libros

El resultado de esta query es que hay **162** registros que se encuentran, como mínimo, dos veces. Por tanto hay pistas de audiolibros que se han contado varias veces falseando las estadísticas. Pasando de las **707.70 horas** de la tabla 4.3 a **555.73 horas**.

Realizando una query con alguno de los resultados se confirma que existen pistas que se encuentran en varios libros y, por tanto, son registros repetidos. En la tabla 4.4 se puede ver una de las pistas que se encuentra repetida. Con este tipo de queries se puede ver que de los **4227** registros, únicos son **2802**.

| id | book_name | track_name |
|-----------|--|---|
| 1351 | Antología de Cuentos Fantásticos | antologiacuentosfantasticos_01_various_64kb.mp3 |
| 3410 | Coppelius (1ra Parte) (in Antología de Cuentos Fantásticos) | antologiacuentosfantasticos_01_various_64kb.mp3 |
| 3935 | De lo que aconteció a un deán de Santiago con don Illan el mágico, que moraba en Toledo (in Antología de Cuentos Fantásticos) | antologiacuentosfantasticos_01_various_64kb.mp3 |

Tabla 4.4: Ejemplo de registros repetidos

4.2 Análisis de los datos de audio

En el procesamiento digital de señal existen dos dominios en los que analizar la señal, el dominio del tiempo y el dominio de la frecuencia. Para entender los tipos de datos que forman el dataset, se van a analizar en ambos dominios.

4.2.1 Dominio de la frecuencia

El procesamiento digital de los datos de audio engloba muchos tipos de familias de audio. No es lo mismo analizar música clásica, que guitarras eléctricas, que música

pop o conversaciones de voz. Cada instrumento se mueve en un registro de frecuencias diferente. Esto es, su frecuencia fundamental y sus armónicos son diferentes, de ahí que produzcan sonidos distintos y se ecualicen de manera diferente.

A la voz humana le ocurre lo mismo, tiene un frecuencia fundamental conocida como *pitch* y una serie de armónicos más o menos atenuados o potenciados según la disposición de nuestros dientes, nuestra lengua y todos y cada uno de los elementos a partir de los cuales generamos los sonidos. Todos los elementos de nuestra boca actúan como filtros o resonadores a la hora de hablar o emitir sonidos. En la figura 4.1 se puede observar el espectrograma de los primeros 10 segundos de un audiolibro. El eje vertical representa las componentes frecuenciales y está limitado a 3000 hercios dado que la voz humana apenas llega a tan altas frecuencias. El eje horizontal representa el tiempo, i.e, la evolución temporal de las componentes frecuenciales y el color, la potencia de dicha componente para ese determinado instante. Cada una de las líneas amarillas-blancas son las principales componentes de la voz y son líneas semi-paralelas porque son el tono fundamental y sus armónicos. Por otro lado, se aprecian grandes bandas verticales negras, esto representan silencios prolongados en el audio, mientras que las bandas verticales más estrechas representan silencios más cortos.

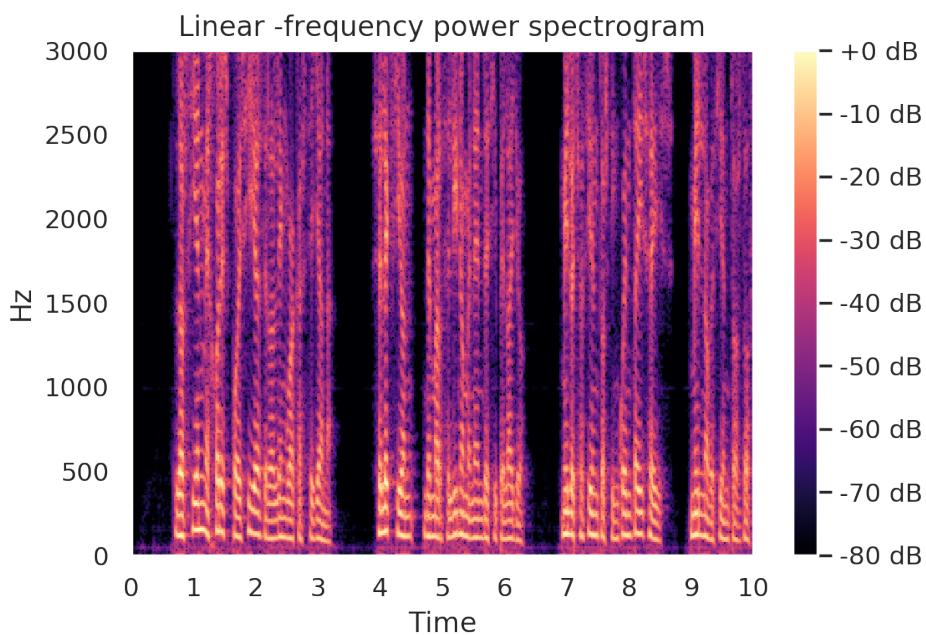


Figura 4.1: Espectrograma de la voz humana

Por el contrario, el ruido blanco cuenta con un ancho banda espectral infinito, i.d.,

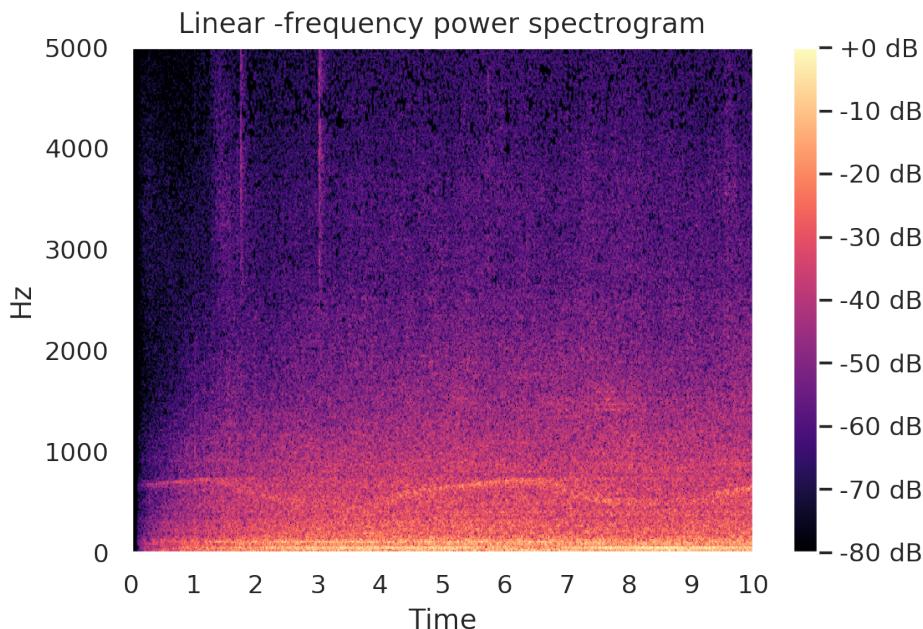


Figura 4.2: Espectrograma del ruido

se manifiesta en todas las frecuencias. El ruido de ciudad no es exactamente ruido blanco porque un pitido de un coche sí es una frecuencia característica. En la figura 4.2 se puede ver el espectrograma del ruido de una ciudad. La forma de interpretar el tipo de gráfico es idéntica a la explicada anteriormente. En este caso se ve un espectro más ancho donde predominan las bajas frecuencias y se aprecia algún tipo de ruido que no es de gran ancho de banda, en este caso es una sirena de algún vehículo de emergencias (viene representada por la línea de mayor intensidad cuya frecuencia varía entre 800 Hz y 500 Hz aproximadamente).

Otro tipo de gráfico interesante para analizar es el espectro. El espectro se genera cortando verticalmente el espectrograma. Esto es, un instante de tiempo del espectrograma, de modo que se genera un gráfico bidimensional donde el eje vertical representa la potencia y el horizontal la frecuencia. Otra forma de verlo, sería la inversa, el espectrograma representa la evolución temporal de todos los espectros. Este tipo de gráficos permiten ver las componentes frecuenciales de la voz con más claridad dado que el ruido es constante para todo el ancho de banda. Las gráficas 4.3 y 4.4 representan varios de estos instantes de tiempo superpuestos para cada pista de audio. En la gráfica 4.3 los picos que se aprecian para la curva **FFT_35** representan las componentes principales de la voz. Estos picos no van a aparecer en todas las FFTs porque la voz es intermitente y además varía de frecuencia a lo

largo del tiempo, por eso no se superponen perfectamente para cada FFT.

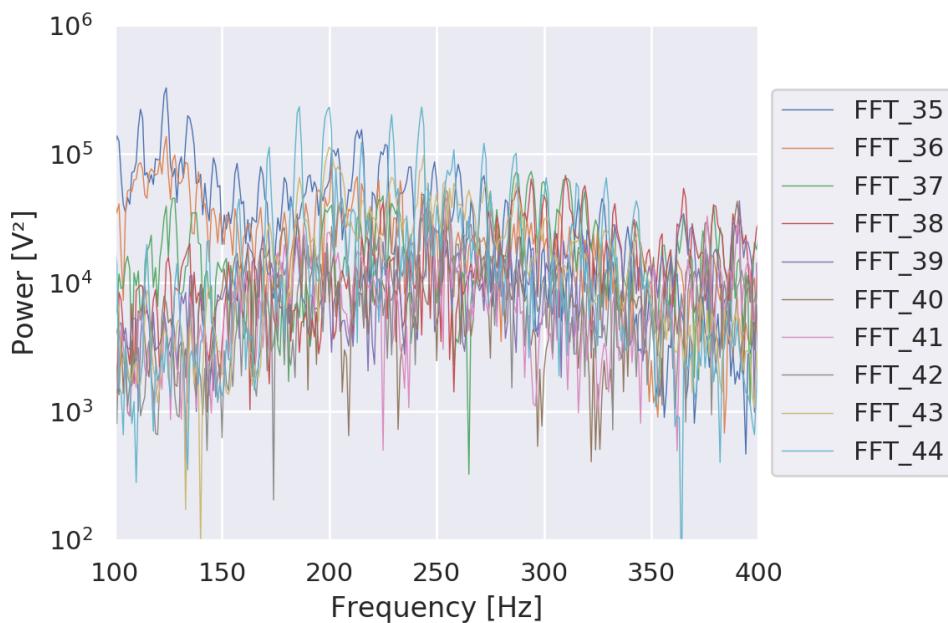


Figura 4.3: 10 FFTs continuas con un overlap del 50 % de la pista de audio

Combinación de pistas de audio

El siguiente paso sería analizar el audio como combinación de las pistas de ruido y audiolibro. En la figura 4.5 se puede ver la combinación de ambos audios.

En este caso la combinación es con los niveles de potencia que tiene cada uno de ellos sin adaptar su amplitud. Idealmente, la combinación de los audios debería ser para un Signal to Noise Ratio (SNR) dado. Es decir, adaptar las potencias de ruido y audio para que sea cual fuere la combinación de audio y ruido la relación entre señal y ruido fuera la misma. De este modo el algoritmo se entrenaría para un nivel de ruido constante dado un nivel de señal constante, un problema mucho más acotado que dejar libre las potencias de entrada de ruido a audio. Esto no se ha realizado porque se consideraba fuera de la primera aproximación que es lo que este trabajo propone.

Se ha comprobado que la combinación de audio y ruido sea audible y, por consiguiente, el algoritmo pueda discernir entre los niveles de potencia. Esto se demuestra en la figura 4.5 donde aún estando el ruido se pueden distinguir perfectamente

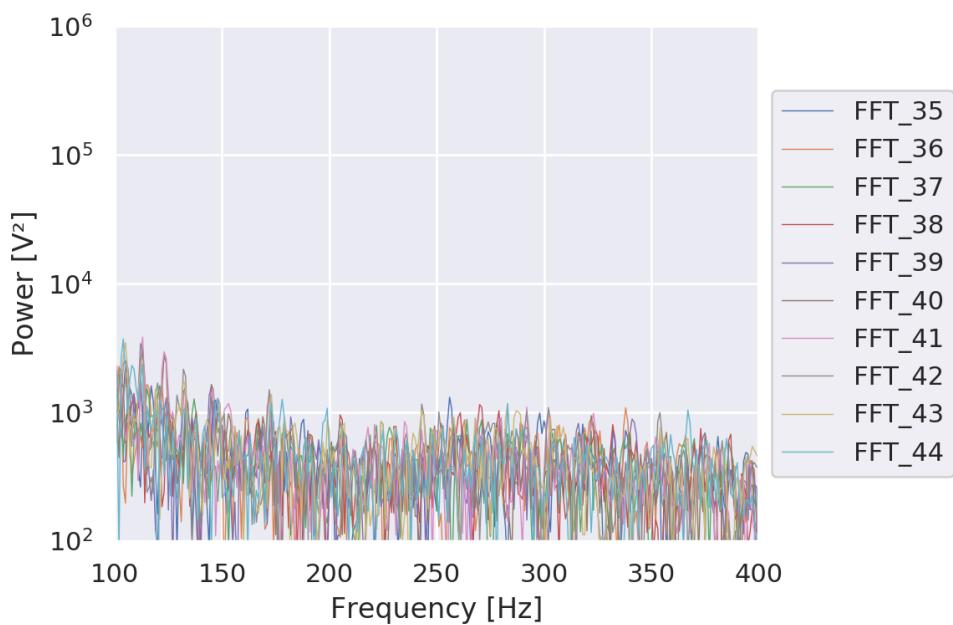


Figura 4.4: 10 FFTs continuas con un overlap del 50 % de la pista de ruido

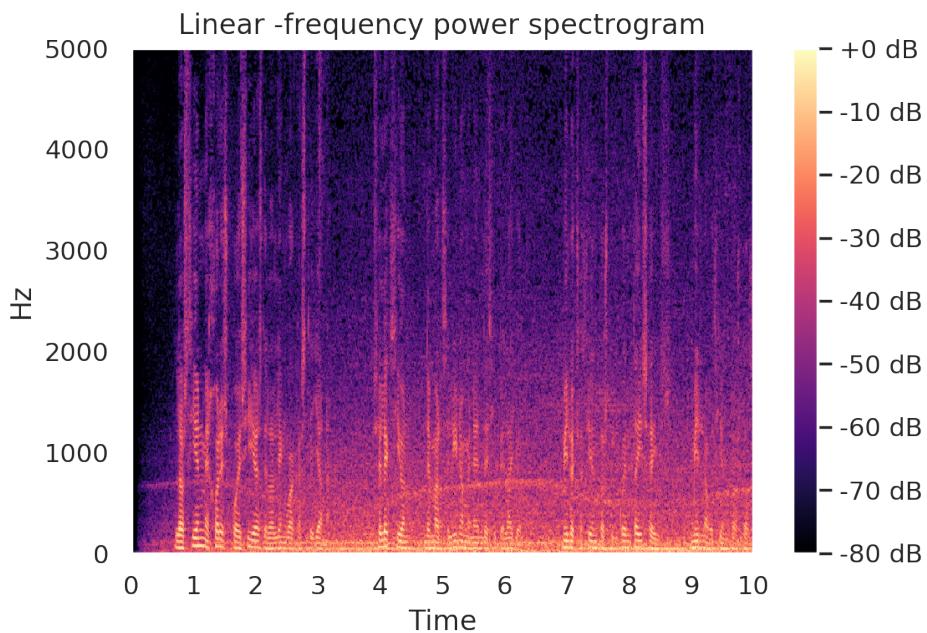


Figura 4.5: Espectrograma de las pistas combinadas

las componentes frecuenciales de la voz. Una gráfica interesante es el histograma comparativo entre la señal sin ruido y la combinación de ambas como se muestra en la figura 4.6. Se puede apreciar que las formas de las distribuciones son similares teniendo la señal libre de ruido más componentes de energía baja, como es de esperar, debido a aquellas bandas en las cuales la persona no habla.

