Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Separación automática de instrumentos musicales utilizando aprendizaje máquina

Autor: Hugo Guerra Cornejo
Tutor: Francisco José Simois Tirado

Dpto. Teoría de la Señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2023

Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Separación automática de instrumentos musicales utilizando aprendizaje máquina

Autor:
Hugo Guerra Cornejo

Tutor:
Francisco José Simois Tirado
Profesor Contratado Doctor

Dpto. de Teoría de la Señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2023

Trabajo Fin de Grado: Separación automática de instrumentos musicales utilizando aprendizaje máquina

Autor: Hugo Guerra Cornejo

Tutor: Francisco José Simois Tirado

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2023

El Secretario del Tribunal

Agradecimientos

Este trabajo supone el fin de una etapa. Por ello, me gustaría agradecer a todas aquellas personas que han estado conmigo en este periodo:

A mi familia, en especial a mis padres, por sus enseñanzas, cariño y apoyo incondicional en los buenos y malos momentos. Sin ellos no sería posible haber recorrido este camino.

A mis compañeros y amigos, por todos los buenos ratos vividos y por haberme aportado otras visiones de la vida.

A mi tutor, Francisco José Simois, por brindarme la oportunidad de hacer este trabajo y su buena atención conmigo cada vez que lo he necesitado.

Resumen

La separación de fuentes sonoras ha sido en los últimos tiempos, uno de los temas más recurrentes en el ámbito del procesado de señal. El avance en disciplinas tales como el Deep Learning ha supuesto una gran contribución en la mejora de los algoritmos que tratan de recrear la habilidad humana de identificar fuentes sonoras de manera individual.

Este trabajo se centra en abordar el estudio de las distintas técnicas de separación de fuentes musicales en mezclas estereofónicas. En él se presentan las claves para la implementación y evaluación de un modelo que permita separar los diferentes instrumentos musicales de mezclas musicales profesionales. Para ello se comienza analizando las propuestas existentes de sistemas basados en redes neuronales y su rendimiento para, posteriormente, implementar una arquitectura de separación automática de fuentes musicales y realizar al sistema resultante una evaluación de medidas objetivas.

Abstract

Sound source separation has been, in recent times, one of the top trending topics in the field of signal processing. Advances in disciplines such as Deep Learning have made a great contribution to the improvement of algorithms that try to recreate the human ability to identify individual sound sources.

This paper focuses on the study of different techniques for musical source separation in stereo mixtures. It presents the keys for the implementation and evaluation of a model that allows the separation of different musical instruments in professional musical mixtures. For this purpose, existing proposals for neural network-based systems and their performance have been analyzed in order to implement an architecture for the automatic separation of musical sources and perform an objective measurement evaluation of the resulting system.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

ÍNDICE DE ECUACIONES

1 INTRODUCCIÓN

La separación de fuentes sonoras (SSS) es el proceso por el cual se extraen una o varias señales de interés de una mezcla musical que contiene múltiples señales superpuestas. Esto tiene una especial importancia en el procesamiento de señales puesto que eliminar fuentes indeseadas como el ruido o aislar las señales de interés permite operar con señales sin interferencias.

El ser humano cuenta con la habilidad de separar las diferentes fuentes de audio. Esto es algo que en la década de los cincuenta fue denominado efecto de fiesta de cóctel [1]. Aunque es cierto que se han logrado notables avances en los últimos tiempos para recrear esta habilidad en ordenadores, es un campo que continúa en constante desarrollo e investigación.

Una rama concreta enmarcada dentro de la separación de fuentes sonoras es la separación de fuentes musicales (MSS). Esta tiene como objetivo dividir una mezcla de audio musical en los diferentes instrumentos que la componen. Esta separación tiene aplicaciones tales como la realización de un "karaoke", el remezclado de canciones, la generación de sonido envolvente, la restauración de grabaciones antiguas, el análisis de instrumentos por separado y un largo etcétera. Por ello la separación de fuentes musicales es un tema ampliamente estudiado dentro de la comunidad de recuperación de información musical (MIR) y sobre el que habitualmente se organizan retos y campañas de evaluación tales como la SISEC [2].

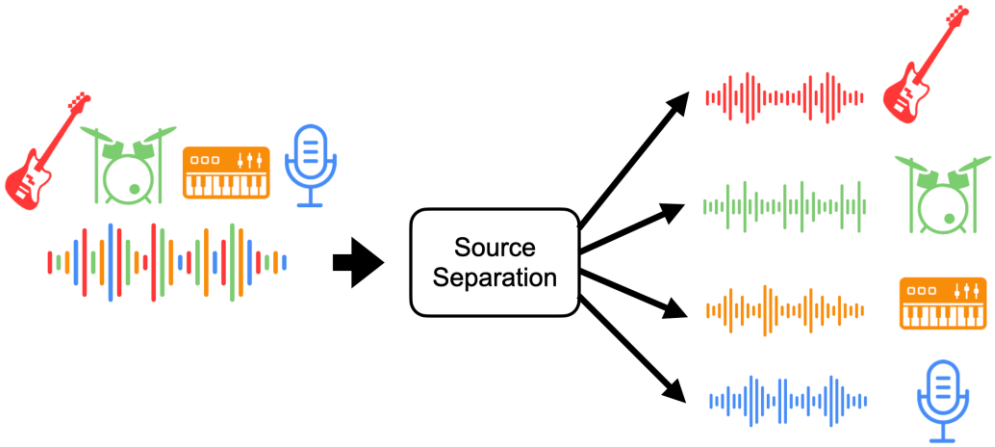


Figura 1-1. Separación de fuentes musicales [3].

En este trabajo nos centraremos en la separación de instrumentos (fuentes) musicales a partir de mezclas de audio estereofónicas. En concreto, partiendo de dichas mezclas trataremos de obtener archivos de audio correspondientes a tres fuentes musicales: voz, batería y bajo. Además, separaremos una cuarta fuente relativa al resto de instrumentos presentes en la composición musical.

Siguiendo con este apartado introductorio, a continuación, se enumerarán de manera sintetizada los distintos objetivos que se pretenden alcanzar en este trabajo y, finalmente, se pasará a detallar la manera en la que está estructurada la memoria.

1.1 Objetivos

Los principales objetivos que este trabajo pretende cubrir son los siguientes:

1. Realizar un estudio del estado del arte de las principales técnicas de separación de fuentes sonoras haciendo hincapié en las enfocadas a separación de instrumentos musicales.
2. Analizar y comprender dos de los principales algoritmos de separación de fuentes musicales existentes.
3. Generar un código en Python de un modelo que permita realizar la separación automática de instrumentos.
4. Evaluar de manera objetiva los resultados obtenidos a partir de un conjunto de datos de prueba.

1.2 Organización de la memoria

La memoria del presente trabajo se estructura en los siete capítulos que se describen a continuación:

El primer capítulo, en el que se encuentra esta sección, aporta una breve *introducción* del tema a tratar en el trabajo, así como los propósitos que se tienen y un resumen de cómo se abordará la materia en las distintas secciones de la memoria, para que el lector pueda comprender de manera gradual la temática estudiada.

El segundo capítulo se centrará en los *fundamentos teóricos* previos a la presentación del trabajo realizado. En él se introducirán diferentes conceptos relacionados con las redes neuronales la representación y el enmascarado de audio. Así mismo, se presentarán las medidas para realizar una evaluación objetiva de la separación.

Posteriormente, en el tercer capítulo, se hará una revisión resumida del *estado del arte* relativo a la separación de fuentes sonoras haciendo especial mención a los algoritmos de separación de instrumentos musicales.

En el cuarto capítulo de esta memoria, se detallará la *metodología* empleada, mencionando los distintos requisitos necesarios para la realización de este trabajo y se describirán detalladamente las arquitecturas de redes neuronales seleccionadas para nuestra tarea que nos ocupa.

A lo largo del quinto capítulo se comentarán los detalles relativos al *entrenamiento* de los distintos modelos seleccionados, así como del modelo que se propone.

En el sexto capítulo se expondrán los *resultados* obtenidos con las arquitecturas con las que se han trabajado y una posterior comparación con otros modelos existentes, haciendo uso de medidas de evaluación objetivas.