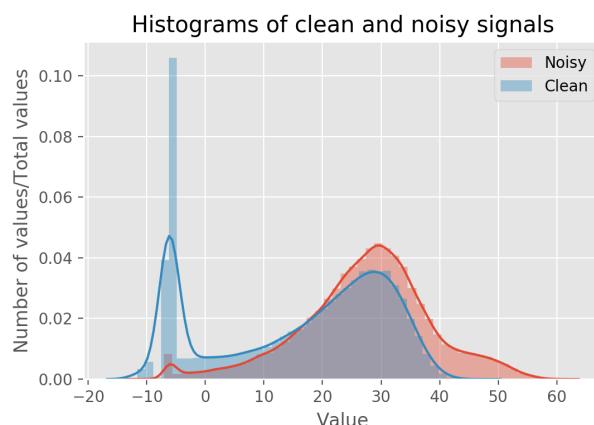


Figura 4.6: Espectrograma de las pistas combinadas

Conclusiones del análisis en el dominio de la frecuencia

De este breve análisis en el dominio de la frecuencia se pueden extraer varias conclusiones:

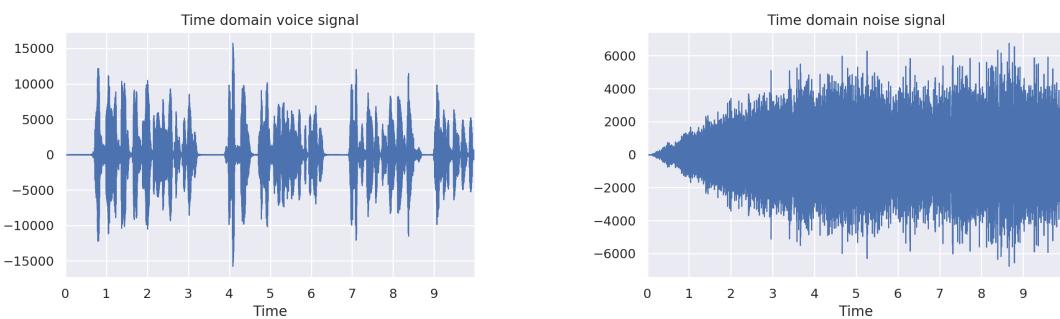
- ▷ **La voz humana no es continua en el tiempo** (la persona se para para respirar mientras habla) lo que crea un espectro seccionado en el tiempo. Por el contrario el **ruido sí es continuo en el tiempo**.*
- ▷ El **espectro de la voz humana está acotado en frecuencia** mientras que el espectro del ruido, en general, no. Como implicación directa tiene que para analizar el audio en el dominio de la frecuencia se puede acotar la tasa de muestreo a la banda donde la voz humana se encuentra, ahorrando mucho procesado.
- ▷ La **voz humana** se caracteriza por tener unas **frecuencias características** que varían en el tiempo y están relacionadas entre sí (tono fundamental y armónicos). Por el contrario el ruido, no presenta relación entre sus componentes spectrales.

*Esta conclusión se verá también en el dominio del tiempo donde la amplitud de la señal cae para el audio humano.

- ▷ Tras la combinación de los audios, si las amplitudes del ruido no son lo suficientemente grandes, las componentes frecuenciales de la voz siguen siendo predominantes en el espectrograma.

4.2.2 Dominio del tiempo

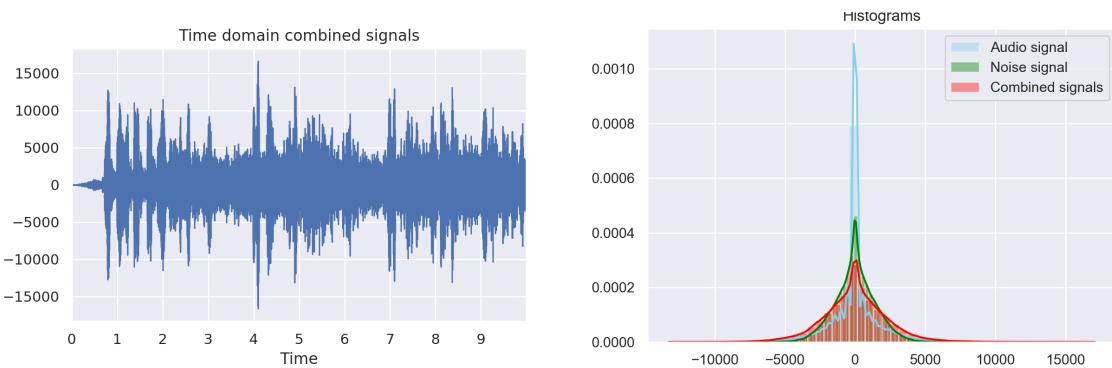
En el dominio del tiempo analizar una señal de tan alta tasa de muestreo es algo complicado. Es mucho más interesante estudiar la envuelta de la señal. En las figuras 4.7a y 4.7b se pueden ver las envueltas de las señales de audio y ruido respectivamente. En primer lugar se aprecia, al igual que en el espectro, que la voz es intermitente en el tiempo, existiendo pausas cortas y largas. Otro punto a destacar son las amplitudes, la del ruido es mucho menor, lo que va a permitir que al combinar ambos audios, el audio no se enmascare detrás del ruido como se ve en la figura 4.8a.



(a) Forma de la señal de audio en el dominio del tiempo (b) Forma de la señal de ruido en el dominio del tiempo

Figura 4.7: Series temporales de las pistas de audio y ruido

Otro aspecto interesante de las señales sería estudiar su histograma. En la figura 4.8b se pueden ver los histogramas de las tres señales. Las conclusiones que se pueden extraer es que las tres señales están centradas en cero, algo lógico debido a la naturaleza de la medida, el audio es una onda de presión. La tres distribuciones son muy parecidas y no es útil usar el histograma para pasar de una distribución a otra.



(a) Forma de la señal combinada de audio y ruido en el dominio del tiempo

(b) Histogramas superpuestos de las tres señales

Figura 4.8: Serie temporal del audio combinado y comparativa de los histogramas en el dominio del tiempo

Capítulo 5

Diseño e implementación de los modelos o técnicas necesarias

En el capítulo 4 se expuso la naturaleza espectral del problema, sería lógico pensar que una forma de reducir el ruido de una conversación sería filtrar para todas aquellas frecuencias que no sean la frecuencia fundamental y sus armónicos característicos del habla de una persona. Esta solución sería perfectamente válida y se ajusta a la necesidad del problema pero tiene un gran inconveniente, el tuneo fino y adaptable de los filtros.

Para poder filtrar las frecuencias fundamentales, éstas deben ser encontradas y esa no es tarea fácil. Además dichas frecuencias varían en el tiempo de modo que los filtros deben adaptarse a cada instante a la nuevas frecuencias, esto complica la tarea aún más. Existe una familia de filtros llamados *comb*, peine, que consisten en una serie de picos equi-espaciados creando mediante retardos de la propia señal perfectamente válidos para la detección y filtrado de la frecuencia fundamental(Tadokoro, Matsumoto, y Yamaguchi, 2002), de nuevo el tuneo fino es una tarea compleja. Por ello, este trabajo propone un sistema completo de redes neuronales en el cual no se aplican técnicas de clásicas de DSP.

Dado que en el capítulo 4 se presentaron los análisis en frecuencia y en tiempo, quedó de manifiesto que en el dominio de la frecuencia las pistas con ruido y sin él se diferencian mucho. Por esta razón el algoritmo que se propone en este trabajo se basa en el análisis de audio en el dominio de la frecuencia, para ello se deben pre-procesar los datos eligiendo unos parámetros que determinarán como serán los datos, a continuación se explican los algoritmos matemáticos de las transformaciones que se van a aplicar a las pistas de audio.

5.1 Preprocesado de datos de audio para el modelo

Los modelos se van a alimentar de FFTs calculadas a partir de las muestras temporales de los audios. Estas FFTs deben ser lo suficientemente precisas para poder reconstruir el audio pero sin usar una gran cantidad de muestras para no producir un retraso en la señal.

Se ha realizado una prueba experimental para calcular a qué tasa de muestreo el audio sigue siendo posible reconstruir. A menor tasa de muestreo, para la misma resolución en frecuencia, la FFT resultante tiene menos puntos, i.e., menor carga computacional. Es importante este ajuste porque reducir la tasa de muestreo a la mitad o un cuarto implica reducir el vector de entrada de la red a la mitad o un cuarto, respectivamente. Adicionalmente sólo son procesados la mitad de los datos de la salida de la transformada de Fourier. Esto es debido a que la respuesta de una transformada es un vector de números complejos y de éstos sólo se procesa el módulo. La fase se deja intacta y se usa para reconstruir el vector complejo junto con el módulo que sale del modelo. La figura 5.1 presenta un esquema de cuáles son las entradas y salidas del modelo.

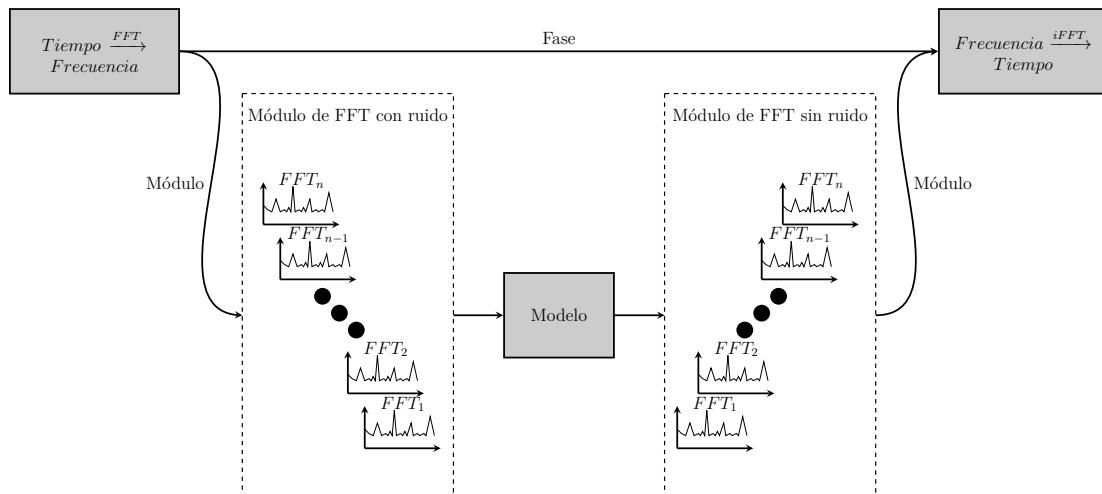


Figura 5.1: Esquema de las entradas y salidas de datos del modelo

Dado que el tamaño de la transformada y la precisión en frecuencia dependen de la tasa de muestreo de la señal, el algoritmo debe funcionar a tasa de muestreo fija. Es decir, sea cual fuere la tasa de muestreo en generación de datos, éstos deben ser re-muestreados, en inglés *resampling*, a la tasa de muestreo objetivo, previo al cálculo de la transformada. Igualmente, el número de puntos de la transformada debe ser fijo. La cadena completa de procesado de señal se muestra en 5.2.

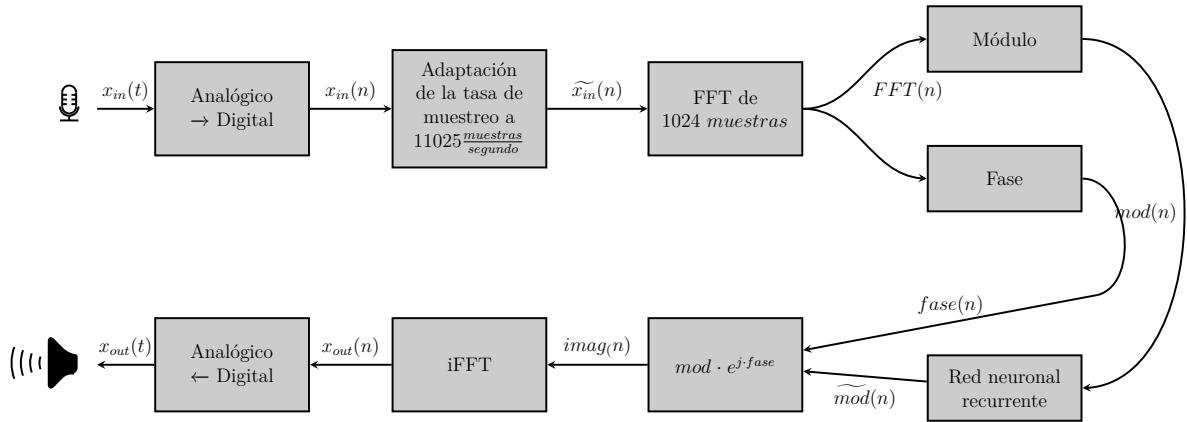
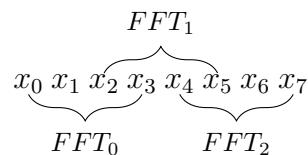


Figura 5.2: Cadena completa de procesado de señal

En esta figura, las diferentes señales significan lo siguiente:

- ▷ $x_{in}(t)$ Muestras analógicas de audio.
- ▷ $x_{in}(n)$ Muestras digitales a la tasa de muestreo de la tarjeta de audio.
- ▷ $\widehat{x}_{in}(n)$ Muestras digitales a la tasa de entrada del modelo $11025 \frac{\text{muestras}}{\text{segundo}}$. Se ha elegido esta banda porque tiene un ancho de banda suficiente como para abarcar el espectro de la voz humana y porque es la cuarta parte de 44100 Hz , una tasa de muestreo muy usada en audio.
- ▷ $FFT(n)$ Vector con las componentes de la FFT. Este bloque acumula 1024 muestras $\widehat{x}_{int}(n)$ y da como salida un vector de 513 **componentes complejas**. Esto da lugar a una precisión frecuencial de $10,74 \text{ Hz}$. Esto quiere decir que la salida de transformada son las energías de las bandas desde 0 Hz hasta 5512 Hz (la tasa de muestreo entre dos, véase **teorema del muestreo**) en bandas de $10,74 \text{ Hz}$ de ancho. Estas bandas permiten reconstruir el audio de manera correcta y no generan un vector demasiado grande. Previo al cálculo de transformada las muestras son enventanadas con una ventana tipo **hann** de 1024 muestras. Adicionalmente, la mitad de las muestras usadas en el cálculo de la transformada es la mitad de las muestras de la ventana anterior. Esto implica que, cada ventana supone un avance en muestras del 50% (overlap del 50%) dado que la transformada es una aplicación de ventana deslizante. A continuación se muestra un ejemplo en el que el tamaño de ventana es de 4 muestras y el overlap del 50%:



- ▷ $mod(n)$ El módulo de cada una de las componentes de la señal $FFT(n)$, i.e., 513*.
- ▷ $\widetilde{mod}(n)$ El módulo de cada una de las componentes de la señal $FFT(n)$ con el ruido eliminado por la red neuronal.
- ▷ $fase(n)$ La fase de cada una de las componentes de la señal $FFT(n)$, i.e., 513 componentes.
- ▷ $imag(n)$ El vector con las componentes de la FFT sin el ruido, para ello se reconstruye el número complejo a partir del módulo y la fase $imag(n)_i = \widetilde{mod}(n)_i \cdot e^{j \cdot fase(n)_i}$.
- ▷ $x_{out}(n)$ Las muestras de salida a $11025 \frac{\text{muestras}}{\text{segundo}}$ tras pasar por la transformada inversa
- ▷ $x_{out}(t)$ Las muestras analógicas listas para ser reproducidas por los altavoces, de nuevo esta salida la genera la tarjeta de sonido del equipo.

5.1.1 Generación de datos de entrenamiento y validación

Para el entrenamiento más rápido del modelo, se pueden pre-procesar todos los datos con la cadena de DSP mostrada en la figura 5.2 previo a la entrada a la red. Para ello, se ha utilizado el código mostrado en A.5.

Este algoritmo consiste en solicitar la combinación de pistas de audio-ruido a la base de datos para los ruidos solicitados. El algoritmo va a ser entrenado únicamente para un tipo de ruido debido a la cantidad de datos y al tiempo de entrenamiento que necesita para cada época. En este caso, se va a usar un ruido de la ciudad de Nueva York y se va a mezclar con todos los audiolibros. A esos audios se les va a calcular la transformada de Fourier a partir de la librería de *señal de Scipy*. Para ello se va a utilizar la Short-Time Fourier Transform (STFT) que genera una matriz de números complejos con todas las FFTs. La salida se va a separar en módulo y fase y el módulo va a ser transformado de amplitud a decibelios. Este proceso se hace para tener valores más representativos. Los módulos de las mezclas de audios serán las entradas y los módulos de los audiolibros solos las salidas esperadas, todo en decibelios.

Finalmente, estos datos junto con sus metadatos de generación serán almacenados en un formato binario de acceso indexado muy ampliamente usado por la comunidad para el procesado de datos a grandes escalas en local, el HDF5.

*Al modelo entran las 250 primeras componentes para reducir la carga computacional dado que reducir el espectro hasta los 3000Hz no hace perder mucha calidad del sonido.

5.2 Modelo de capas LSTM

Este trabajo presenta como solución al problema un modelo de capas LSTM. Previo al diseño del propio modelo, cabe destacar los tipos de arquitecturas que se pueden usar en las redes recurrentes, es decir, cuando se trabaja con secuencias.

5.2.1 Tipos de arquitectura

Cuando se trabaja con redes neuronales recurrentes existen diferentes tipos de arquitecturas en función del número de elementos del eje tiempo que se introduzcan en la red, en el caso de este trabajo sería cuantos vectores FFT tiene el algoritmo en cuenta, dado que un instante de tiempo viene representado por un vector FFT completo.

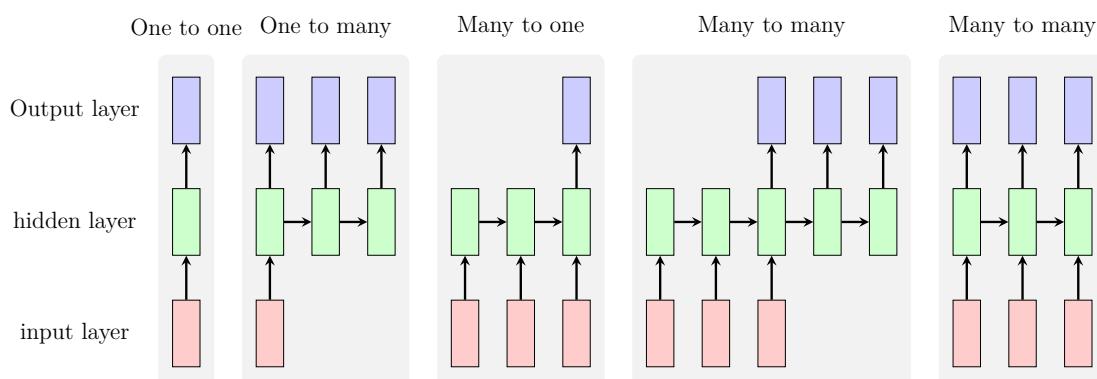


Figura 5.3: Arquitecturas de modelo

En la figura 5.3 se pueden ver las diferentes arquitecturas que se pueden utilizar cuando se realiza un modelo(Karpathy, 2015). En cada tipo de arquitectura, la capa roja representa la capa de entrada, para cada capa de entrada, la entrenada es un vector. La capa verde representa las capas ocultas con sus respectivas propagaciones de estado y las azules, las salidas. Se ha respetado la nomenclatura en inglés original de la bibliografía.

Un ejemplo de cada tipo sería:

- ▷ **One to one** → clasificación de imágenes, una entrada (matriz de la imagen), una salida, la categoría.
- ▷ **One to many** → generación de títulos para imágenes, una entrada (matriz de la imagen), varias salidas, las diferentes palabras.

- ▷ **Many to one** → análisis de sentimiento, entran varias palabras, se clasifica como positiva o negativa.
- ▷ **Many to many** → traducción de texto, varias palabras en un idioma a la entrada y varias a la salida en otro idioma.
- ▷ **Many to many** → entradas y salidas sincronizadas, clasificación de vídeo.

5.2.2 Arquitectura del modelo

Como se explicó anteriormente, las frecuencias fundamentales de la voz y sus armónicos son continuos en el tiempo, de ahí que estudiar cada FFT como un evento independiente sería menos efectivo por tanto, la arquitectura en la cual la entrada es un instante de tiempo no tiene sentido porque se analizaría cada FFT como un evento independiente. Como lo que se espera es una única FFT de salida, las arquitecturas que se probaron durante el trabajo fueron **one to one** y **many to one**^{**}. Aquí cabe destacar que hay dos grandes posibilidades para el caso **many to one**:

- ▷ Entrada de varias FFTs anteriores a la actual y limpias de ruido (es decir darle varias salidas anteriores como entrada) y la FFT con ruido actual
- ▷ Entrada de varias FFTs anteriores a la actual y la actual pero todas con ruido.

Este trabajo se va a implementar la segunda de las opciones porque en la primera, se realimenta la salida, algo que con el estado de la celda ya tiene información de ello y, tiene el problema añadido que, mientras la red no funcione, las FFTs "limpias" de ruido realmente no lo están.

La entrada al algoritmo será, por tanto, una serie de FFTs de 1024 puntos calculadas a una señal de $11025 \frac{\text{muestras}}{\text{segundo}}$ lo que implica que la frecuencia de muestreo de las FFTs es 0.0464, es decir, unas 20 FFTs por segundo, dado que el overlap es del 50 %. El rango de espectro es de 0 a 5512 hercios, una frecuencia muy alta para la voz humana, por esta razón se van a recortar las FFTs a la entrada hasta los 3000 hercios, lo que reduce la matriz de entrada a casi la mitad. De este modo, el tamaño de la FFT a la entrada es de 250 componentes.

El modelo consiste en tres capas LSTM de 512 celdas y una capa de neuronas totalmente conectadas *Dense* con 250 neuronas, dado el tamaño de la FFT. Esto

^{**}Los resultados de esta red en precisión de entrenamiento fueron por debajo del 3 % y se mantenía constante con el paso de las épocas. Se considera que se debe a una mala configuración de la red y los resultados se han omitido.

genera una red de $\approx 6M$ parámetros. En 5.1 se puede ver el resumen de la red y la interconexión de las capas. Cabe destacar que las LSTM requieren de un tensor de tres dimensiones donde la primera es el número de elementos (típicamente tamaño del batch), la segunda, el número de pasos temporales y, la tercera, el número de parámetros, en este caso, las 250 bandas de frecuencia.

| Layer (type) | Output Shape | Param # |
|---------------|----------------|---------|
| lstm (LSTM) | (None, 1, 512) | 1562624 |
| lstm_1 (LSTM) | (None, 1, 512) | 2099200 |
| lstm_2 (LSTM) | (None, 512) | 2099200 |
| dense (Dense) | (None, 250) | 128250 |

Total params: 5,889,274
Trainable params: 5,889,274
Non-trainable params: 0

Listing 5.1: Resumen del modelo

Generador de secuencias

Para poder entrenar el modelo con una gran cantidad de datos que no quepan en la memoria Random Access Memory (RAM) del ordenador de entrenamiento, se requiere de un generador de secuencias. Este algoritmo se va a encargar de preparar los datos en bloques del tamaño del batch para entrenar con ellos el modelo.

En este caso, los datos han sido pre-procesados previamente y almacenados en el formato HDF5, como se explicó en el apartado anterior. Esto se ha hecho para reducir la carga computacional dado que la carga del audio y el ruido, la mezcla de ambos y el cálculo de la STFT es un proceso costoso computacionalmente. De este modo, el *Generador de Secuencias* sólo debe cargar los datos, normalizarlos y juntarlos en grupos del tamaño de pasos temporales y el tamaño del batch. En A.6 se tiene el código que implementa esta parte del proyecto. A continuación se detallan cada una de las partes.

Carga de datos La carga de datos consiste en extraer las FFTs que se encuentran almacenadas en el Hard Disk Drive (HDD) y almacenarlas en RAM. Este paso se divide en dos partes:

- ▷ Parseo de la estructura del archivo para contabilizar el número de FFT total
→ ejecutado 1 vez.
- ▷ Carga de las correspondientes FFTs y actualización de los correspondientes índices para generar un tensor del tamaño del batch → ejecutado tantas

veces como batches hay en el HDF5.

El archivo generado para 1500 de los audios que tengan una tasa de muestreo múltiplo de la del modelo 11025 y para un único ruido tiene un tamaño de 294 GBytes. A modo explicativo se presenta en las figuras 5.4 y 5.5 un ejemplo de la estructura del HDF5 y cómo se encuentran los datos.

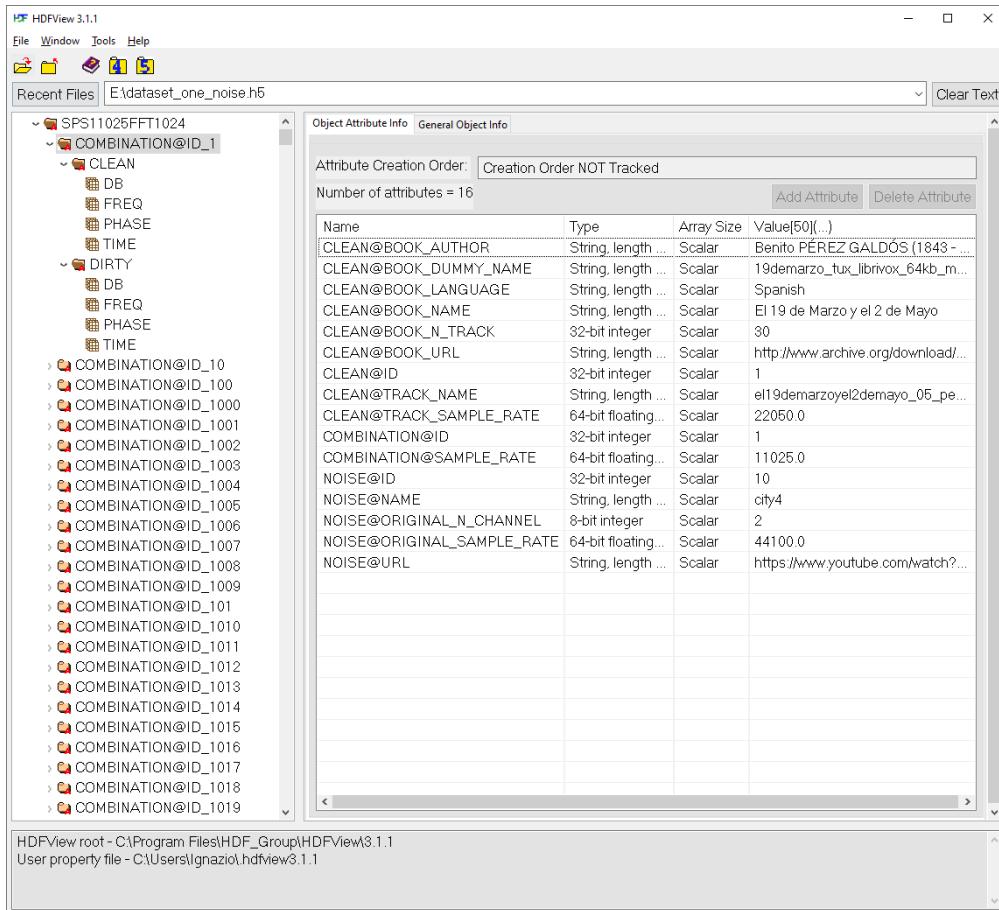


Figura 5.4: Estructura del archivo HDF5

Normalización El proceso de normalización de los datos es una de las partes más importantes y puede ser el causante de que el modelo no funcione correctamente. En este caso, el apartado se refiere a la normalización de las potencias de las bandas frecuenciales calculadas en decibelios. Existe otra normalización que consiste en cargar los datos de los audios a una señal temporal normalizada, i.e., en el rango [-1, 1].

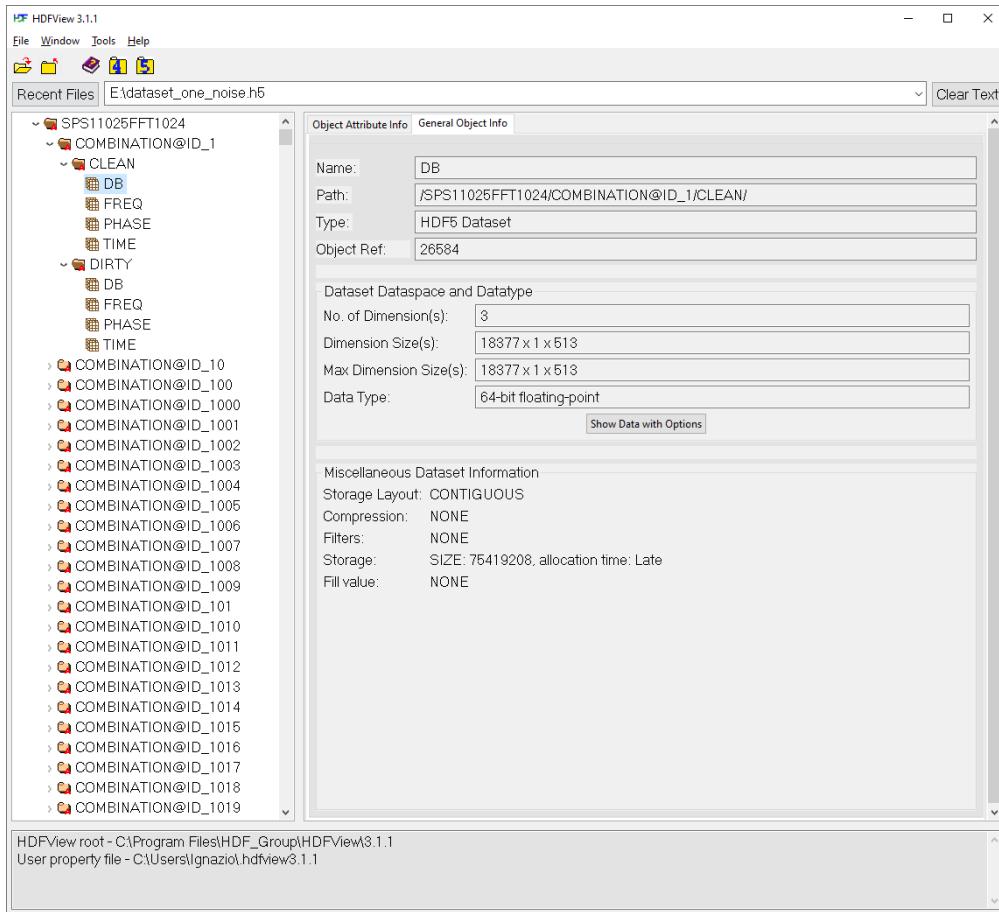


Figura 5.5: Estructura de los datos almacenados en el archivo HDF5

La normalización de las bandas se justifica como que cada banda es una señal y se normaliza aplicando la ecuación 2 que representa la normalización de una componente genérica de potencia i de una banda genérica b .