Las *conclusiones*, así como las *líneas futuras de investigación* quedarás recogidas en el séptimo y último capítulo.

Además, se aportará un *anexo* en el que se incluirá el código en Python del modelo de red neuronal implementado

Junto con los siete capítulos y el anexo que se acaba de mencionar, la memoria contará con un apartado dedicado a las referencias empleadas a lo largo de la misma, índices de tablas, figuras y ecuaciones, además de un glosario.

2 FUNDAMENTOS TEÓRICOS

A lo largo de este capítulo se introducirán aquellos conceptos fundamentales para comprender los sistemas de separación automático de fuentes musicales mediante aprendizaje máquina como los que abordaremos más adelante. Debido a que existen multitud de elementos susceptibles de ser usados en estos procedimientos de separación, se explicaran con mayor profundidad aquellos que intervienen en los sistemas tratados en este trabajo. En primer lugar, se presentarán los fundamentos básicos del aprendizaje máquina y de las redes neuronales, y posteriormente, se abordarán las cuestiones relacionadas con la representación y el enmascaramiento en tiempo-frecuencia. Para finalizar este capítulo teórico, se comentarán las métricas que emplearemos para evaluar objetivamente los modelos tratados en este trabajo.

2.1 Aprendizaje Máquina

El *aprendizaje máquina* (Machine Learning) es una rama de la inteligencia artificial (IA) que consiste en enseñar a equipos informáticos a que aprendan de la experiencia. Para ello, sus algoritmos se apoyan en métodos computacionales enfocados a aprender información directamente a través de datos, sin necesitar para ello una ecuación predeterminada como modelo.

Su rendimiento se optimiza de manera adaptativa conforme aumenta el número de muestras con las que cuentan para el aprendizaje [4].

Estos algoritmos pueden clasificarse en cuatro tipos de aprendizaje dependiendo de la salida esperada, así como del tipo de entrada:

- **Aprendizaje supervisado**. El sistema aprende en base un conjunto de datos de entrenamiento etiquetados y definidos para evaluar las correlaciones, especificándose las entradas y salidas del algoritmo. Este tipo de aprendizaje tiene como ventaja la simplicidad y facilidad de diseño, pero supone el desafío de etiquetar la gran cantidad de datos que necesita.
- **Aprendizaje no supervisado**. El sistema detecta patrones y categoriza datos a partir de un conjunto de datos entrenamiento no etiquetados, sin disponer de los datos de salida deseados. Si bien poseen una configuración fácil dado que no requieren etiquetado, estos modelos no suelen ofrecer predicciones precisas.
- **Aprendizaje semisupervisado**. En este tipo de aprendizaje se combinan los dos tipos de aprendizaje anteriormente comentados. En el entrenamiento se usan una pequeña cantidad de datos etiquetados y una gran cantidad sin etiquetar de manera que, una vez entrenado parcialmente el algoritmo con datos etiquetados, este se es capaz de etiquetar los datos no etiquetados.
- **Aprendizaje por refuerzo**: Se aprende mediante recompensas, de manera que el objetivo del modelo es conseguir el mayor número de puntos de recompensas posible y alcanzar una meta [5].

El conjunto de datos con el que trabajan estos algoritmos suele dividirse en los siguientes subconjuntos:

- **Datos de entrenamiento**, sobre los que se realiza la optimización de los parámetros del modelo (pesos y umbrales).
- **Datos de validación**, que sirven para verificar el comportamiento del modelo y así poder ajustar los hiperparámetros en base a las predicciones que el modelo realiza sobre ellos.
- **Datos de prueba**, sobre los que se realiza la evaluación final del modelo.

Dentro del campo del aprendizaje máquina, existe una variedad especializada denominada *aprendizaje profundo* (Deep Learning) que aumenta la precisión de los algoritmos hasta el punto de superar en ocasiones al propio rendimiento humano. Sus modelos entrenan con una gran cantidad de datos etiquetados y estos se encargan por sí solos de extraer las características a partir de los datos empleando para ello arquitecturas basadas en *redes neuronales* que se describirán en el siguiente punto.

El avance en los últimos años del aprendizaje profundo (DL) ha supuesto que los algoritmos tradicionales empleados en la separación de fuentes tales como el análisis de componentes principales (PCA) [6] o la factorización de matrices no negativas (NMF) [7], se hayan visto sustituidas por arquitecturas de redes neuronales. Esto se debe, en gran medida, a que estos algoritmos de aprendizaje son capaces de proporcionar implementaciones más rápidas y complejas que los anteriores ofreciendo mejores resultados.

2.1.1 Redes neuronales

Las redes neuronales, también llamadas *redes neuronales artificiales* (ANN), son un subconjunto del aprendizaje máquina y representan una parte fundamental en los algoritmos de aprendizaje profundo. Se trata de sistemas inspirados en la estructura y el funcionamiento del cerebro humano. Las ANN están basadas en entrenar datos con el fin de aprender y mejorar su precisión con el tiempo. Una vez ajustados de manera precisa, son consideradas potentes herramientas en el mundo de la inteligencia artificial que permiten clasificar y agrupar datos a gran velocidad [8].

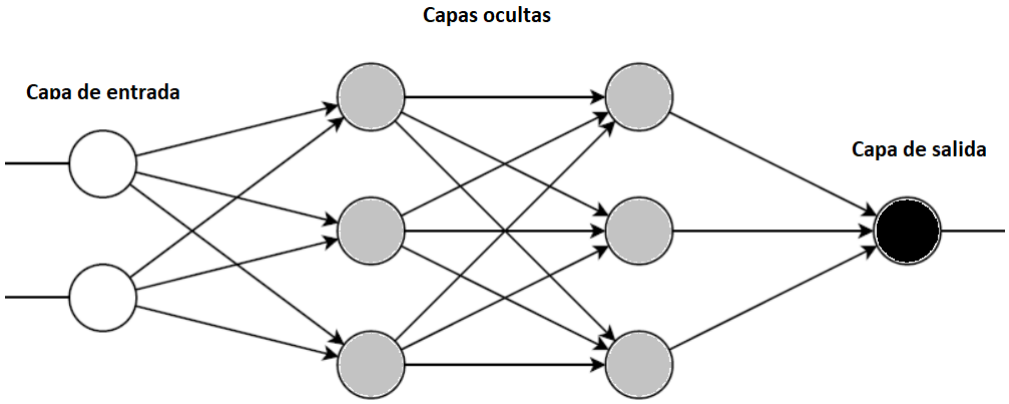


Figura 2-1. Diagrama de una red neuronal.

Estas redes están constituidas por capas formadas por una serie de nodos interconectados entre sí, denominados neuronas artificiales. En la figura 2-1 se muestra la estructura de una red neuronal básica. Las redes neuronales contienen una capa de entrada, una o varias capas ocultas y una capa de salida. En este ejemplo la red contendría dos variables de entrada, tres neuronas en cada una de las dos capas ocultas y una en la capa de salida.

2.1.1.1 Neuronas artificiales

Como acabamos de ver, la unidad básica de la que están dotados las redes neuronales son las *neuronas artificiales*. En la figura 2-2 puede apreciarse un diagrama de una neurona artificial.

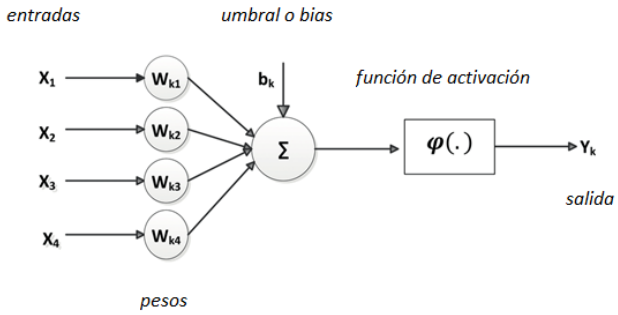


Figura 2-2. Diagrama de una neurona artificial. Adaptado de [9].