$$P_{norm\ b,i} = \frac{P_{b,i} - \mu}{\sigma}$$

donde :

$$\mu = \frac{\sum_{i=0}^N P_{b,i}}{N}$$

$$\sigma = \sqrt{\left(\frac{\sum_{i=0}^N |P_{b,i} - \mu|^2}{N} \right)}$$

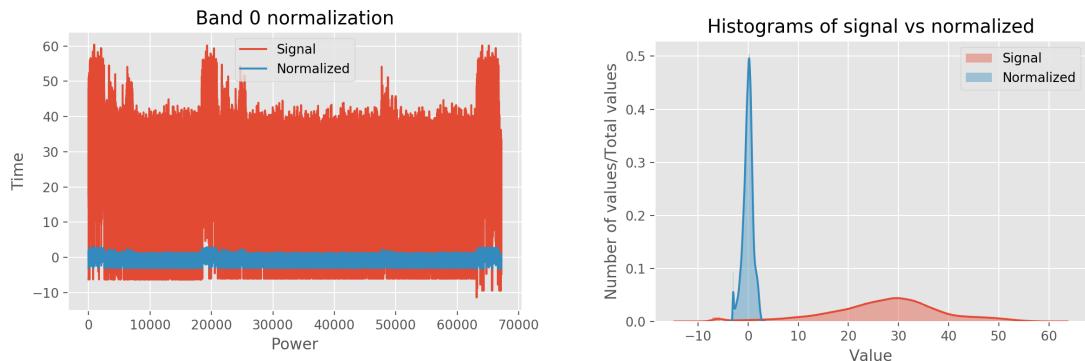


Figura 5.6: Comparación de la señal y su normalización para la primera banda de un audio combinado con ruido

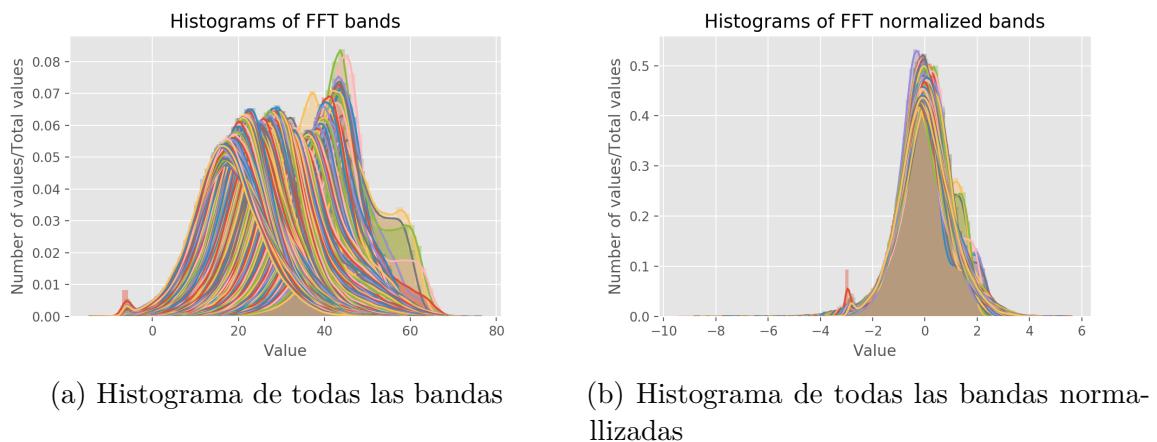


Figura 5.7: Histogramas de todas las bandas de un audio combinado con ruido y su normalización

Ecuación 2: Ecuación de la normalización.

Las figuras 5.6a y 5.6b presentan las comparaciones en tiempo e histograma, respectivamente, entre la señal sin normalizar y la señal normalizada para la banda cero de una misma pista de audio más el ruido. La figuras 5.7a y 5.7b presentan todas las bandas de un audio sin normalizar y normalizadas respectivamente.

Capítulo 6

Análisis de los resultados obtenidos

El algoritmo planteado fue entrenado con el dataset completo sin obtener resultados durante pocas épocas dado el elevado tiempo de entrenamiento por época. La precisión en entrenamiento no aumentaba con el paso de las épocas y se decidió reducir la complejidad del problema. Para ello, se entrenó el algoritmo con dos grabaciones de audio y se validó para una única grabación de audio. El algoritmo se entrenó con las siguientes características:

- ▷ **Tamaño de batch → 32**
- ▷ **Número de FFTs**
 - Entrenamiento → 63132
 - Validación → 4068
- ▷ **Número de épocas → 2000***
- ▷ **Tasa de aprendizaje →** Variable en el tiempo como muestra la gráfica 6.1

Entrenado con esta configuración, el tiempo de entrenamiento por época es de 36 segundos con $\approx 569\mu\text{s}$ por batch en una tarjeta gráfica Advanced Micro Devices (AMD) Radeon Rx570 mediante el Framework ROCm.

Los resultados obtenidos distan mucho de un correcto funcionamiento. La precisión en entrenamiento se estabiliza en torno al 7% y la precisión en validación llega al 2% y decae hasta el 1% claro síntoma de **overfitting** a pesar de tener una pobre precisión en entrenamiento.

En las figuras 6.2 y 6.3 se puede ver las comparativas de pérdidas y precisión para entrenamiento y validación, respectivamente. En la gráficas se aprecia el claro overfitting. Algo a destacar es la lenta memorización de datos porque una red

*El entrenamiento se paró en 1390 al comprobar que los resultados empeoraban



Figura 6.1: Variación de la tasa de aprendizaje con las épocas

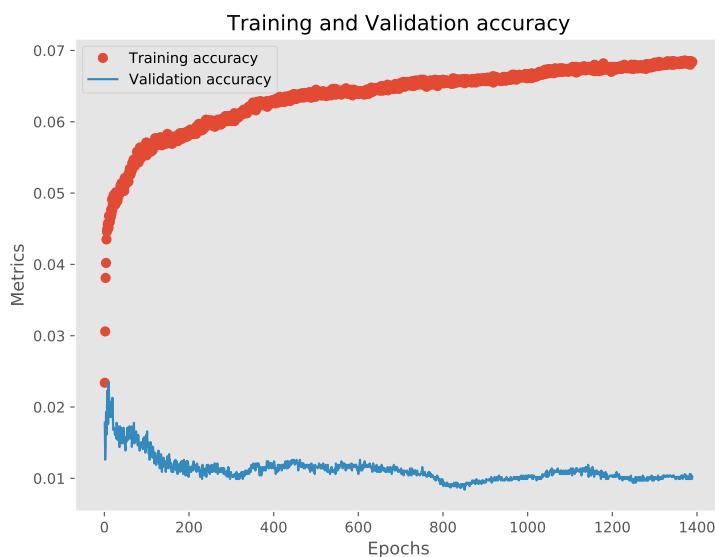


Figura 6.2: Precisión en entrenamiento y validación

tan grande debería ser capaz de memorizar los datos de entrenamiento. Como se aprecia la precisión en entrenamiento no llega a una asíntota de modo que, idealmente debería continuar memorizando los datos con el paso de las épocas.

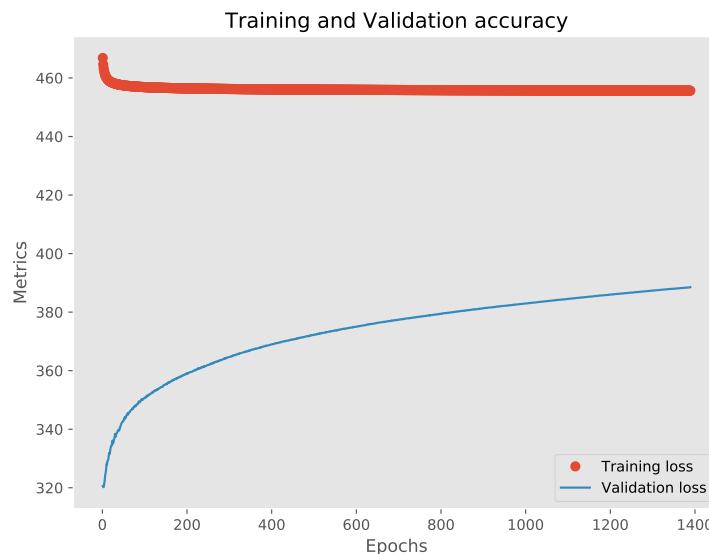


Figura 6.3: Pérdidas en entrenamiento y validación

6.1 Comparación con una red mono-capa

Dada la escasez de precisión de los resultados obtenidos se decidió crear una red distinta con una sola capa LSTM. Los resultados obtenidos fueron muy superiores y con 300 épocas para dos audios en entrenamiento y dos en validación los resultados continuaron mejorando pero a un ritmo ralentizado.

| Layer (type) | Output Shape | Param # |
|------------------------------|--------------|----------|
| <hr/> | | |
| lstm (LSTM) | (None, 2048) | 18833408 |
| dense (Dense) | (None, 250) | 512250 |
| <hr/> | | |
| Total params: 19,345,658 | | |
| Trainable params: 19,345,658 | | |
| Non-trainable params: 0 | | |

Listing 6.1: Resumen del modelo mono-capa

En 6.1 se tiene el modelo presentado en la comparación. Se trata de un modelo con muchos más parámetros pero más simple. Este modelo ha sido entrenado con los siguientes parámetros:

- ▷ **Tamaño de batch → 128**
- ▷ **Número de FFTs**

- Entrenamiento → 55657
- Validación → 11543
- ▷ Número de épocas → 300
- ▷ Tasa de aprendizaje → Variable en el tiempo como muestra la gráfica 6.4

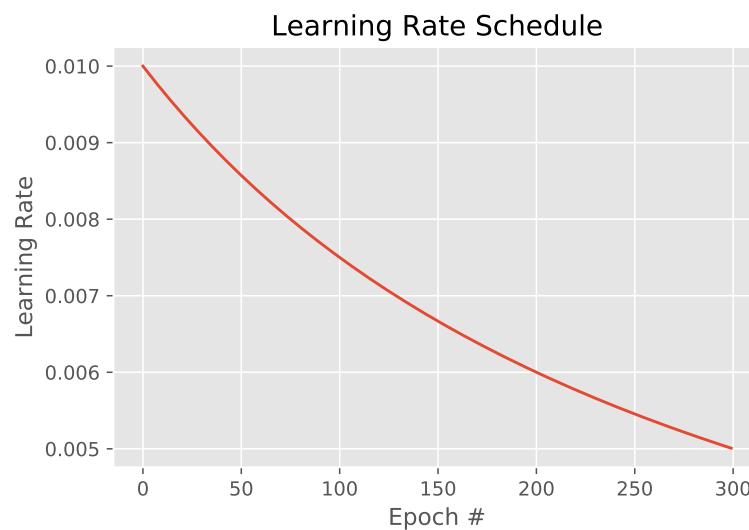


Figura 6.4: Variación de la tasa de aprendizaje con las épocas

Los resultados obtenidos mejoran mucho a la red multi-capa siendo la red idónea para entrenar frente a un mayor abanico de audios y no una muestra reducida. La figura 6.5 muestra los resultados obtenidos y, aunque se ve cierto comportamiento asintótico en los resultados de validación, el algoritmo continúa mejorando en validación, aunque con una tasa reducida.

Después de probar varios modelos, los resultados obtenidos con este último son los mejores dado que en entrenamiento continúa memorizando los datos, lo que da síntomas de que el tamaño de la red comienza a ser el adecuado. La precisión después de 300 épocas es del 18.5 % en entrenamiento frente al 7 % que da la red multi-capa menor (siendo ésta última entrenada en 1300 épocas) y en validación un 6 % frente a un 1 %.

El código relaciona con esta red se adjunta en A.8 como un archivo de Jupyter Notebook dado que, para prototipado de arquitectura y dada la reducida cantidad de datos de entrenamiento (sobre la cual se prototipa), es la forma mejor de desarrollar. Una vez el algoritmo sea definitivo se le debe lanzar una batería mayor de

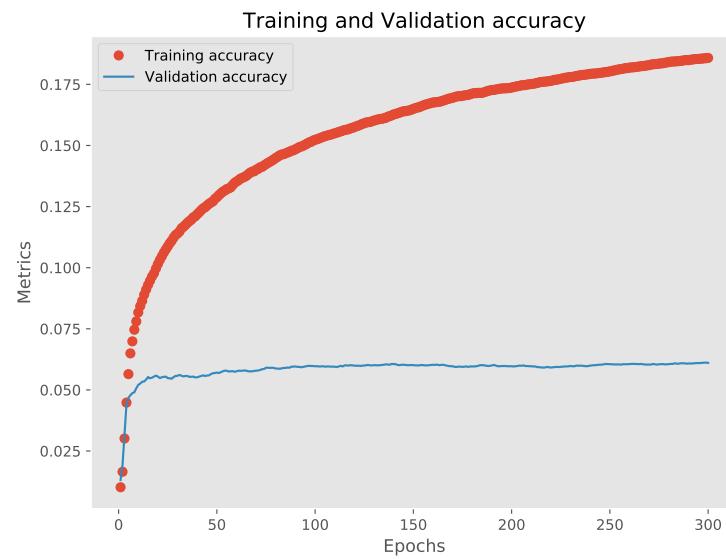


Figura 6.5: Precisión en entrenamiento y validación

entrenamiento, para ello se entrenaría con el generador de secuencias presentado en 5.2.2.

Capítulo 7

Conclusiones y planes de mejora

Del siguiente trabajo se pueden extraer varias conclusiones en diferentes campos obtenidas durante su desarrollo:

- ▷ Hardware AMD Radeon empleado.
- ▷ Métodos de Web scraping
- ▷ Estructuración de los metadatos en base de datos.
- ▷ Análisis en el dominio de la frecuencia y con redes LSTM

Hardware AMD Radeon empleado. En ciencia de datos es muy común creer que los únicos sistemas para acelerar por hardware el entrenamiento de una red neuronal son gráficas NVIDIA mediante su lenguaje CUDA. Esto no es realmente cierto, aunque la inversión de NVIDIA en este campo es mayor a sus competidores* no es la única opción. Existen centros de procesado de datos con aceleración hardware basados en Xilinx Alveo y otras Field Programmable Gate Arrays (FPGAs) de Amazon Web Services (AWS) y su competidor más directo AMD. Las gráficas AMD dan rendimiento similar a las NVIDIA a menor coste. Recientemente, AMD ha sacado ROCm para competir con CUDA en el procesamiento de redes neuronales con TensorFlow. Su resultado es bastante aceptable aunque sigue muy por debajo de NVIDIA. No obstante, se destaca la labor de un framework de código abierto con unos drivers muy jóvenes con una grandísima capacidad de mejora. Sin duda, la competencia favorece al consumidor y este trabajo, ha sido entrenado con dicho framework dando un uso del 100 % a la gráfica AMD Radeon 570x durante más de 48 horas seguidas. Hay que destacar que, mientras NVIDIA CUDA corre bajo varios sistemas operativos, ROCm lo hace sólo bajo Kernel Linux(AMD, 2020).

*Gráficas RTX con núcleos Tensor y la compra de Mellanox por 6900M\$ hace un año

Métodos de Web scraping. La capacidad de estos métodos ha quedado más que demostrada con la cantidad de datos que se es capaz de generar con unas líneas de código. Aquí aparece de nuevo un debate ético entre, qué se debe y qué no se debe hacer. Estas herramientas permiten la descarga masiva de datos de un servidor y, a la vez que son una herramienta para desarrolladores, también lo son para hackers y software malicioso.

Estructuración de los metadatos en base de datos. Este sin lugar a dudas ha sido uno de los puntos clave del trabajo. La evolución de un trabajo con datos es altamente dependiente de los datos de los cuales se parta y de su organización. Una correcta estructuración de los datos conlleva una inversión de tiempo y esfuerzo que se ve recompensada con la reducción de problemas a la hora de explotarlos, lo que implica la mayoría de las veces repetir trabajo. El almacenamiento de los metadatos en la base datos ha permitido, hacer una estimación del tamaño de la base de datos, sacar un relación de audios compatibles con la tasa de muestreo del modelo (sin llegar a obtener un error o, lo que es peor, procesar datos de manera errónea) y tener una monitorización de la procedencia de los mismos.

Análisis en el dominio de la frecuencia y con redes LSTM A pesar de la gran cantidad de variaciones que se han hecho sobre la red, no se ha conseguido llegar a unos buenos resultados. El trabajo en el dominio de la frecuencia es complejo y el tuneo de las celdas LSTM es una tarea complicada. A pesar de ello, se han barajado muchas posibilidad y se ha llegado a un alto grado de comprensión de las redes recurrentes. Pero, resolver un problema tan complejo con la eliminación de ruido en el dominio de la frecuencia no es una tarea fácil. Por tanto, este trabajo presenta las bases sobre las que empezar a desarrollar un abanico de redes para intentar resolver el problema.

Propuestas de mejora Dado que los resultados no son los esperados se proponen una serie de mejoras para intentar mejorar el algoritmo.

- ▷ **Crear un mecanismo de detección activa de voz.** Con este mecanismo automáticamente se elimina todo el espectro, o se le aplica una ganancia negativa muy alta, a las partes en las que no se detecte voz. De esta manera se elimina mucho ruido sin necesidad de llegar a la red neuronal.
- ▷ **Detección de la frecuencia principal y sus armónicos (pitch frequency).** Detectar la frecuencia fundamental y sus armónicos y pre-limpiar el espectro a la entrada del algoritmo.

- ▷ **Trabajar sobre la normalización de los datos.** Este es uno de los puntos con mayor influencia sobre los resultados. Los resultados de la red variaban mucho con datos normalizados o sin normalización. Se debe plantear una normalización que sea posible aplicarla en tiempo real, es decir sólo se conoce el valor de las muestras actuales y las pasadas, no las futuras como en entrenamiento.
- ▷ **Entrenar el último modelo con mayor número de datos.** Dados los resultados obtenidos, el primer punto a tratar sería entrenar el modelo de mayor precisión con más datos ayudándose del generador de secuencias.

7.1 Cierre y agradecimientos

Este trabajo ha hecho uso de librerías de código abierto así como de publicaciones científicas, frameworks de acceso público y el temario aprendido durante el curso. El desarrollo de código se genera en su totalidad en Python 3.7 con las librerías:

- ▷ numpy
- ▷ sqlite3
- ▷ logging
- ▷ os
- ▷ socket
- ▷ getpass
- ▷ time
- ▷ random
- ▷ selenium
- ▷ beautifulsoup
- ▷ requests
- ▷ tqdm
- ▷ audioread
- ▷ copy
- ▷ zipfile
- ▷ youtube_dl

- ▷ argparse
- ▷ scipy
- ▷ librosa
- ▷ pydub
- ▷ h5py
- ▷ tensorflow
- ▷ matplotlib

El presente documento ha sido desarrollado con L^AT_EXy compilado con X_EL^AT_EX. Todos los gráficos aquí presentes se han desarrollado en Tikz para este documento salvo lo obtenidos de la bibliografía (Valin, 2018) que son importados desde la publicación.

Con todo esto, se agradece a la comunidad **open-source** dado que sin sus aportes este trabajo no sería posible.

Apéndice A

Código de la actividad

A.0.1 Códigos de la gestión de la base de datos

```
import logging
import sqlite3
import os
from socket import gethostname
from getpass import getuser

logger = logging.getLogger('DBManager')

class DBManager:
    def __init__(self):
        self.isOpen = False

        ''' Private attributes '''
        self.__name = './model.db'
        self.__exists = False
        self.__conn = None
        self.__cursor = None

    self.createDB()

    def __connect(self):
        self.__conn = sqlite3.connect(self.__name)
        self.__cursor = self.__conn.cursor()
        self.isOpen = True

    def __close(self):
        self.__conn.close()
        self.__cursor = None
        self.__conn = None
        self.isOpen = False

    def createDB(self):
        try:
            if os.path.exists(self.__name):
                logger.info('Data base already exists. Avoiding its creation')
                self.__exists = True
            else:
                logger.info('Connecting data base')
                self.__connect()
                logger.info('Creating table model_checkpoint')
                query = "CREATE TABLE IF NOT EXISTS model_checkpoint ("
                query += ".__add__("
                query += "id integer PRIMARY KEY, ".__add__(_
                query += "datetime text NOT NULL, ".__add__(_
                query += "user text NOT NULL, ".__add__(_
                query += "machine text NOT NULL, ".__add__(_
                query += "model_name text NOT NULL, ".__add__(_
                query += "model_version text NOT NULL, ".__add__(_
                query += "checkpoint_status text NOT NULL, ".__add__(_
                query += "checkpoint_path text NOT NULL))")
                self.__cursor.execute(query)
                logger.info('Creating table model_info')
                query = "CREATE TABLE IF NOT EXISTS model_info (".__add__(_

```

```

    "name text NOT NULL,").__add__(  
    "version text NOT NULL,").__add__(  
    "status text NOT NULL,").__add__(  
    "path text NOT NULL,").__add__(  
    "CONSTRAINT pk_model_info PRIMARY KEY (name, version))")  
self.__cursor.execute(query)  
logger.info('Creating table noise_files')  
query = "CREATE TABLE IF NOT EXISTS noise_files (".__add__(  
    "id integer PRIMARY KEY,").__add__(  
    "name text NOT NULL,").__add__(  
    "url text NOT NULL,").__add__(  
    "path text NOT NULL,").__add__(  
    "channels int NOT NULL,").__add__(  
    "sample_rate int NOT NULL,").__add__(  
    "duration real NOT NULL,").__add__(  
    "status text NOT NULL,").__add__(  
    "insert_datetime int NOT NULL)")  
self.__cursor.execute(query)  
logger.info('Creating table audio_books_tracks')  
query = "CREATE TABLE IF NOT EXISTS audio_books_tracks (".__add__(  
    "id integer PRIMARY KEY,").__add__(  
    "book_dummy_name text NOT NULL,").__add__(  
    "book_name text NOT NULL,").__add__(  
    "book_author text NOT NULL,").__add__(  
    "book_url text NOT NULL,").__add__(  
    "book_language text NOT NULL,").__add__(  
    "book_path text NOT NULL,").__add__(  
    "book_n_tracks int NOT NULL,").__add__(  
    "track_name text NOT NULL,").__add__(  
    "track_path text NOT NULL,").__add__(  
    "track_channels int NOT NULL,").__add__(  
    "track_sample_rate int NOT NULL,").__add__(  
    "track_duration real NOT NULL,").__add__(  
    "track_status text NOT NULL,").__add__(  
    "track_insert_datetime int NOT NULL)")  
self.__cursor.execute(query)  
logger.info('Creating table training_track')  
query = "CREATE TABLE IF NOT EXISTS training_track (".__add__(  
    "id integer PRIMARY KEY,").__add__(  
    "model_name text NOT NULL,").__add__(  
    "model_version text NOT NULL,").__add__(  
    "start_checkpoint_id integer NOT NULL,").__add__(  
    "end_checkpoint_id integer NOT NULL,").__add__(  
    "audio_book_track_id integer NOT NULL,").__add__(  
    "noise_id integer NOT NULL,").__add__(  
    "epoch integer NOT NULL,").__add__(  
    "status text NOT NULL,").__add__(  
    "insert_datetime int NOT NULL")  
self.__cursor.execute(query)  
logger.info('Committing changes')  
self.__conn.commit()  
logger.info('Closing connection')  
self.__close()  
except Exception:  
    self.__close()  
  
def noiseExist(self, name):  
    try:  
        self.__connect()  
        self.__cursor.execute(  
            "SELECT name " +  
            "FROM noise_files " +  
            "WHERE name = '" + name + "'")  
        cursorVal = self.__cursor.fetchall()  
        if not cursorVal:  
            retVal = False  
        else:  
            retVal = True  
        self.__close()  
        return retVal  
    except Exception as error:  
        self.__close()  
        logging.error(str(error), exc_info=True)  
        raise  
  
def noiseCreate(self, name, url, path, channels, sampleRate, duration):  
    try:  
        self.__connect()  
        self.__cursor.execute(  
            "SELECT COALESCE(MAX(id) + 1,1) FROM noise_files"  
        )  
        newId = self.__cursor.fetchone()

```

```

query = "INSERT INTO noise_files ".__add__(  
    "(id, name, url, path, channels, sample_rate, duration, status, insert_datetime) ".__add__(  
        "VALUES (?, ?, ?, ?, ?, ?, ?, ?) ".__add__(  
            self.__cursor.execute(query, [int(newId[0][0]), name, url, path,  
                int(channels), int(sampleRate), duration, "OK"])  
            self.__conn.commit()  
            self.__close()  
        except Exception as error:  
            self.__close()  
            logging.error(str(error), exc_info=True)  
            raise  
  
    def noiseGetByName(self, name):  
        try:  
            self.__connect()  
            self.__cursor.execute(  
                "SELECT * FROM noise_files " +  
                "WHERE name = '" + name + "' " +  
                "ORDER BY insert_datetime DESC " +  
                "LIMIT 1"  
            )  
            retQuery = self.__cursor.fetchall()  
            self.__close()  
            return retQuery  
        except Exception as error:  
            self.__close()  
            logging.error(str(error), exc_info=True)  
            raise  
  
    def noise GetById(self, noiseId):  
        try:  
            self.__connect()  
            self.__cursor.execute(  
                "SELECT * FROM noise_files " +  
                "WHERE id = " + str(noiseId) + " " +  
                "ORDER BY insert_datetime DESC " +  
                "LIMIT 1"  
            )  
            retQuery = self.__cursor.fetchall()  
            self.__close()  
            return retQuery  
        except Exception as error:  
            self.__close()  
            logging.error(str(error), exc_info=True)  
            raise  
  
    def noiseUpdateStatusByName(self, name, status):  
        try:  
            self.__connect()  
            query = "UPDATE noise_files ".__add__(  
                "SET status = '" + status + "' ".__add__(  
                    "WHERE id = (" .__add__(  
                        "SELECT id ".__add__(  
                            "FROM noise_files ".__add__(  
                                "WHERE name = '" + name + "' ".__add__(  
                                    "ORDER BY insert_datetime DESC ".__add__(  
                                        "LIMIT 1").__add__(  
                                            ")"))  
                self.__cursor.execute(query)  
                self.__conn.commit()  
                self.__close()  
            except Exception as error:  
                self.__close()  
                logging.error(str(error), exc_info=True)  
                raise  
  
    def modelExist(self, name):  
        try:  
            self.__connect()  
            self.__cursor.execute(  
                "SELECT name " +  
                "FROM model_info " +  
                "WHERE name = '" + name + "'"  
            )  
            cursorVal = self.__cursor.fetchall()  
            if not cursorVal:  
                retVal = False  
            else:  
                retVal = True  
            self.__close()  
            return retVal  
        except Exception as error:  
            self.__close()

```

```

logging.error(str(error), exc_info=True)
raise

def modelCreate(self, name, version, pyFilePath, checkpointPath):
    try:
        self.__connect()
        query = "INSERT INTO model_info ".__add__(
            "(name, version, status, path)".__add__(

                "VALUES (?, ?, ?, ?)"))

        self.__cursor.execute(query, [name, version, "CREATED", pyFilePath])
        self.__conn.commit()
        self.__close()
        self.modelCreateCheckpoint(name, version, checkpointPath, "CREATED")
    except Exception as error:
        self.__close()
        logging.error(str(error), exc_info=True)
        raise

def modelCreateCheckpoint(self, name, version, checkpointPath, checkpoint_status):
    try:
        self.__connect()
        self.__cursor.execute(
            "SELECT COALESCE(MAX(id) + 1,1) FROM model_checkpoint"
        )
        newId = self.__cursor.fetchone()[0][0]
        query = "INSERT INTO model_checkpoint ".__add__(
            "(id, datetime, user, machine, model_name, model_version, ".__add__(

                "checkpoint_status, checkpoint_path)".__add__(

                    "VALUES (?, datetime('now'), ?, ?, ?, ?, ?, ?)"))

        self.__cursor.execute(query, [int(newId), getuser(),
                                      gethostname(), name, version, checkpoint_status, checkpointPath])
        self.__conn.commit()
        self.__close()
    except Exception as error:
        self.__close()
        logging.error(str(error), exc_info=True)
        raise

def modelGetInfo(self, name, ver):
    try:
        self.__connect()
        query = "SELECT * ".__add__(

            "FROM model_info ".__add__(

                "WHERE name ='" + name + "'").__add__(

                    "AND version = '" + ver + "'"))

        self.__cursor.execute(query)
        cursorVal = self.__cursor.fetchall()
        if not cursorVal:
            retVal = None
        else:
            retVal = {'name' : cursorVal[0][0],
                      'version' : cursorVal[0][1],
                      'status' : cursorVal[0][2],
                      'path' : cursorVal[0][3]}
        query = "SELECT * ".__add__(

            "FROM model_checkpoint ".__add__(

                "WHERE model_name ='" + name + "'").__add__(

                    "AND model_version ='" + ver + "'").__add__(

                        "ORDER BY datetime DESC ").__add__(

                            "LIMIT 1"))

        self.__cursor.execute(query)
        cursorVal = self.__cursor.fetchall()
        retVal['checkpoint_status'] = cursorVal[0][6]
        retVal['checkpoint_path'] = cursorVal[0][7]
        self.__close()
        return retVal
    except Exception as error:
        self.__close()
        logging.error(str(error), exc_info=True)
        raise

def modelTrainNext(self, name, ver):
    try:
        self.__connect()
        query = "SELECT * ".__add__(

            "FROM training_track ".__add__(

                "WHERE model_name ='" + name + "'").__add__(

                    "AND model_version = '" + ver + "'").__add__(

                        "AND status = 'NO TRAINED'").__add__(

                            "ORDER BY id ASC ").__add__(

                                "LIMIT 1"))

        self.__cursor.execute(query)
    
```

```

        cursorVal = self._cursor.fetchall()
        self._close()
        return cursorVal
    except Exception as error:
        self._close()
        logging.error(str(error), exc_info=True)
        raise

    def modelTrainNewEpoch(self, name, ver, target_sample_rate):
        try:
            self._connect()
            self._cursor.execute(
                "SELECT COALESCE(MAX(epoch) + 1,1) FROM training_track "
                "WHERE model_name = '" + name + "' "
                "AND model_version = '" + ver + "' "
            )
            newEpoch = self._cursor.fetchall()[0][0]
            self._cursor.execute(
                "SELECT MAX(id) FROM model_checkpoint "
                "WHERE model_name = '" + name + "' "
                "AND model_version = '" + ver + "' "
            )
            start_checkpoint_id = self._cursor.fetchall()[0][0]
            query = "INSERT INTO training_track" + "_add_"
            query += "SELECT ROW_NUMBER() OVER ( ORDER BY tracks.id, noise.id ) + ".add_()
            query += str(int(newEpoch - 1)) + " AS id,".add_()
            query += "'" + name + "' AS model_name,".add_()
            query += "'" + ver + "' AS model_ver,".add_()
            query += str(start_checkpoint_id) + " AS start_checkpoint_id,".add_()
            query += str(-1) + " AS end_checkpoint_id,".add_()
            query += "tracks.id AS audio_book_track_id,".add_()
            query += "noise.id AS noise_id,".add_()
            query += str(newEpoch) + " AS epoch,".add_()
            query += "'NO TRAINED' AS status,".add_()
            query += "datetime('now') AS insert_datetime,".add_()
            query += "FROM audio_books_tracks AS tracks".".add_()
            query += "JOIN noise_files AS noise".".add_()
            query += "ON 1=1".".add_()
            query += "WHERE (tracks.track_sample_rate%" + "%d) = 0 " % target_sample_rate).add_()
            query += "AND (noise.sample_rate%" + "%d) = 0 " % target_sample_rate)
            self._cursor.execute(query)
            self._conn.commit()
            self._close()
        except Exception as error:
            self._close()
            logging.error(str(error), exc_info=True)
            raise

    def modelTrainGetCombination(self, target_sample_rate, noises, limit):
        try:
            self._connect()
            query = "SELECT ROW_NUMBER() OVER ( ORDER BY tracks.id, noise.id ), ".add_()
            query += "tracks.id AS audio_book_track_id,".add_()
            query += "noise.id AS noise_id,".add_()
            query += "FROM audio_books_tracks AS tracks".".add_()
            query += "JOIN noise_files AS noise".".add_()
            query += "ON 1=1".".add_()
            query += "WHERE (tracks.track_sample_rate%" + "%d) = 0 " % target_sample_rate).add_()
            query += "AND (noise.sample_rate%" + "%d) = 0 " % target_sample_rate).add_()
            query += "AND noise.name IN ('" + "','"'.join(noises) + "') ".add_()
            query += "GROUP BY tracks.track_name"
            if limit != 0:
                query += " LIMIT %d" % limit
            self._cursor.execute(query)
            ret_val = self._cursor.fetchall()
            self._close()
            return ret_val
        except Exception as error:
            self._close()
            logging.error(str(error), exc_info=True)
            raise

    def modelTrainUpdateStatus(self, id_training_track, status):
        try:
            self._connect()
            query = "UPDATE training_track" + ".add_()
            query += "SET status = '" + status + "' ".add_()
            query += "WHERE id = " + str(id_training_track))
            self._cursor.execute(query)
            self._conn.commit()
            self._close()
        except Exception as error:
            self._close()