Una neurona consta básicamente de los siguientes elementos:

- **Entradas** (x_j): Reciben las salidas de otras neuronas o las entradas de la red neuronal.
- **Pesos** (w_{kj}): Determinan el efecto de las entradas en la neurona artificial.
- **Umbral** (b_k): Entrada externa a la neurona. Su valor es modificado durante el entrenamiento.
- **Función de activación** ($\varphi(\cdot)$): Determina la salida de la neurona aplicando una función matemática a la entrada efectiva. Son diferenciables y suelen ser no lineales. Algunas de las más empleadas son las siguientes:
    - Sigmoidea (σ): Es muy adecuada para crear las máscaras suaves necesarias en la separación y que se verán en una sección posterior. En la figura 2-5 a puede observarse su gráfica.

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Ecuación 2-1. Función de activación sigmoidea

    - Tangente hiperbólica (tanh): Es más común emplearla en zonas intermedias de la red que en los extremos. La figura 2-5 b muestra la gráfica de esta función.

$$\tanh(x) = \frac{e^{-x} - e^{-x}}{e^{-x} + e^{-x}}$$

Ecuación 2-2. Función de activación tangente hiperbólica

    - Unidad lineal rectificada (ReLU): A veces se emplea para generar máscaras y algunos sistemas la emplean para generar formas de onda [3]. Su gráfica se muestra en la figura 2-5 c.

$$f(x) = \max(0, x)$$

Ecuación 2-3. Función de activación ReLU

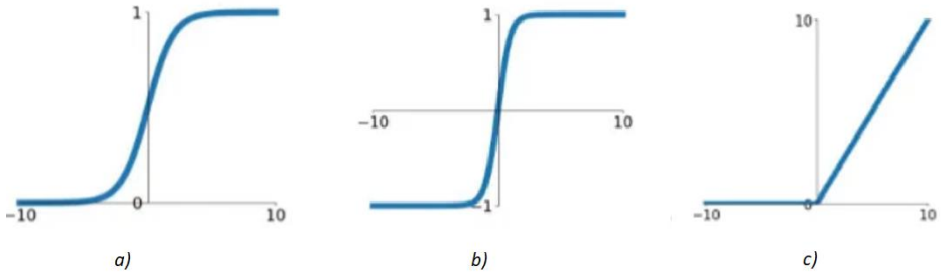


Figura 2-3. Funciones de activación.

- **Salida** (Y_k): Resultado de la transformación de la entrada efectiva a la neurona mediante la función de activación. Matemáticamente puede formularse del siguiente modo:

$$Y_k = \varphi(\sum_j w_{kj} x_j + b_k)$$

Ecuación 2-4. Salida de una neurona artificial.

2.1.1.2 Entrenamiento de la red neuronal

El entrenamiento de una red neuronal tiene básicamente como objetivo calcular unos valores para los pesos con los cuales, el error cometido al evaluar los ejemplos de entrenamiento sea lo más pequeño posible. Una vez establecidos estos valores, la red estará preparada para ser probada con otros ejemplos del conjunto de prueba con los que no ha sido entrenada.

El proceso de aprendizaje en el entrenamiento puede resumirse en cuatro pasos fundamentales:

1. **Prealimentación (Feedforward)**: Se alimenta a la red mediante un conjunto de datos de entrada y se calculan las salidas.
2. **Cálculo del error**: Se calcula la función de coste. Esta función proporciona una medida del error (cuánto difieren las salidas de la red neuronal de las salidas deseadas).
3. **Propagación hacia atrás del error (Backpropagation)**: Se propaga el error en la salida hacia las demás capas de la red neuronal, de forma que se haga posible el cálculo del gradiente de la función de coste respeto a pesos y umbrales de la red.
4. **Actualización de parámetros**: Se actualizan los pesos y umbrales de la red con el objetivo de disminuir el error en la salida. En este trabajo emplearemos el algoritmo de descenso de gradiente de mini lotes, dividiendo el conjunto de datos de entrenamiento en pequeños tamaños de lote (Batch Size).

Estos pasos se repiten múltiples veces. Una época es completada cuando todos los ejemplos del conjunto de entrenamientos han ingresado en la red. En el entrenamiento de la red neuronal tienen lugar varias épocas (hasta que el error converge a un valor deseado) [10].

El propósito de esto que la red sea capaz de generalizar, es decir, que se comporte bien ante entradas distintas a las usadas en el entrenamiento.

Un error que hay que evitar cuando se realiza el entrenamiento de una red neuronal es especializarla en exceso con respecto a los datos de entrenamiento. Esto puede dar lugar a problemas de *sobreajuste* (overfitting) en los que la red presenta un error de entrenamiento bajo pero un error de prueba alto y se produce porque el modelo memoriza el ruido o información irrelevante de los datos de entrenamiento, pero es incapaz de generalizar ante el conjunto de datos de prueba. La figura 2-6 muestra un ejemplo de sobreajuste.

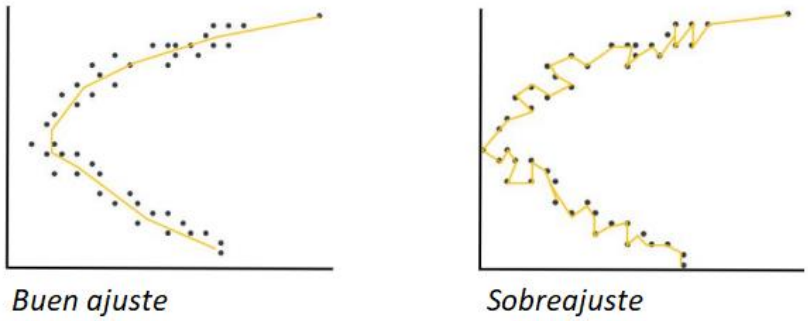


Figura 2-4. Efecto del sobreajuste

Para intentar mitigar estos problemas de sobreajuste se emplean diversas técnicas entre las que destacan las siguientes:

- **Aumento de los datos de entrenamiento.** Se aplican transformaciones sobre las entradas obteniendo muestras diferentes, pero iguales en esencia, lo que permite una mejor inferencia.
- **Empleo de parámetros compartidos.** Se permite la reducción del número de parámetros del modelo.
- **Parada temprana.** Se guardan los pesos en cada época del entrenamiento y se seleccionan aquellos para los que el error de validación es menor.
- **Normalización por lotes.** Se añade un paso adicional, habitualmente entre las neuronas y la función de activación, en el cual se logra obtener una salida normalizada. Cada salida de cada neurona de la red se normaliza de forma independiente, por lo que en cada iteración se calcula la media y varianza de cada salida para el mini lote en curso [11]
- **Dropout.** Se anula la salida de un determinado porcentaje de neuronas de una capa de forma aleatoria. En la figura 2-7 se observa un ejemplo del efecto de esta técnica.

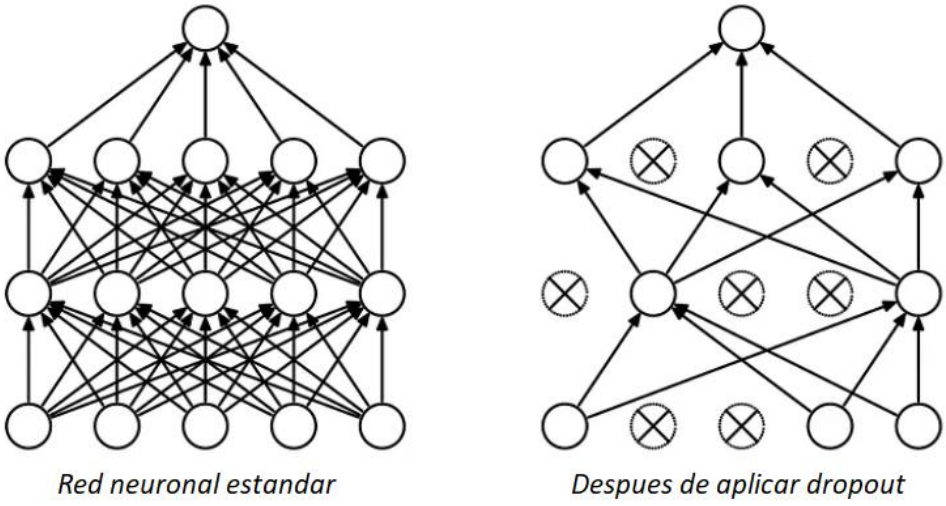


Figura 2-5. Funcionamiento del dropout. Adaptada de [12].

2.1.1.3 Redes neuronales convolucionales (CNN)

Las redes neuronales están compuestas principalmente por tres tipos de capas: capa convolucional, capa de agrupación y capa totalmente conectada.

Con cada capa, la red neuronal convolucional aumenta gradualmente en complejidad. Las primeras capas se centran en características simples y conforme avanzan los datos a través de las capas, la red va reconociendo elementos o formas más grandes hasta que finalmente identifica el objeto esperado.

La capa convolucional tiene un detector de características, (kernel o filtro), que se mueve por los campos receptivos de la imagen para comprobar si la característica está presente (convolución) [13]

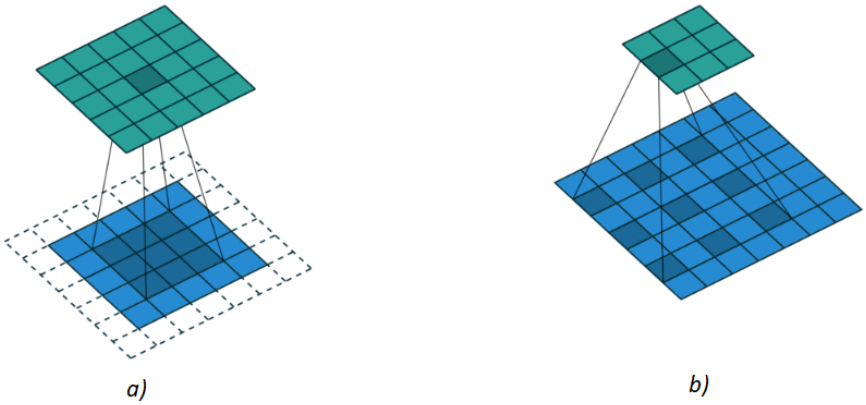


Figura 2-6. Ejemplo de dos convoluciones con entradas 2D [3].

Los parámetros que definen una capa convolucional son:

- **Tamaño del kernel**: Determina el número y la forma de los nodos que la capa actual ve de la anterior (campo receptivo de las neuronas).
- **Paso**: Establece la distancia del campo receptivo entre dos nodos adyacentes. Permite reducir la dimensión de las salidas.
- **Relleno**: Determina qué hacer en los bordes. Se pueden añadir ceros en los bordes de la entrada para conservar la dimensionalidad en las salidas.
- **Dilatación**: Dicta el espacio entre nodos de entrada que ve cada nodo convolucional [3], [10].

En la figura 2.6 se muestra un ejemplo de convoluciones con entradas 2D. En 2-6a, el kernel es 5x5 y relleno de 1. en 2-6b el kernel es 3x3 y un factor de dilatación 1. En ambas los cuadros azules son las entradas y los cian son las salidas.

La capa de agrupación permite reducir la dimensión reduciendo el número de parámetros de entrada. En ella se barre toda la entrada con un filtro, aplicando una función de agregación a los valores dentro del campo receptivo y llena así la matriz de salida. Hay dos tipos principales de agrupación:

- **Agrupación máxima (max pooling):** conforme el filtro recorre la entrada, selecciona el píxel con el valor más alto para enviarlo a la matriz de salida.
- **Agrupación media (average pooling):** a medida que el filtro avanza por la entrada, calcula el valor promedio dentro del campo receptivo para enviarlo a la matriz de salida.

En la figura 2-7 puede verse un ejemplo de los dos tipos de agrupación comentados anteriormente.

Esta capa permite reducir la complejidad, limita el riesgo de sobreajuste y mejora la eficiencia, aunque supone la pérdida de mucha información.

En la *capa totalmente conectada*, cada nodo de la capa de salida está conectado directamente a un nodo de la capa anterior. Esta capa realiza la tarea de clasificación basándose en las características extraídas de las capas anteriores y sus diferentes filtros [13].

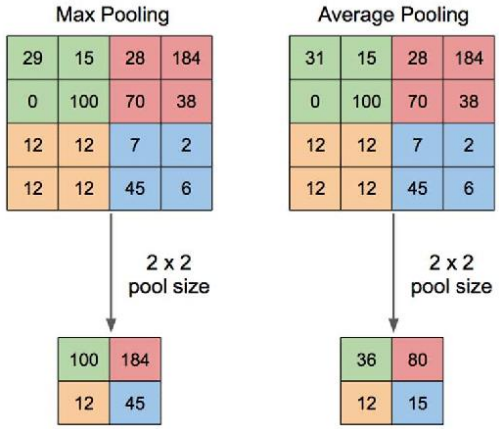


Figura 2-7. Ejemplo de agrupación máxima y agrupación media [14].

2.1.1.4 Redes neuronales recurrentes (RNN)

Una *red neuronal recurrente* (RNN) es un tipo de red neuronal que usa datos secuenciales o de series temporales. Estas redes se distinguen por obtener información de entradas anteriores para influir en la entrada y salida actuales. Así mismo, se caracterizan por compartir parámetros en cada capa de la red.

En la separación de fuentes, las capas recurrentes incorporan el audio a medida que cambia a lo largo del tiempo de manera que, si por ejemplo se ingresa un espectrograma en la capa recurrente, esta incorporará una columna de este cada vez (el espectro completo en ese paso de tiempo).

Las RNN usan el algoritmo de propagación hacia atrás a través del tiempo (BPTT) para el cálculo de los gradientes, el cual es algo diferente al de retropropagación tradicional al ser específico para la secuenciación de los datos. La principal diferencia es que el BPTT suma los errores en cada paso temporal [15].

Las capas recurrentes más empleadas en separación de fuentes son las de memoria de corto-largo plazo (LSTM). Estas están dotadas de celdas (figura 2-8) en las capas ocultas de la red neuronal que tienen tres puertas: una puerta de entrada, una de salida y una puerta del olvido. Estas puertas controlan el flujo de información necesario para pronosticar la salida en la red.

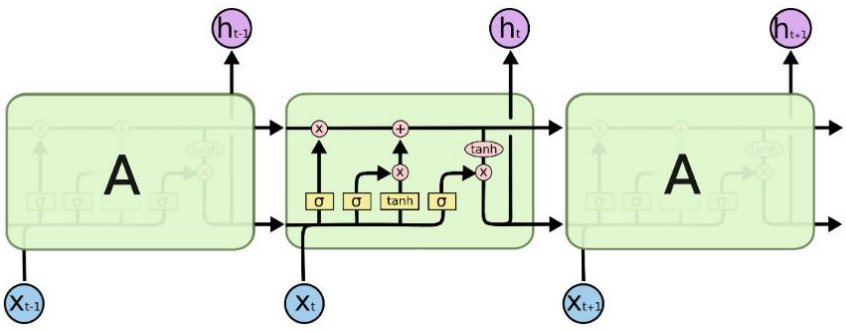


Figura 2-8. Implementación de modulo LSTM [16].

2.2 Representación de las señales de audio

2.2.1 Formas de onda

Las formas de onda son una forma de representar el sonido, en la cual, como puede apreciarse en la Figura 2-9, el eje de abscisas corresponde al tiempo mientras que el eje de ordenadas corresponde a la amplitud. Se obtienen tras el registro, mediante un instrumento (normalmente un micrófono), de los cambios de presión del aire durante un periodo de tiempo y la posterior conversión en una señal eléctrica que se muestrea en intervalos de tiempo y se cuantifica.

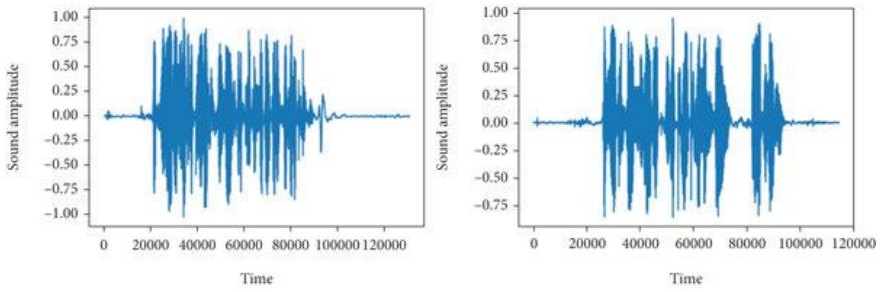


Figura 2-9. Ejemplo de forma de onda para la voz humana masculina [17].