```



```

logging.error(str(error), exc_info=True)
raise

def audioBookUpdateStatusByName(self, name, status):
    try:
        self.__connect()
        self.__cursor.execute(
            "SELECT book_n_tracks FROM audio_books_tracks " +
            "WHERE book_name = '" + name + "' " +
            "ORDER BY track_insert_datetime DESC " +
            "LIMIT 1"
        )
        nTracks = self.__cursor.fetchall()
        query = "UPDATE audio_books_tracks ".__add__(
            "SET status = '" + status + "'").__add__(
            "WHERE book_name = '" + name + "'").__add__(
            "ORDER BY insert_datetime DESC ").__add__(
            "LIMIT " + str(int(nTracks[0][0])))
        )
        self.__cursor.execute(query)
        self.__conn.commit()
        self.__close()
    except Exception as error:
        self.__close()
        logging.error(str(error), exc_info=True)
        raise

if __name__ == '__main__':
    bdManager = DBManager()
    bdManager.createDB()

```

Listing A.1: Código la gestión de la base de datos

A.0.2 Códigos del scraper de LibriVox

```

import logging
import time
import random
import os
from selenium import webdriver
from bs4 import BeautifulSoup
import requests
from tqdm import tqdm

logger = logging.getLogger('LibrivoxScrapper')

class Book:
    def __init__(self, dataInfo, downloadInfo):
        self.title = dataInfo['title']
        self.authorName = dataInfo['author'][name]
        self.authorUrl = dataInfo['author'][url]
        self.state = dataInfo['metadata'][state]
        self.type = dataInfo['metadata'][type]
        self.language = dataInfo['metadata'][language]
        self.dummy = ''
        self.nTracks = 0
        if downloadInfo:
            self.url = downloadInfo[url]
            self.size = downloadInfo[size]
        else:
            self.url = ''
            self.size = ''

class Track(Book):
    def __init__(self, dataInfo, downloadInfo):
        super().__init__(dataInfo, downloadInfo)
        self.name = ''
        self.channels = 0
        self.sampleRate = 0
        self.duration = 0
        self.path = ''
        self.zip = ''

```

```

class LibrivoxScraper:
    def __init__(self, webDriver=''):
        self.__mainURL = r"https://librivox.org/search?"
        self.__primaryKey = r"primary_key="
        self.__searchCategory = "&search_category="
        self.__searchPage = "&search_page="
        self.__searchForm = "&search_form=get_results"
        self.__languages = {'english' : 1,
                           'french' : 2,
                           'german' : 3,
                           'italian' : 4,
                           'spanish' : 5}
        self.__categories = {'author' : 'author',
                             'title' : 'title',
                             'language' : 'language'}
        self.__wrongInit = None
        self.__driverPath = None
        self.__browser = None
        self.setDriver(webDriver)
        self.__books = []

    def setDriver(self, webDriver=''):
        self.__driverPath = webDriver
        if not os.path.exists(self.__driverPath):
            print('Driver does not exist')
            self.__wrongInit = True
        else:
            self.__wrongInit = False

    def __bookExists(self, title):
        for book in self.__books:
            if book.title == title:
                return True
        return False

    def __parseDownload(self, result):
        downloadBtn = result.findAll('div', {'class': 'download-btn'})
        if len(downloadBtn) != 1:
            return None
        else:
            spanNode = downloadBtn[0].findAll('span')
            if len(spanNode) != 1:
                return None
            downloadSize = spanNode[0].contents[0]
            downloadUrl = downloadBtn[0].contents[1]['href']
            downloadInfo = {'url' : downloadUrl,
                            'size' : downloadSize}
            return downloadInfo

    def __parseData(self, result):
        data = result.findAll('div', {'class': 'result-data'})
        bookTitle = data[0].findAll('h3')[0].findAll('a')[0].contents[0]
        bookAuthor = data[0].findAll('p', {'class': 'book-author'})
        if len(bookAuthor[0]) == 1:
            authorName = bookAuthor[0].contents[0].strip()
            authorUrl = ''
        else:
            authorName = bookAuthor[0].contents[1].strip()
            authorUrl = bookAuthor[0].contents[1]['href']
        bookMeta = data[0].findAll('p', {'class': 'book-meta'})
        values = [x.strip() for x in bookMeta[0].contents[0].strip().split('|')]
        authorDict = {'name' : authorName,
                      'url' : authorUrl}
        metadataDict = {'state' : values[0],
                        'type' : values[1],
                        'language' : values[2]}
        dataInfo = {'title' : bookTitle,
                    'author' : authorDict,
                    'metadata' : metadataDict}
        return dataInfo

    def getBooksByLanguage(self, language):
        if self.__wrongInit:
            logger.error('Driver does not exist')
            return None
        self.__browser = webdriver.Chrome(self.__driverPath)
        if language not in self.__languages.keys():
            logger.error('Unknown requested language. Known languages are:\n' +
                         '\tself.__languages.keys()')
        return None
        availablePage = True
        page = 1

```

```

increaseSleep = 0
lastPage = 0
while availablePage:
    logger.info('Retrieving information for language ' + language + ' and page ' + str(page))
    url = self.__mainURL + \
        self.__primaryKey + str(self.__languages[language]) + \
        self.__searchCategory + self.__categories['language'] + \
        self.__searchPage + str(page) + \
        self.__searchForm
    self.__browser.get(url)
    time.sleep(increaseSleep + random.uniform(0.5, 1))
    soup = BeautifulSoup(self.__browser.page_source, "html.parser")
    results = soup.findAll('li', {'class': 'catalog-result'})
    lastPageNode = soup.findAll('a', {'class': 'page-number last'})
    if len(lastPageNode) == 0:
        logger.debug('Enough time, waiting one second more')
        increaseSleep += 1
        continue
    else:
        increaseSleep = 0
    if lastPage == 0:
        lastPage = int(lastPageNode[0]['data-page-number'])
        logger.info('Retrieved ' + str(lastPage) + ' pages for target language')
        logger.info('Retrieved ' + str(len(results)) + ' books for page ' + str(page) +
                   ' for ' + language + ' language')
    for result in results:
        downloadInfo = self.__parseDownload(result)
        dataInfo = self.__parseData(result)
        if dataInfo:
            if not self.__bookExists(dataInfo['title']):
                if not downloadInfo:
                    logger.warning('Retrieved information for book ' + dataInfo['title'] +
                                   ' without download information')
                else:
                    logger.info('Retrieved information for book ' + dataInfo['title'] +
                               ' located in ' + downloadInfo['url'])
                    self.__books.append(Book(dataInfo, downloadInfo))
            if page == lastPage:
                availablePage = False
            page += 1
    self.__browser.close()
    return self.__books

def downloadFile(self, url='', filename='', downloadLib='requests'):
    logger.info('Downloading ' + url + ' file with ' + downloadLib + '.\n' +
               'ItStoring data in ' + filename)
    if downloadLib == 'selenium':
        chrome_options = webdriver.ChromeOptions()
        prefs = {'download.default_directory': filename}
        chrome_options.add_experimental_option('prefs', prefs)
        if self.__wrongInit:
            print('Driver does not exist')
            return None
        self.__browser = webdriver.Chrome(self.__driverPath, chrome_options=chrome_options)
        self.__browser.get(url)
        self.__browser.close()
    else:
        response = requests.get(url, stream=True)
        with open(filename, "wb") as handle:
            for data in tqdm(response.iter_content()):
                handle.write(data)

if __name__ == '__main__':
    myScraper = LibrivoxScraper(r"C:\Program Files (x86)\Google\ChromeDriver\chromedriver.exe")
    books = myScraper.getBooksByLanguage('spanish')
    print(len(books))

```

Listing A.2: Código de LibrivoxScraper

A.0.3 Códigos del gestor de audios

```

import audioread
import copy
import logging
import os

```

```

import zipfile

from src.DBManager import DBManager
from src.LibrivoxScraper import LibrivoxScraper, Track

logger = logging.getLogger('AudioBooksManager')

class AudioBooksManager:
    def __init__(self, db=DBManager(), driver=None):
        self.db = db
        self.__languages = ['spanish']
        self.__librivoxScraper = LibrivoxScraper(driver)
        self.__books = {}

    def __getBooks(self, language):
        self.__books[language] = self.__librivoxScraper.getBooksByLanguage(language)

    def __removeDuplicates(self, language=''):
        urls = []
        for book in self.__books[language]:
            if book.url in urls:
                logging.info('Book ' + book.title + ' already exist in url ' + book.url + '. Removing this book')
                self.__books[language].remove(book)
            else:
                logging.info('Registering url ' + book.url)
                urls.append(book.url)

    def downloadData(self, dstPath='./downloads', sizeMB=19990):
        try:
            downloadNow = False
            sizeMBDownloaded = 0
            if not os.path.exists(dstPath):
                logger.info('Destination path does not exist. Create folder ' + dstPath)
                os.mkdir(dstPath)
            else:
                logger.info('Destination path already exists.')
            for language in self.__languages:
                logger.info('Getting books for language ' + language)
                self.__getBooks(language)
                logger.info('Removing books with the same url')
                self.__removeDuplicates(language)
                for book in self.__books[language]:
                    trackList = []
                    if book.url == '':
                        continue
                    filename = os.path.join(dstPath, os.path.basename(book.url))
                    book.dummy = os.path.splitext(os.path.basename(book.url))[0]
                    if not os.path.isfile(filename):
                        logger.info('File ' + filename + ' does not exist. Downloading it')
                        downloadNow = True
                        self.__librivoxScraper.downloadFile(url=book.url, filename=filename, downloadLib='requests')
                    else:
                        logger.info('File ' + filename + ' already exists')
                        bookFolder = os.path.join(dstPath, book.dummy)
                        if not os.path.exists(bookFolder):
                            logger.info('Folder ' + bookFolder + ' does not exist. Creating it')
                            os.mkdir(bookFolder)
                        else:
                            logger.info('Folder ' + bookFolder + ' already exists')
                            try:
                                with zipfile.ZipFile(filename, 'r') as zipId:
                                    fileList = zipId.namelist()
                                    for file in fileList:
                                        trackPath = os.path.join(bookFolder, file.title())
                                        if not os.path.isfile(trackPath):
                                            logger.info('Unzipping file ' + file.title())
                                            zipId.extract(file, bookFolder)
                                        else:
                                            logger.info('File ' + file.title() + ' already exists do not unzip')
                            except zipfile.BadZipFile as e:
                                logging.warn(str(e) + ' Avoiding file ' + filename)
                                continue
                            if downloadNow:
                                if self.db.audioBookExist(book.dummy):
                                    logger.info('Data base entree already exists and file just downloaded, updating status ' +
                                               'for old register')
                                    self.db.audioBookUpdateStatusByName(book.dummy, 'DELETED')
                                if not (not downloadNow and self.db.audioBookExist(book.dummy)):
                                    logger.info('Creating new data base entree for all tracks within file ' + bookFolder)
                                    for trackFile in os.listdir(bookFolder):
                                        track = copy.copy(book)
                                        track.__class__ = Track

```

```

track.name = trackFile
track.path = os.path.join(bookFolder, trackFile)
track.nTracks = len(os.listdir(bookFolder))
track.zip = filename
with audioread.audio_open(track.path) as fId:
    track.channels = fId.channels
    track.sampleRate = fId.samplerate
    track.duration = fId.duration
trackList.append(track)
logger.info('Found file ' + trackFile + 'with:' +
           '\n\tPath' + track.path +
           '\n\tNTracks' + str(track.nTracks) +
           '\n\tZip' + track.zip +
           '\n\tChannels' + str(track.channels) +
           '\n\tSample Rate [sps]' + str(track.sampleRate) +
           '\n\tDuration [secs]' + str(track.duration))
self.db.audioBookCreate(trackList)
sizeMBDownloaded += os.path.getsize(filename)/(10**6)
logger.info('Total size downloaded %.3f MBytes' % (sizeMBDownloaded))
if sizeMBDownloaded > sizeMB:
    logger.info('Total size downloaded %.3f MBytes exceeds requested target %.3f MBytes' %
                (sizeMBDownloaded, sizeMB))
    break
if sizeMBDownloaded > sizeMB:
    break
except Exception as e:
    logging.error(str(e), exc_info=True)
    raise

if __name__ == '__main__':
    audioBooksManager = AudioBooksManager(driver=r"C:\Program Files (x86)\Google\ChromeDriver\chromedriver.exe")
    audioBooksManager.downloadData()

```

Listing A.3: Código del gestor de las pistas de los audiolibros

A.0.4 Código del gestor de ruido

```

from __future__ import unicode_literals
import audioread
import logging
import os
import youtube_dl

from src.DBManager import DBManager

logger = logging.getLogger('NoiseManager')

class NoiseManager:
    def __init__(self, db=DBManager()):
        self.resources = {
            'airDryer': 'https://www.youtube.com/watch?v=PNAQqh2h3AA',
            'serverRoom': 'https://www.youtube.com/watch?v=gLvxiiUsrc',
            'coffeeShop': 'https://www.youtube.com/watch?v=B0dLmxxy06H0',
            'crowd': 'https://www.youtube.com/watch?v=IKB3Qiglyro',
            'peopleTalking': 'https://www.youtube.com/watch?v=PHBJNN-M_Mo',
            'cityRain': 'https://www.youtube.com/watch?v=eeZ4Q_58UTU',
            'city1': 'https://www.youtube.com/watch?v=cDWZkXjDySc',
            'city2': 'https://www.youtube.com/watch?v=YF3pj_3mdMc',
            'city3': 'https://www.youtube.com/watch?v=8s5H76F3SIs',
            'city4': 'https://www.youtube.com/watch?v=VgimpD1BICI'
        }
        self.ydl_opts = {
            'format': 'bestaudio/best',
            ''''postprocessors': [
                {'key': 'FFmpegExtractAudio',
                 'preferredcodec': 'mp3',
                 'preferredquality': '192'},
            ],
            'outtmpl': '',
        }
        self.db = db

    def downloadData(self, dstPath='./downloads'):
        downloadNow = False

```

```

if not os.path.exists(dstPath):
    logger.info('Destination path does not exist. Create folder ' + dstPath)
    os.mkdir(dstPath)
else:
    logger.info('Destination path already exists.')
for key in self.resources:
    filename = os.path.join(dstPath, key)
    if not os.path.isfile(filename):
        logger.info('File ' + filename + ' does not exist. Downloading it')
        downloadNow = True
        self.ydl_opts['outtmpl'] = filename
        with youtube_dl.YoutubeDL(self.ydl_opts) as ydl:
            success = ydl.download([self.resources[key]])
    else:
        logger.info('File ' + filename + ' already exists')
    if downloadNow:
        if self.db.noiseExist(key):
            logger.info('Data base entree already exists and file just downloaded, updating status ' +
                       'for old register')
            self.db.noiseUpdateStatusByName(key, 'DELETED')
if not (not downloadNow and self.db.noiseExist(key)):
    with audioread.audio_open(filename) as fId:
        logger.info('Creating new data base entree for noise file ' + filename + ' with:' +
                   '\n\tChannels ' + str(fId.channels) +
                   '\n\tSample Rate [sps] ' + str(fId.samplerate) +
                   '\n\tDuration [secs] ' + str(fId.duration))
        self.db.noiseCreate(key, self.resources[key],
                           filename, fId.channels,
                           fId.samplerate, fId.duration)

if __name__ == '__main__':
    noiseManager = NoiseManager()
    noiseManager.downloadData()