Aunque es cierto que en la forma de onda se encuentra toda la información de la señal y algunos enfoques de separación operan directamente sobre ella, su interpretación directa es compleja, por lo que frecuentemente se opta por realizar una transformación de la señal que proporcione información útil. Así pues, se pasa al dominio de tiempo-frecuencia.

2.2.2 Representación en tiempo-frecuencia

Una representación en tiempo-frecuencia puede definirse como una matriz bidimensional la cual describe el contenido frecuencial de una señal de audio a lo largo del tiempo. Esta representación es el tipo más común en los distintos enfoques de separación de fuentes.

En particular, la representación tiempo-frecuencia denominada *Transformada de Fourier de tiempo corto* es muy empleada y sus fundamentos se describen a continuación.

2.2.2.1 Transformada de Fourier de tiempo corto (STFT)

La Transformada en tiempo corto de Fourier (STFT) se calcula a partir de la obtención de la Transformada Discreta de Fourier (DFT) en ventanas cortas $w[n]$ de un numero de muestras N desplazadas un tamaño de salto de H muestras en el tiempo. Matemáticamente, la aplicación de estas operaciones sobre una señal $x[n]$ puede ser expresarse del siguiente modo:

$$X[t,k] = \sum_{n=0}^{N-1} w[n]x[tH+n]e^{-j\frac{2\pi kn}{N}}$$

Ecuación 2-5. Cálculo de la STFT

Cabe destacar que los coeficientes de la STFT resultante, constituyen una matriz con valores complejos, o lo que es lo mismo, poseen componentes de magnitud y de fase. Ambas componentes serán necesarias para su reconversión en forma de onda.

A partir de las componentes de magnitud se puede obtener una representación gráfica denominada *espectrograma* que muestra el tiempo en el eje de abscisas, frecuencia en el de ordenadas y el valor absoluto de la STFT en cada punto de tiempo-frecuencia en un tercer eje empleando habitualmente una escala de colores o escala de grises. En la Figura 2-10 puede observarse un ejemplo del espectrograma que acabamos de comentar.

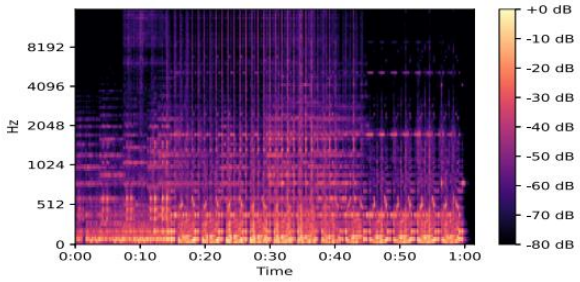


Figura 2-10. Ejemplo de espectrograma.

Dos de los aspectos importantes en el cálculo la STFT son el *enventanado* y el *tamaño de salto*.

Por un lado, conviene citar que el empleo de ventanas distorsiona el espectro de la señal bien produciendo picos en frecuencias incorrectas (fuga) debido a los lóbulos secundarios del espectro de la ventana o bien produciendo pérdida de resolución en frecuencia (manchado) a causa del ancho del lóbulo principal de la ventana.

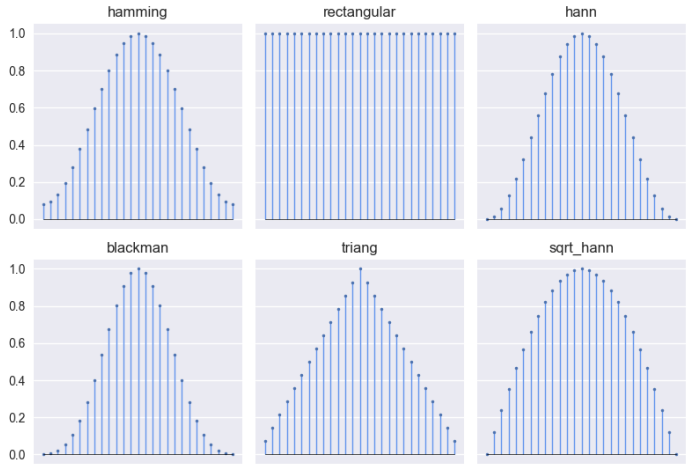


Figura 2-11. Tipos de ventanas más comunes en la separación de fuentes [3].

Con el fin de disminuir estas distorsiones espectrales, se han diseñado numerosos tipos de ventanas cada uno de las cuales ofrece un rendimiento distinto frente a diferentes escenarios. En la Figura 2-11 se muestran seis tipos de ventana comunes en el cálculo de la DTFT para la separación de fuentes.

Por otro lado, es importante la elección óptima del tamaño de la ventana, ya que determina cuántas muestras se incluyen, así como la resolución del eje de frecuencia de la STFT. A mayor tamaño (menor resolución en tiempo), mayor será la resolución en frecuencia y viceversa.

El tamaño de salto, por su parte, es un parámetro que indica la distancia en muestras que hay entre dos ventanas adyacentes, pudiéndose así acortar o alargar eje del tiempo en una STFT.

Conviene a su vez señalar la existencia de una técnica denominada *solapamiento-suma* (overlap-add), la cual es empleada en la etapa de síntesis, para recuperar la señal en tiempo que se basan en un conjunto de parámetros llamados COLA (Constant Overlap-Add) los cuales suman un valor constante al aplicar ventanas sucesivas [3] . En la figura 2-12 se muestra el proceso por el cual se obtiene la STFT.

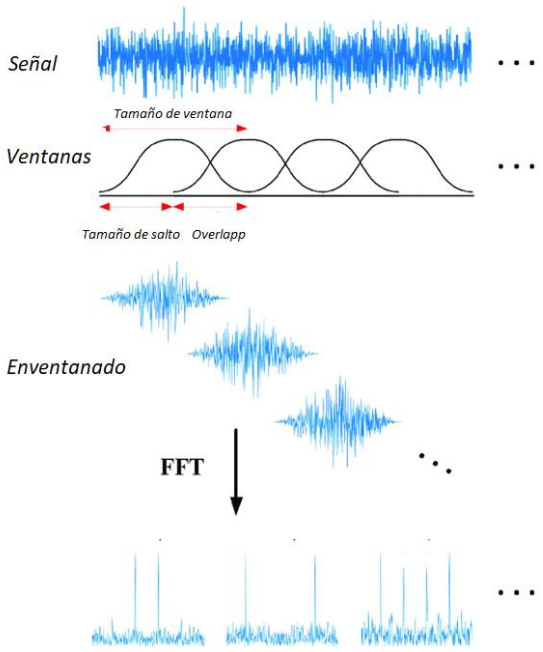


Figura 2-12. Proceso de obtención de la STFT [18].

Para finalizar este apartado en el que hemos puesto el foco en la STFT, cabe mencionar la existencia de algunas representaciones tales como la CQT [19], la CFT [20] o la MCFT [21], entre otras, que también han sido también usadas en enfoques de separación de fuentes.

2.3 Máscaras tiempo-frecuencia

La separación de fuentes musicales requiere la utilización de un filtro variante en el tiempo. Es por ello por lo que las *máscaras tiempo-frecuencia* sean un recurso ampliamente usado en los sistemas de separación modernos como los que nos ocupan.

Como podemos ver en la figura 2-13 estas máscaras (matrices que contienen valores en el intervalo [0.0,1.0]) realizan su tarea de filtrado mediante la multiplicación elemento a elemento con el espectrograma a filtrar como paso previo a la inversión de este y recuperación de la señal de tiempo.

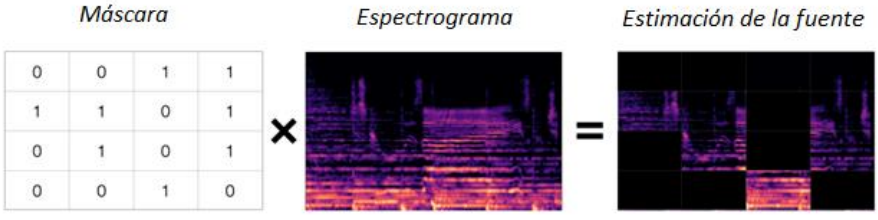


Figura 2-13. Aplicación de una máscara tiempo-frecuencia a un espectrograma. Adaptada de [3]

Algunas de las máscaras tiempo-frecuencia empleadas en el ámbito de la separación de fuentes sonoras son:

- **Máscara binaria**. Solo toman valores 1, cuando se asume que la señal a separar está dominada por una fuente, y 0 en caso contrario. Si bien su uso se ha reducido en estimaciones finales de fuentes, sigue siendo válida en tareas de entrenamiento.
- **Máscara suave**. Pueden tomar cualquier valor dentro del intervalo [0.0,1.0], lo que las hace más flexibles que las anteriores y por lo general conducen a resultados con menor distorsión [22].

El cálculo de las máscaras requiere una estimación de las fuentes individuales en el dominio de tiempo-frecuencia. Esta tarea puede llevarse a cabo mediante el uso de una red neuronal que las prediga [10].

2.4 Medidas de evaluación objetiva

Con el fin de evaluar objetivamente los resultados de un modelo de separación de fuentes, se han desarrollado medidas que se basan en la descomposición de la fuente estimada $\hat{s}_j$ en varias señales[23]: s_{target} (versión de la fuente original afectada por una distorsión permitida), e_{interf} (interferencia procedente de fuentes no deseadas), e_{noise} (ruido procedente de los sensores empleados para la captura de la mezcla) y e_{artif} (artefactos propios del algoritmo de separación).

$$\hat{s}_j = s_{target} + e_{interf} + e_{noise} + e_{artif}$$

Ecuación 2-6. Descomposición de la fuente estimada para su evaluación.

A continuación, se presentan unas métricas ampliamente usadas en la evaluación objetiva de la separación basadas en descomposición que acabamos de comentar:

- **Relación señal a distorsión (SDR):**

$$SDR = 10\log_{10}\frac{\left\|s_{target}\right\|^2}{\left\|e_{interf} + e_{noise} + e_{artif}\right\|^2}$$

Ecuación 2-7. Cálculo de la relación señal a distorsión.

- **Relación señal a interferencia (SIR):**

$$SIR = 10\log_{10}\frac{\left\|s_{target}\right\|^2}{\left\|e_{interf}\right\|^2}$$

Ecuación 2-8. Cálculo de la relación señal a interferencia.

- **Relación señal a artefactos (SAR):**

$$SAR = 10\log_{10}\frac{\left\|s_{target} + e_{interf} + e_{noise}\right\|^2}{\left\|e_{artif}\right\|^2}$$

Ecuación 2-9. Cálculo de la relación señal a artefactos.

3 ESTADO DEL ARTE

La separación de fuentes musicales es una tarea que ha atraído a multitud de investigadores a lo largo de las cinco últimas décadas. Esto es en parte debido a que existen diferentes formas de abordar el problema en cuestión. A lo largo de este capítulo se enumerarán los distintos enfoques que se ha propuesto a lo largo del tiempo para resolver esta tarea de separación.

Los métodos de separación de fuentes pueden dividirse en dos categorías principales: una que usa modelos basados en los espectrogramas y otra que emplea modelos basados en la forma de onda.

Los modelos que trabajan en representaciones de tiempo-frecuencia predicen un espectrograma de potencia para cada una de las fuentes y reutilizan la fase de la mezcla de entrada para sintetizar las formas de onda.

A principios de este siglo, las técnicas de factorización no negativa de matrices [24] (NMF) se empezaron a aplicar con éxito a la separación de fuentes musicales. Estas técnicas se basan en modelar el espectro de potencia como una suma ponderada de un diccionario espectral, cuyos elementos son agrupados en fuentes individuales. A estas técnicas han contado con distintas mejoras tales como su extensión a números complejos [25], permitiendo modelar fase y magnitud de forma simultánea.

A su vez, otro de los métodos propuestos para abordar el problema de la separación ha sido el análisis de componentes independientes, basado en supuestos de independencia y la existencia de múltiples micrófonos.

El avance en los últimos años de las técnicas de aprendizaje profundo ha permitido mejorar los enfoques anteriores.

En 2014 se realizó un primer trabajo para la separación del habla [26]. A este le siguieron trabajos sobre música empleando redes completamente conectadas [27][28] así como redes convolucionales y recurrentes [29] [30].

En 2016, las técnicas basadas en espectrogramas comienzan a usar filtrado de Wiener, una vez demostrada su eficiencia [31], siendo usado en la actualidad por los modelos de mejor rendimiento.

En 2019 se publica *Open Unmix* [32], un modelo reproducible, entrenado sobre la base de datos *MusDB* [33] que presenta una gran base para la investigación de esta temática.

Un año más tarde aparece *D3Net* [34] mejorando la tecnología en el dominio del espectrograma. Este modelo usa convoluciones dilatadas con conexión densa. También en 2020 se presenta otra arquitectura llamada *Spleeter* [35] que ha sido ampliamente adoptada en la industria musical debido a su gran rendimiento.

Por otra parte, recientemente se han desarrollado modelos que operan sobre la forma de onda en lugar de hacerlo sobre el espectrograma.

En primer lugar, aparece *Wave-U-Net* [36], un modelo que fue adaptado al dominio de la forma a partir de otro inicial basado en el espectrograma. Si bien este modelo supuso un punto de partida, su rendimiento está muy alejado de los modelos basados en espectrogramas que se han comentado anteriormente.

Posteriormente se presentan *Tasnet* [37] y *Conv-Tasnet* [38]. Este último es una versión refinada de *Tasnet* que sustituye la LSTM propuesta inicialmente por una superposición de convoluciones dilatadas. Esto supuso una notable mejora en los resultados.

Por último, cabe destacar la aparición de *Demucs* [39], un modelo al que le siguieron sucesivas versiones y que continúa actualizándose a modelos más complejos y se ha situado como uno de los mejores a nivel de rendimiento en el marco de la separación de fuentes musicales.

4 METODOLOGÍA

Una vez realizada la introducción teórica acerca de la separación automática de fuentes musicales y presentado el estado de la técnica de esta, en este capítulo se pasará a describir los recursos, datasets y arquitecturas empleadas para cubrir nuestro objetivo final de análisis y diseño de un modelo de separación de fuentes musicales. Para ello, en un primer lugar se partirá del análisis de los modelos existentes DeepConvSep [29] y Open Unmix [32] y finalmente se realizarán una serie de modificaciones en la arquitectura de red de este último modelo.

4.1 Recursos utilizados

4.1.1 Hardware

Tal y como se ha comentado en la parte teórica de este trabajo, entrenar un modelo de Machine Learning es una tarea que puede requerir una gran capacidad de cálculo computacional. Si bien todos los modelos tratados en este trabajo pueden ser entrenados con la CPU, esta no resulta especialmente eficiente y prolonga notablemente el tiempo de entrenamiento. Es por ello por lo que hemos hecho uso de las GPUs, específicamente diseñadas para resolver simultáneamente un gran número de operaciones. Concretamente, se ha usado un sistema con las siguientes características:

Tabla 4-1. Especificaciones de la computadora usada.

CPU	I9-12900 H 2.5 GHz 12th Gen
RAM	32 GB
ALMACENAMIENTO	1 TB SSD
GPU	NVIDIA GeForce RTX 3070 8 GB
SISTEMA OPERATIVO	Linux Pop!_OS

4.1.2 Software

Para cada uno de los modelos tratados en este trabajo se ha hecho uso de diferentes versiones de Python y de distintas librerías.

Figura 4-1. Logo de python

Así pues, **DeepConvSep**, se desarrolla en *Python 2.7* y emplea *Theano* y *Lasagne* para el entrenamiento de las redes neuronales. Este modelo usa otras dependencias tales como *NumPy*, *SciPy* o *climate*.

Figura 4-2. Logos de Theano y Lasagne.

Por su parte, **Open-Unmix** y nuestra versión modificada del mismo están realizadas en *Python 3.7* y emplean la librería *PyTorch* en sus implementaciones además de diferentes librerías entre las que destacan *scikit-learn*, *NumPy*, *SciPy*, *tqdm*, *cudatoolkit* o *FFmpeg*.

Figura 4-3. Logos de PyTorch y NumPy.