```

Listing A.4: Código del gestor de ruido

A.0.5 Código de la preparación de los datos para el entrenamiento

```

import logging
import os
import numpy as np
from argparse import ArgumentParser
from scipy.signal import decimate, spectrogram, get_window
from librosa.core import amplitude_to_db
from pydub import AudioSegment, effects
from h5py import File
from src.errors import ResamplingError
from src.DBManager import DBManager
from src.AudioBooksManager import AudioBooksManager
from src.NoiseManager import NoiseManager

logger = logging.getLogger('DataConverter')

class DataManager:
    def __init__(self):
        self.__INPUT_SAMPLING_RATE = int(11025)
        self.__N_SAMPLES_WINDOW = int(1024)
        self.__N_SAMPLES_OVERLAP = int(0.5 * self.__N_SAMPLES_WINDOW)
        self.__WINDOW = 'hann'
        self.__CHROME_DRIVER_PATH = r"resources/chromedriver"

        self.__db = DBManager()
        self.__audio_manager = AudioBooksManager(self.__db, self.__CHROME_DRIVER_PATH)
        self.__noise_manager = NoiseManager(self.__db)

    def main(self, filename='', mode='', download=0, noises=[], limit=0):
        try:
            if download:
                logging.info('Downloading audio books for training model')
                self.__audio_manager.downloadData()
                logging.info('Downloading noise audios for training model')
                self.__noise_manager.downloadData()
            logging.info('Retrieving audio-noise combinations')
            file_combinations = self.__db.modelTrainGetCombination(self.__INPUT_SAMPLING_RATE, noises, limit)

```

```

with File(filename, mode) as f:
    logging.info('Creating group for SPS:%d and FFT:%d' % (self._INPUT_SAMPLING_RATE,
                                                               self._N_SAMPLES_WINDOW))
    main_group = f.create_group(np.string_('SPS%dFFT%d' % (self._INPUT_SAMPLING_RATE,
                                                               self._N_SAMPLES_WINDOW)))
    main_group.attrs.create(np.string_('SAMPLE_RATE'), np.string_(self._INPUT_SAMPLING_RATE))
    main_group.attrs.create(np.string_('FFT_SIZE'), np.string_(self._N_SAMPLES_WINDOW))
    for idx, file_combination in enumerate(file_combinations):
        try:
            logging.info('Loading data')
            clean_info = self._db.audioBookGetById(file_combination[1])
            clean = self.load_audio(clean_info[0][9], normalized=False)
            if idx > 0:
                if file_combination[2] != file_combinations[idx - 1][2]:
                    noise_info = self._db.noiseGetById(file_combination[2])
                    noise = self.load_audio(noise_info[0][3], normalized=False)
            else:
                noise_info = self._db.noiseGetById(file_combination[2])
                noise = self.load_audio(noise_info[0][3], normalized=False)

            if clean.duration_seconds > noise.duration_seconds:
                logging.info('Clipping clean audio to fit noise audio duration')
                clean = clean[:noise.duration_seconds]

            logging.info('Overlaying noise and clean audios')
            dirty = clean.overlay(noise)
            clean_samples = np.array(clean.get_array_of_samples(), dtype=np.float32)
            clean_sampling_rate = clean.frame_rate
            dirty_samples = np.array(dirty.get_array_of_samples(), dtype=np.float32)
            dirty_sampling_rate = dirty.frame_rate
            logging.info('Processing data')
            dirty_freq, dirty_time, dirty_db, dirty_phase = self._prepreateInput(dirty_samples,
                                                                               dirty_sampling_rate)
            clean_freq, clean_time, clean_db, clean_phase = self._prepreateInput(clean_samples,
                                                                               clean_sampling_rate)
            logging.info('Storing data')
            self._store_h5_data(main_group, file_combination, clean_info[0], noise_info[0],
                                clean_freq, clean_time, clean_db, clean_phase,
                                dirty_freq, dirty_time, dirty_db, dirty_phase)
        except ResamplingError as e:
            logging.warning(str(e), exc_info=True)

    except Exception as e:
        logging.error(str(e), exc_info=True)
        raise

def __resample(self, input_signal, input_sampling_rate):
    if input_sampling_rate % self._INPUT_SAMPLING_RATE:
        raise ResamplingError('Downsampling factor is not integer number\n'
                              '\tInput sampling rate: %d\n' % input_sampling_rate +
                              '\tTarget sampling rate: %d\n' % self._INPUT_SAMPLING_RATE)
    factor = input_sampling_rate / self._INPUT_SAMPLING_RATE
    logger.info('Input sampling rate is different from the expected by the model.\n' +
               '\tInput sampling rate: ' + str(input_sampling_rate) + '\n' +
               '\tModel sampling rate: ' + str(self._INPUT_SAMPLING_RATE) + '\n' +
               '\tResampling input signal by factor: ' + str(factor))
    in_signal = decimate(input_signal, int(factor))
    return in_signal

def __prepreateInput(self, input_signal, sampling_rate):
    if sampling_rate != self._INPUT_SAMPLING_RATE:
        input_signal = self._resample(input_signal, sampling_rate)
    freq, time, stft = spectrogram(
        input_signal, fs=self._INPUT_SAMPLING_RATE,
        window=get_window(self._WINDOW, self._N_SAMPLES_WINDOW),
        nperseg=None,
        noverlap=self._N_SAMPLES_OVERLAP, nfft=self._N_SAMPLES_WINDOW,
        # detrend='constant',
        return_onesided=True, scaling='spectrum', axis=-1, mode='complex')
    db_values = amplitude_to_db(np.abs(stft))
    db_values = np.transpose(db_values)[ :, np.newaxis, : ]
    phase = np.angle(stft)
    return [freq, time, db_values, phase]

def __store_h5_data(self, main_group, file_combination, clean_info, noise_info,
                   clean_freq, clean_time, clean_db, clean_phase,
                   dirty_freq, dirty_time, dirty_db, dirty_phase):
    combination_group = main_group.create_group(np.string_('COMBINATION@ID_%d' % file_combination[0]))
    combination_group.attrs.create(np.string_('COMBINATIONID'), np.int32(file_combination[0]))
    combination_group.attrs.create(np.string_('COMBINATIONSAMPLE_RATE'), np.float64(self._INPUT_SAMPLING_RATE))
    combination_group.attrs.create(np.string_('CLEAN@ID'), np.int32(clean_info[0]))
    combination_group.attrs.create(np.string_('CLEAN@BOOK_DUMMY_NAME'), np.string_(clean_info[1]))
    combination_group.attrs.create(np.string_('CLEAN@BOOK_NAME'), clean_info[2])

```

```

combination_group.attrs.create(np.string_('CLEAN@BOOK_AUTHOR'), clean_info[3])
combination_group.attrs.create(np.string_('CLEAN@BOOK_URL'), np.string_(clean_info[4]))
combination_group.attrs.create(np.string_('CLEAN@BOOK_LANGUAGE'), clean_info[5])
combination_group.attrs.create(np.string_('CLEAN@BOOK_N_TRACK'), np.int32(clean_info[7]))
combination_group.attrs.create(np.string_('CLEAN@TRACK_NAME'), np.string_(clean_info[8]))
combination_group.attrs.create(np.string_('CLEAN@TRACK_SAMPLE_RATE'), np.float64(clean_info[11]))
combination_group.attrs.create(np.string_('NOISE@ID'), np.int32(noise_info[0]))
combination_group.attrs.create(np.string_('NOISE@NAME'), noise_info[1])
combination_group.attrs.create(np.string_('NOISE@URL'), np.string_(noise_info[2]))
combination_group.attrs.create(np.string_('NOISE@ORIGINAL_N_CHANNEL'), np.int8(noise_info[4]))
combination_group.attrs.create(np.string_('NOISE@ORIGINAL_SAMPLE_RATE'), np.float64(noise_info[5]))
clean_group = combination_group.create_group(r'CLEAN')
clean_group.create_dataset('FREQ', data=clean_freq)
clean_group.create_dataset('TIME', data=clean_time)
clean_group.create_dataset('DB', data=clean_db)
clean_group.create_dataset('PHASE', data=clean_phase)
clean_group.attrs.create(np.string_('FFT@SIZE'), np.int32(self._N_SAMPLES_WINDOW))
clean_group.attrs.create(np.string_('FFT@N_SAMPLES_OVERLAP'), np.int32(self._N_SAMPLES_OVERLAP))
clean_group.attrs.create(np.string_('FFT@WINDOW'), np.string_(self._WINDOW))
dirty_group = combination_group.create_group(r'DIRTY')
dirty_group.create_dataset('FREQ', data=dirty_freq)
dirty_group.create_dataset('TIME', data=dirty_time)
dirty_group.create_dataset('DB', data=dirty_db)
dirty_group.create_dataset('PHASE', data=dirty_phase)
dirty_group.attrs.create(np.string_('FFT@SIZE'), np.int32(self._N_SAMPLES_WINDOW))
dirty_group.attrs.create(np.string_('FFT@N_SAMPLES_OVERLAP'), np.int32(self._N_SAMPLES_OVERLAP))
dirty_group.attrs.create(np.string_('FFT@WINDOW'), np.string_(self._WINDOW))

@staticmethod
def load_audio(path, normalized=True):
    ext = os.path.splitext(path)[1][1:]
    logging.info('Loading audio ' + path + ' with file type ' + ext)
    rawSound = AudioSegment.from_file(path, ext)
    if rawSound.channels != 1:
        logging.info('Audio contains more than one channel. Setting to single channel')
        rawSound = rawSound.set_channels(1)
    if normalized:
        logging.info('Normalize audio')
        return effects.normalize(rawSound)
    else:
        return rawSound

if __name__ == "__main__":
    try:
        # set up logging to file
        logging.basicConfig(level=logging.DEBUG,
                            format='%(asctime)s %(name)-20s %(levelname)-8s %(message)s',
                            datefmt='%m-%d %H:%M',
                            filename='./DataConverter.log',
                            filemode='w+')
        # define a Handler which writes DEBUG messages or higher to the sys.stderr
        console = logging.StreamHandler()
        console.setLevel(logging.DEBUG)
        # set a format which is simpler for console use
        formatter = logging.Formatter('%(asctime)s %(name)-20s %(levelname)-8s %(message)s')
        # tell the handler to use this format
        console.setFormatter(formatter)
        # add the handler to the root logger
        logging.getLogger('').addHandler(console)

        parser = ArgumentParser()
        parser.add_argument("-d", "--download", action='count', help="Download data and log into database", default=0)
        parser.add_argument("-f", "--file", help="H5 file name", default='./h5_default.h5')
        parser.add_argument("-m", "--mode", choices=['r', 'r+', 'w', 'a'], help="Mode of opening h5 file", default='a')
        parser.add_argument("-n", "--noise", help="Noises to mix in h5 file", type=str, nargs='+')
        parser.add_argument("-l", "--limit", help="Number of tracks (0 means all)", type=int, default=0, )
        args = parser.parse_args()

        logging.info('Starting program execution')
        data_manager = DataManager()
        data_manager.main(filename=args.file, mode=args.mode, download=args.download,
                          noises=args.noise, limit=args.limit)
    except Exception as e:
        logging.error('Something was wrong', exc_info=True)

```

Listing A.5: Código del conversor de datos para el entrenamiento del modelo

A.0.6 Código del generador de secuencias

```

import logging
from collections import deque
from tensorflow.keras.utils import Sequence
import numpy as np
from h5py import File as H5File

logger = logging.getLogger('LSTM-based Model')

class GroupInfo:
    def __init__(self):
        self.path = ''
        self.clean_shape = None
        self.dirty_shape = None
        self.n_fft = 0
        self.attributes = dict()

class DataGenerator(Sequence):
    """Generates data for Keras"""
    def __init__(self, h5_file_path, batch_size=32, shuffle=True, fft_size=0, n_time_steps=1, n_batches=0):
        self.batch_size = batch_size
        self.h5_file_path = h5_file_path
        self.shuffle = shuffle
        self.fft_size = fft_size
        self.n_time_steps = n_time_steps
        self.n_batches = n_batches
        self.h5_info = None
        self.groups = []
        self.n_fft = 0
        self.group_idx = 0
        self.target_group = None
        self.fft_idx = 0
        self.batch = deque(maxlen=n_time_steps)

    def __startup__(self):
        with H5File(self.h5_file_path, 'r') as f:
            for main_group_key in f.keys():
                main_group = f[main_group_key]
                for group_key in main_group.keys():
                    group = main_group[group_key]
                    self.__parse_group_info(group)
                    if self.n_batches != 0 and (self.n_fft / self.batch_size) >= self.n_batches:
                        break

        self.target_group = self.groups[0]

    def __parse_group_info(self, h5_group):
        group_info = GroupInfo()
        group_info.path = h5_group.name
        group_info.clean_shape = h5_group['CLEAN/DB'].shape
        group_info.dirty_shape = h5_group['DIRTY/DB'].shape
        for key in h5_group.attrs.keys():
            group_info.attributes[key] = h5_group.attrs[key]
        if group_info.clean_shape == group_info.dirty_shape:
            logger.info('Found %d FFTs in %s' % (group_info.clean_shape[0], group_info.path))
            group_info.n_fft = group_info.clean_shape[0]
            self.n_fft += group_info.n_fft
            self.groups.append(group_info)
        else:
            logger.info('Number of FFTs mismatch\n\tClean %d\n\tDirty %d' %
                       (group_info.clean_shape[0], group_info.dirty_shape[0]))

    def __increment_target_group(self):
        self.group_idx += 1
        self.target_group = self.groups[self.group_idx]
        self.fft_idx = 0

    def __len__(self):
        """Denotes the number of batches per epoch"""
        n_batches = int(np.floor((self.n_fft - self.n_time_steps + 1) / self.batch_size))
        #logger.debug('Number of batches %d' % n_batches)
        return n_batches

    def __getitem__(self, index):
        """Generate one batch of data"""

```

```

# Generate data
X, Y = self.__data_generation()

return X, Y

def on_epoch_end(self):
    'Updates indexes after each epoch'
    pass

def __data_generation(self):
    """Generates data containing in batch_size"""
    X = np.empty((self.batch_size, self.n_time_steps, self.fft_size), dtype=np.float64)
    Y = np.empty((self.batch_size, self.fft_size), dtype=np.float64)

    # Generate data
    with H5File(self.h5_file_path, 'r') as f:
        for idx in range(self.batch_size):
            # Store sample
            if not self.batch:
                for idx_time in range(self.n_time_steps - 1):
                    self.batch.append(f[self.target_group.path + '/DIRTY/DB'][self.fft_idx, :, :self.fft_size])
                    self.fft_idx += 1
            self.batch.append(f[self.target_group.path + '/DIRTY/DB'][self.fft_idx, :, :self.fft_size])
            X[idx, :, :] = self.batch
            Y[idx, :] = f[self.target_group.path + '/CLEAN/DB'][self.fft_idx, 0, :self.fft_size]
            self.fft_idx += 1
            if self.fft_idx >= self.target_group.n_fft:
                self.__increment_target_group()

    return X, Y

```

Listing A.6: Código del generador de secuencias

A.0.7 Código del modelo

```

import logging
from argparse import ArgumentParser
from tensorflow import keras
from tensorflow.keras.callbacks import Callback
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import LSTM
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.regularizers import l2
from src.models.DataGenerator import DataGenerator, GroupInfo

logger = logging.getLogger('LSTM-based Model')

class ModelSaver(Callback):
    def __init__(self, N):
        self.N = N
        self.batch = 0
        self.epoch = 0

    def on_batch_end(self, batch, logs={}):
        if self.batch % self.N == 0:
            name = './checkpoints/LSTM%16d.h5' % self.batch
            logger.info('Saving model %s' % name)
            self.model.save(name)
        self.batch += 1

    def on_epoch_begin(self, epoch, logs=None):
        self.epoch += 1
        name = './checkpoints/LSTM_epoch%5d.h5' % self.epoch
        logger.info('Saving model %s' % name)
        self.model.save(name)

class LSTMModel:
    def __init__(self, checkpoint=None):
        self.batch_size = 32
        self.reg = 0.05
        self.learning_rate = 1e-2
        self.n_units = 512
        self.decay = 1e-3