A continuación, se describirán brevemente tanto el lenguaje de programación python y algunas de las librerías destacadas que se han comentado anteriormente:

- **Python**: Es un lenguaje de programación ampliamente utilizado en Machine Learning si bien su ámbito de uso es mucho más amplio (aplicaciones web, desarrollo de software, etc.). Es un lenguaje multiplataforma muy eficiente y fácil de aprender, interpretado, de alto nivel y orientado a objetos y entre sus numerosos beneficios destacan su fácil comprensión, al tener una sintaxis similar al inglés, su productividad o la gran cantidad de bibliotecas con las que cuenta. Además, existe una comunidad activa compuesta por millones de desarrolladores alrededor del mundo que ofrecen soporte ante los distintos problemas que pueden presentarse.
- **Theano**: Es una librería que permite definir, optimizar y evaluar eficientemente expresiones matemáticas de matrices de múltiples dimensiones.
- **Lasagne**: Es una librería para construir y entrenar redes neuronales en Theano.
- **PyTorch**: Es una librería de tensores optimizada para el aprendizaje profundo que cuenta con la capacidad de ejecutarse en GPUs, con la consiguiente aceleración del entrenamiento de los modelos.
- **NumPy**: Es una librería especializada en el cálculo numérico y el análisis de datos que incorpora una clase de objetos denominadas arrays que permite representar colecciones de datos de un mismo tipo en múltiples dimensiones y cuenta con funciones muy eficientes para su manipulación. Permite un procesamiento mucho mayor que las listas convencionales de Python lo que la hace muy útil para el procesamiento de matrices de múltiples dimensiones.
- **SciPy**: Es una librería que incluye módulos estadística, optimización, algebra lineal, transformadas de Fourier, procesamiento de señales, etc.
- **Scikit-Learn**: Es una librería para el aprendizaje automático que incluye versiones de muchos algoritmos de clasificación, regresión y análisis de grupos.

4.2 Dataset (MusDB18)

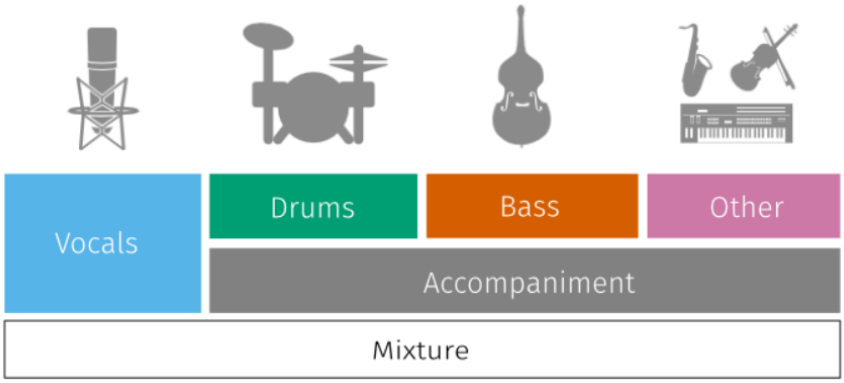


Figura 4-4. Esquema de la composición del dataset[40].

MusDB18 [33] es un conjunto de datos compuesto por mezclas estereofónicas de 150 canciones completas (alrededor de 10 horas de duración), que están acompañadas por sus respectivas pistas aisladas correspondientes a batería, bajo, voz y otros acompañamientos.

En cuanto a la distribución de géneros musicales, tal y como sucede con DSD100, los estilos musicales que predominan son y el pop y el rock tal y como podemos observar en la tabla 4-3.

Tabla 4-2. Distribución de géneros musicales del conjunto de datos MusDB18

GÉNERO MUSICAL	NÚMERO DE CANCIONES
Cantautor	14
Country	3
Electrónica	7
Heavy metal	12
Jazz	3
Pop/Rock	101
Rap/Reggae	10

MusDB18 consta de dos subconjuntos de datos. Uno de ellos está compuesto por 100 canciones y se usa para el entrenamiento del modelo y el otro se compone de 50 canciones y es el relativo a la prueba de este.

Cabe señalar que para nuestro trabajo hemos empleado el dataset MusDB18-HQ, idéntico al descrito anteriormente, tal y como se observa en la figura 4-5, pero que consta de archivos WAV sin comprimir.

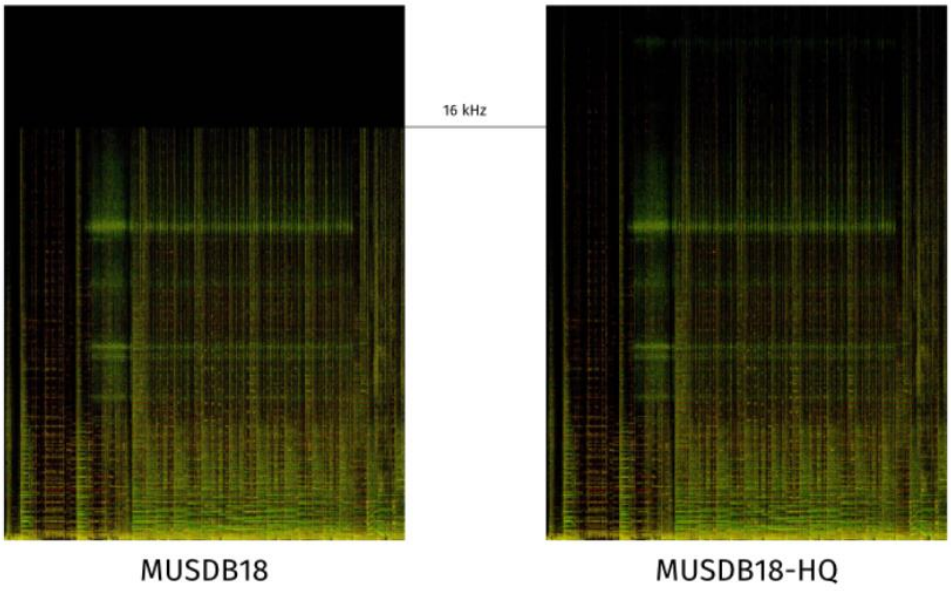


Figura 4-5. Comparativa de MusDB18 y MusDB18-HQ [33].

4.3 Modelos de separación de fuentes musicales empleados

4.3.1 Modelo DeepConvSep

Este modelo de separación automático de fuentes musicales fue propuesto por Pritish Chadna y Marius Miron en el año 2017 [29]. En la figura 4-6 se muestra el diagrama de bloques del sistema:

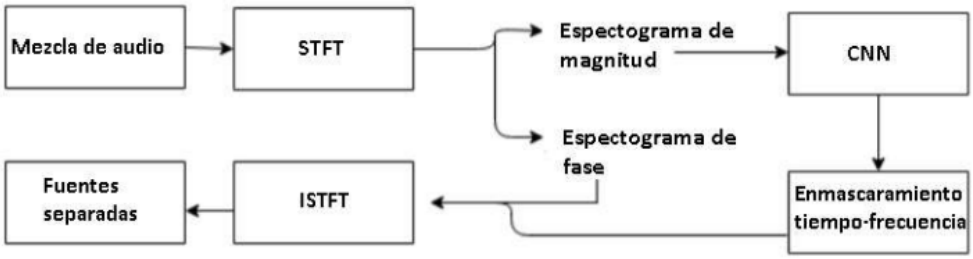


Figura 4-6. Diagrama de bloques del modelo DeepConvSep

La STFT se calcula para cada uno de los segmentos que componen la mezcla de audio. El espectrograma de magnitud resultante pasa a través de la red neuronal convolucional, generándose una estimación para cada una de las fuentes musicales separadas. Esta se usa para el cálculo de las máscaras suaves de tiempo-frecuencia aplicadas al espectrograma de magnitud de la mezcla para calcular las estimaciones de magnitud final para las fuentes separadas. Estas estimaciones, junto con la fase de la mezcla de audio, se emplean para obtener las señales de audio correspondientes a las fuentes musicales separadas.

En la figura 4-6 se puede observar la arquitectura de la red neuronal de este modelo, si bien solo se representan dos salidas, aunque el sistema está compuesto por cuatro salidas (batería, bajo, voz y otros instrumentos).