```

```

self.input_sampling_rate = 11025
self.n_samples_window = 1024
self.n_samples_spectrum = 250
self.overlap = 0.5
self.n_time_steps = 1
self.training_generator = None
self.saver_callback = None
if checkpoint:
    logger.info('Model checkpoint input obtained')
    self.__model = keras.models.load_model(checkpoint.replace('\\', '/'))
else:
    logger.info('Creating new model')
    self.__createModel()

def __createModel(self):
    regularizer = l2(self.reg)
    self.__model = Sequential()
    self.__model.add(LSTM(self.n_units, input_shape=(self.n_time_steps, self.n_samples_spectrum),
                         return_sequences=True, recurrent_dropout=0.3, activation='relu'))
    self.__model.add(LSTM(self.n_units,
                         return_sequences=True, recurrent_dropout=0.3, activation='relu'))
    self.__model.add(LSTM(self.n_units,
                         return_sequences=False, recurrent_dropout=0.3, activation='relu'))
    self.__model.add(Dense(self.n_samples_spectrum, activation='softmax'))

    model_info = []
    self.__model.summary(print_fn=lambda x: model_info.append(x))
    model_info = '\n\n'.join(model_info)
    logger.info(model_info)

    opt = Adam(lr=self.learning_rate,
                decay=self.decay
               )
    self.__model.compile(#loss='mean_squared_error',
                        loss='kullback_leibler_divergence',
                        optimizer=opt,
                        metrics=['acc', 'mse'])
    name = './checkpoints/LSTM_finished.h5'
    self.__model.save(name)

def save(self, filename):
    logger.info('Save model to file ' + filename)
    self.__model.save(filename)

def train(self, file_name, epochs, save, n_batches=0):
    self.training_generator = DataGenerator(h5_file_path=file_name, batch_size=32, fft_size=self.n_samples_spectrum,
                                             n_time_steps=self.n_time_steps, n_batches=n_batches)

    self.saver_callback = ModelSaver(save)
    self.__model.fit(self.training_generator,
                     max_queue_size=50,
                     workers=16, epochs=epochs,
                     callbacks=[self.saver_callback])
    # Fit_generator is deprecated
    '''self.__model.fit_generator(generator=self.training_generator,
                                use_multiprocessing=True,
                                max_queue_size=50,
                                workers=16, epochs=epochs,
                                callbacks=[self.saver_callback])'''

if __name__ == "__main__":
    try:
        # set up logging to file
        logging.basicConfig(level=logging.DEBUG,
                            format='%(asctime)s %(name)-20s %(levelname)-8s %(message)s',
                            datefmt='%m-%d %H:%M',
                            filename='./LSTM.log',
                            filemode='w+')
        # define a Handler which writes DEBUG messages or higher to the sys.stderr
        console = logging.StreamHandler()
        console.setLevel(logging.DEBUG)
        # set a format which is simpler for console use
        formatter = logging.Formatter('%(asctime)s %(name)-20s %(levelname)-8s %(message)s')
        # tell the handler to use this format
        console.setFormatter(formatter)
        # add the handler to the root logger
        logging.getLogger('').addHandler(console)

        parser = ArgumentParser()
        parser.add_argument("-t", "--train", help="Path of the H5 training data", default='')
        parser.add_argument("-e", "--epochs", help="Number of epochs for training the model", type=int, default=20)
        parser.add_argument("-s", "--save", help="Number of epochs for saving the model", type=int, default=400000)
    
```

```

parser.add_argument("-b", "--batch", help="Number of batches for training", type=int, default=0)
parser.add_argument("-l", "--load", help="Load model from checkpoint", default='')
args = parser.parse_args()

logging.info('Starting program execution')
if args.load:
    model = LSTMModel(checkpoint=args.load)
else:
    model = LSTMModel()
if args.train:
    model.train(file_name=args.train, epochs=args.epochs, save=args.save, n_batches=args.batch)

except Exception as e:
    logging.error('Something was wrong', exc_info=True)

```

Listing A.7: Código del modelo

A.0.8 Jupyter Notebook del algoritmo de mayor precisión

```

#!/usr/bin/env python
# coding: utf-8

# In[1]:


import logging
from argparse import ArgumentParser
import numpy as np
from collections import deque
import matplotlib.pyplot as plt
from h5py import File as H5file
from tensorflow import keras
from tensorflow.keras.callbacks import Callback
from tensorflow.keras.models import Sequential
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import LSTM
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import TimeDistributed
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.regularizers import l2


# # 1. Model variables

# In[15]:


file_name = r'/media/ignazio/DATA/dataset_one_noise.h5'

n_samples_spectrum = 250
n_time_steps = 1

n_batches = 525
batch_size = 128

n_epochs = 150
epochs_array = range(n_epochs)

initial_lr = 1e-2
decay_lr = 1
decay = decay_lr/n_epochs
lrs = [initial_lr*(1/(1+decay*i)) for i in epochs_array]

# In[14]:


# Learning rate plot
plt.style.use("ggplot")
plt.figure()
plt.plot(epochs_array, lrs)
plt.title("Learning Rate Schedule")
plt.xlabel("Epoch #")
plt.ylabel("Learning Rate")

```

```

plt.savefig('Learning_rate_schedule_comparison.pdf')

# # 2. Parse and loading data
# ## 2.1 Parse number of fft within file
# In[4]:


class GroupInfo:
    def __init__(self):
        self.path = ''
        self.clean_shape = None
        self.dirty_shape = None
        self.n_fft = 0
        self.attributes = dict()

n_fft = 0
groups = []
with H5File(file_name, 'r') as f:
    for main_group_key in f.keys():
        main_group = f[main_group_key]
        for group_key in main_group.keys():
            group = main_group[group_key]

            group_info = GroupInfo()
            group_info.path = group.name
            group_info.clean_shape = group['CLEAN/DB'].shape
            group_info.dirty_shape = group['DIRTY/DB'].shape
            for key in group.attrs.keys():
                group_info.attributes[key] = group.attrs[key]
            if group_info.clean_shape == group_info.dirty_shape:
                print('Found %d FFTs in %s' % (group_info.clean_shape[0], group_info.path))
                group_info.n_fft = group_info.clean_shape[0]
                n_fft += group_info.n_fft
                groups.append(group_info)
            else:
                print('Number of FFTs mismatch\n\tClean %d\n\tDirty %d' %
                      (group_info.clean_shape[0], group_info.dirty_shape[0]))
        if n_batches != 0 and (n_fft/batch_size) >= n_batches:
            break

# ## 2.2 Data loading
# In[5]:


group_idx = 0
target_group = groups[group_idx]

X = np.empty((n_batches*batch_size, n_time_steps, n_samples_spectrum), dtype=np.float64)
Y = np.empty((n_batches*batch_size, n_samples_spectrum), dtype=np.float64)

fft_idx = 0
circular_buffer = deque(maxlen=n_time_steps)
# Generate data
with H5File(file_name, 'r') as f:
    for idx in range(n_batches*batch_size):
        # Store sample
        if not circular_buffer:
            for idx_time in range(n_time_steps - 1):
                circular_buffer.append(f[target_group.path + '/DIRTY/DB'][fft_idx, :, :n_samples_spectrum])
                fft_idx += 1
            circular_buffer.append(f[target_group.path + '/DIRTY/DB'][fft_idx, :, :n_samples_spectrum])
        X[idx, :, :] = circular_buffer
        Y[idx, :, :] = f[target_group.path + '/DIRTY/DB'][fft_idx, :, :self.n_samples_spectrum]
        fft_idx += 1
        if fft_idx >= target_group.n_fft:
            group_idx += 1
            target_group = groups[group_idx]
            fft_idx = 0

# ## 2.3 Data normalization
# In[6]:


Xnorm = np.empty((n_batches*batch_size, n_time_steps, n_samples_spectrum), dtype=np.float64)
Ynorm = np.empty((n_batches*batch_size, n_samples_spectrum), dtype=np.float64)

```

```

for idx in range(X[0,0,:].shape[0]):
    Xnorm[:, :, idx] = ((X[:, :, idx] - X[:, :, idx].mean())/(X[:, :, idx] - X[:, :, idx].mean()).std())
    Ynorm[:, idx] = ((Y[:, idx] - Y[:, idx].mean())/(Y[:, idx] - Y[:, idx].mean()).std())

stop_idx = int(0.8*n_fft)
x_train = Xnorm[0:stop_idx]
y_train = Ynorm[0:stop_idx]
x_test = Xnorm[stop_idx:]
y_test = Ynorm[stop_idx:]

# # 3. Model
# ## 3.1 Model design

# In[7]:


lstm_units_layer = 512

model = Sequential()
model.add(LSTM(lstm_units_layer,
               input_shape=(n_time_steps, n_samples_spectrum),
               return_sequences=False,
               recurrent_dropout=0.3,
               activation='relu',
               stateful=True,
               kernel_initializer='random_normal',
               bias_initializer='zeros'
              ))
#model.add(LSTM(self.n_units,
#               return_sequences=True, recurrent_dropout=0.3, activation='relu'))
#model.add(LSTM(self.n_units,
#               return_sequences=False, recurrent_dropout=0.3, activation='relu'))
model.add(Dense(n_samples_spectrum, activation='softmax',kernel_initializer='random_normal',
               bias_initializer='zeros'))
model_info = []
model.summary(print_fn=lambda x: model_info.append(x))
model_info = '\n\t'.join(model_info)
print(model_info)

opt = Adam(lr=initial_lr, decay=decay)
model.compile(#loss='mean_squared_error',
             loss='mse',
             optimizer=opt,
             metrics=['acc', 'mse'])

# ## 3.2 Model train

# In[8]:


history = model.fit(x_train, y_train,
                     epochs=n_epochs,
                     max_queue_size=50,
                     workers=16,
                     validation_data=(x_test, y_test),
                     shuffle=False)

# In[9]:


history2 = model.fit(x_train, y_train,
                     epochs=n_epochs,
                     max_queue_size=50,
                     workers=16,
                     validation_data=(x_test, y_test),
                     shuffle=False)

# In[11]:


model.save(r'./checkpoints/LSTM_results.h5')

# In[22]:


acc = np.concatenate((history.history['acc'],history2.history['acc']))
val_acc = np.concatenate((history.history['val_acc'],history2.history['val_acc']))
epochsPlot = range(1,len(acc) + 1)

```

```
plt.figure(num=None, figsize=(8,6), dpi=80, facecolor='w', edgecolor='k')
plt.plot(epochsPlot, acc, 'o', label ='Training accuracy')
plt.plot(epochsPlot, val_acc, label ='Validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Metrics')
plt.title('Training and Validation accuracy')
plt.grid()
plt.legend()

plt.savefig('results_acc_compa.pdf')

# In[23]:


loss = np.concatenate((history.history['loss'],history2.history['loss']))
val_loss = np.concatenate((history.history['val_loss'],history2.history['val_loss']))
epochsPlot = range(1,len(acc) + 1)
plt.figure(num=None, figsize=(8,6), dpi=80, facecolor='w', edgecolor='k')
plt.plot(epochsPlot, loss, 'o', label ='Training loss')
plt.plot(epochsPlot, val_loss, label ='Validation loss')
plt.xlabel('Epochs')
plt.ylabel('Metrics')
plt.title('Training and Validation accuracy')
plt.grid()
plt.legend()

plt.savefig('results_loss_compa.pdf')
```

Listing A.8: Código del Jupyter Notebook de prototipado del modelo de mayor precisión

Glosario

ADC Analogical to Digital Converter. 4, 13, 14

AMD Advanced Micro Devices. 53, 58, 83

API Application Programming Interface. 22, 26

ASCII American Standard Code for Information Interchange. 34

AWS Amazon Web Services. 58

Dataset Conjunto de datos, normalmente estructurados.. 25

DSP Digital Signal Procesing. 9, 14, 15, 20, 43, 46

FFT Fast Fourier Transform. 4, 20–22, 37–39, 44–49, 53, 55

FPGA Field Programmable Gate Array. 58

Framework Un framework, entorno de trabajo o marco de trabajo es un conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática particular que sirve como referencia, para enfrentar y resolver nuevos problemas de índole similar.. 7, 53, 83

GET Es un método de petición del protocolo HyperText Transfer Protocol (HTTP) que se utiliza para solicitar datos a través unos filtros definidos en la URL con la que se hace la petición.. 26, 30

GPS Global Positioning System. 25

GRU Gated Recurrent Unit. 15, 21, 22

HDD Hard Disk Drive. 49

HDF Hierarchical Data Format. 83

HDF5 Hierarchical Data Format (HDF) versión 5. Es un formato binario organizado utilizado para almacenar y acceder a grandes volúmenes de datos de manera indexada. Está estructurado de manera similar a una carpeta, soporta links virtuales para referenciar datasets, valores de relleno por defecto y metadatos, entre otras capacidades.. 5, 46, 49–51

HTML HyperText Markup Language. 26, 27, 30

HTTP HyperText Transfer Protocol. 82, 83

JSON JavaScript Object Notation. 26

LSTM Long Short-Term Memory. 2–4, 15–20, 22, 47–49, 55, 58, 59

ML Machine Learning. 25

RAM Random Access Memory. 49

ROC Radeon Open Compute. 83

ROCM Radeon Open Compute (ROC)m es un Framework de código abierto para desarrolladores utilizado para realizar cálculos computacionales en gráficas AMD Radeon. 53, 58

SNR Signal to Noise Ratio. 38

STFT Short-Time Fourier Transform. 46, 49

URL Uniform Resource Locator. 26–28, 30, 34, 35, 82

Web scraping Es una técnica utilizada mediante programas de software para extraer información de sitios web.1 Usualmente, estos programas simulan la navegación de un humano en la World Wide Web (WWW) ya sea utilizando el protocolo HTTP manualmente, o incrustando un navegador en una aplicación.. 7, 8, 26–28, 58, 59

WWW World Wide Web. 83

Referencias

- AMD. (2020). *Amd rocm platform*. Descargado de link
- Bukkapatnam, A. T. (2020). *Noise reduction using rnns with tensorflow*. link. GitHub.
- Hochreiter, S., y Schmidhuber, J. (1997, 12). Long short-term memory. *Neural computation*, 9, 1735-80. doi: 10.1162/neco.1997.9.8.1735
- Jia, Y., Zhang, Y., Weiss, R. J., Wang, Q., Shen, J., Ren, F., ... Wu, Y. (2019, 1). Transfer learning from speaker verification to multispeaker text-to-speech synthesis. *32nd Conference on Neural Information Processing Systems*.
- Karpathy, A. (2015, 21 de Mayo). *The unreasonable effectiveness of recurrent neural networks*. Descargado de link
- Krisp. (2020). *World's best innovative noise cancellation technology powered by deep neural network*. Descargado de link
- NVIDIA. (2020). *Nvidia rtx voice: Setup guide*. Descargado de link
- Olah, C. (2017, 27 de Agosto). *Understanding lstm networks*. Descargado de link
- Tadokoro, Y., Matsumoto, W., y Yamaguchi, M. (2002). Pitch detection of musical sounds using adaptive comb filters controlled by time delay. , 1, 109-112 vol.1.
- Valin, J.-M. (2018, 5). A hybrid dsp/deep learning approach to real-time full-band speech enhancement. Descargado de link
- Valin, J.-M., y Richards, G. (2019). *Rnnoise*. link. GitHub.

Yuuuyoru, S. (2018). *Noise reduction using gru.* link. GitHub.