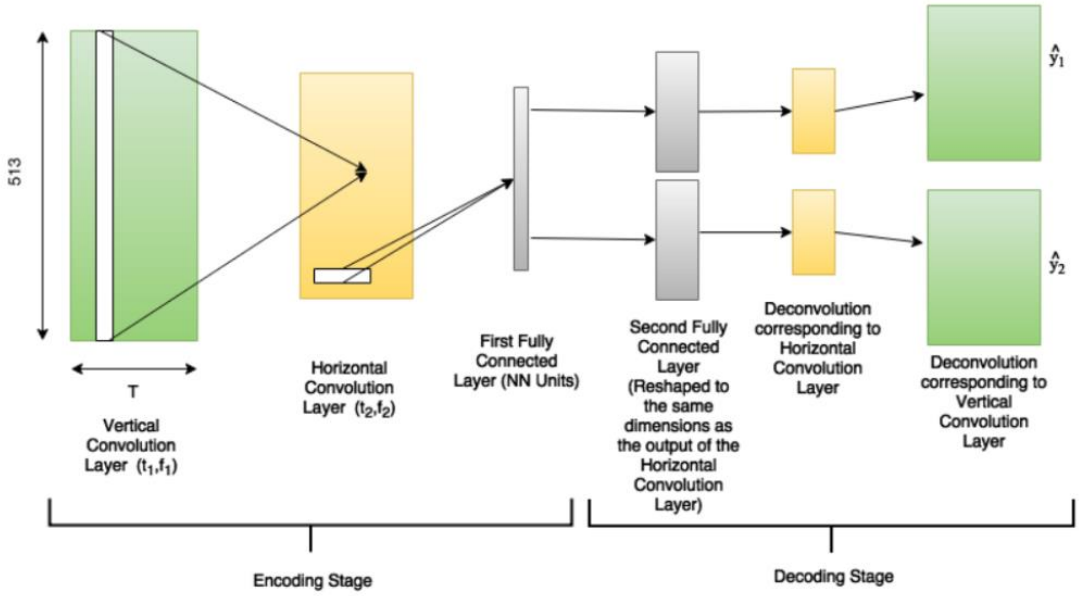


Figura 4-7. Arquitectura de la red neuronal del modelo DeepConvSep [29]

Como se observa en la figura 4-7, en el avance a través de la red neuronal tienen lugar una etapa de codificación y otra de decodificación. En ellas intervienen las siguientes capas:

- **Capa de Convolución Vertical**: En ella se lleva a cabo una convolución vertical que tiene por objetivo obtener una representación de las características de frecuencia de la entrada.
- **Capa de Convolución Horizontal**: Modela la evolución temporal para los diferentes instrumentos a partir de las características aprendidas en la capa anterior.
- **Primera Capa completamente conectada**: Esta capa que es la última de la fase de codificación, tiene como objetivo facilitar la obtención de características no lineales a partir de las entradas anteriores. Es una capa compartida por todas las salidas de las fuentes y actúa como cuello de botella de la red. Usa una función de activación ReLU.
- **Segunda Capa completamente conectada**: Actúa como filtro para determinar qué características son mejores para determinar las salidas para cada fuente.
- **Capa de Deconvolución Horizontal**: En ella se lleva a cabo el paso opuesto al que tiene lugar en la Capa de Convolución Horizontal.
- **Capa de Deconvolución Vertical**: Esta capa proporciona la salida final de la red que puede interpretarse como las máscaras de las fuentes.

4.3.2 Modelo Open-Unmix

El segundo modelo con el que hemos trabajado es Open-Unmix. Se trata de un software de código abierto que opera con los espectrogramas de audio. Se caracteriza por el empleo de capas LSTM bidireccionales en su red neuronal y entre sus principales ventajas destacan su facilidad de reproducción y ampliación, así como la rapidez en el aprendizaje de la red. En la figura 4-8 puede observarse la estructura de este modelo.

Tal y como sucede en el modelo anterior, Open-Unmix opera en el dominio de tiempo-frecuencia. En una primera etapa la entrada de audio se divide en segmentos garantizándose así que no quede comprometido el coste computacional. El espectrograma de entrada se recorta a 16 KHz y se normaliza

usando la media global y la desviación estándar. Además, se aplica la normalización por lotes en distintas etapas del modelo con lo que se reduce el riesgo de sobreajuste durante el entrenamiento.

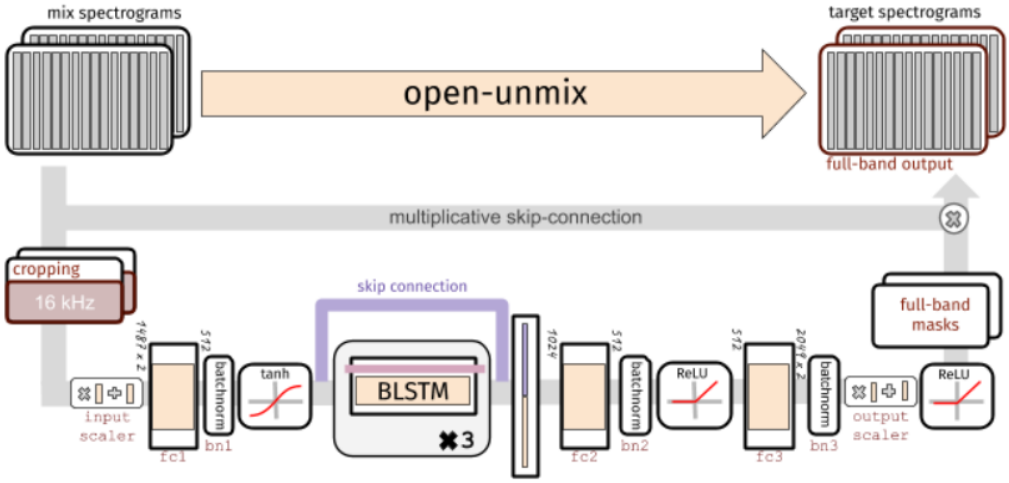


Figura 4-8. Estructura del modelo Open-Unmix [41].

Como se puede ver en la figura 4-8, el núcleo de este modelo está formado por una red LSTM bidireccional de tres capas. El modelo toma información de pasado y futuro simultáneamente por lo que, con esta configuración, no puede trabajar en tiempo real.

Dado que la LSTM no es capaz de funcionar con la resolución original del espectrograma de entrada, por lo que es necesaria una reducción de dimensionalidad, en la que la red aprende a comprimir el eje de frecuencias del modelo reduciéndose la redundancia del modelo y haciendo que el mismo converja con mayor rapidez. Para llevar a cabo esta reducción, se usan capas completamente conectadas.

Tras su paso por la red LSTM la señal se decodifica a su dimensionalidad original. La salida se multiplica por el espectrograma de entrada y finalmente la síntesis de la señal de salida se lleva a cabo mediante la STFT inversa. En la última etapa del modelo, para llevar a cabo la inferencia, la señal es procesada mediante la implementación de un filtro Wiener multicanal.

Además de lo descrito anteriormente, y como se puede observar en la figura 4-8, se emplean las funciones de activación ReLU, tanh y sigmoidea a lo largo del mismo cuyo funcionamiento y cometido fue descrito anteriormente en la introducción teórica de esta memoria.

4.3.3 Modelo propuesto

Una vez estudiados los distintos modelos de separación de fuentes musicales que acabamos de describir pasamos a proponer un modelo diseñado por nosotros que se inspira en la arquitectura de red neuronal de DeepConvSep y que se implementa haciendo uso del software de código abierto propuesto por Open-Unmix. Con ello pretendíamos poner a prueba el comportamiento de las redes convolucionales sobre un software más actualizado y novedoso, ya que uno de los principales inconvenientes que encontramos en primer modelo estudiado (DeepConvSep) es que estaba implementado sobre Python 2.7 el cual no cuenta con actualizaciones desde hace años. A su vez, se decidió implementar el sistema sobre el software del segundo de los modelos estudiados (Open-Unmix) debido a que este está diseñado para ser fácilmente modificable y ampliable y servir así de punto de partida a investigadores interesados en esta temática.

Como acabamos de comentar, la arquitectura está basada en el trabajo de Chadna et al [29].. Consiste en un codificador que se encarga de reducir la dimensionalidad de entrada seguido de un decodificador (en lugar de los cuatro propuesto en DeepConvSep, ya que la separación de cada instrumento se entrena por separado). En la figura 4-9 se está representada la arquitectura del modelo propuesto.

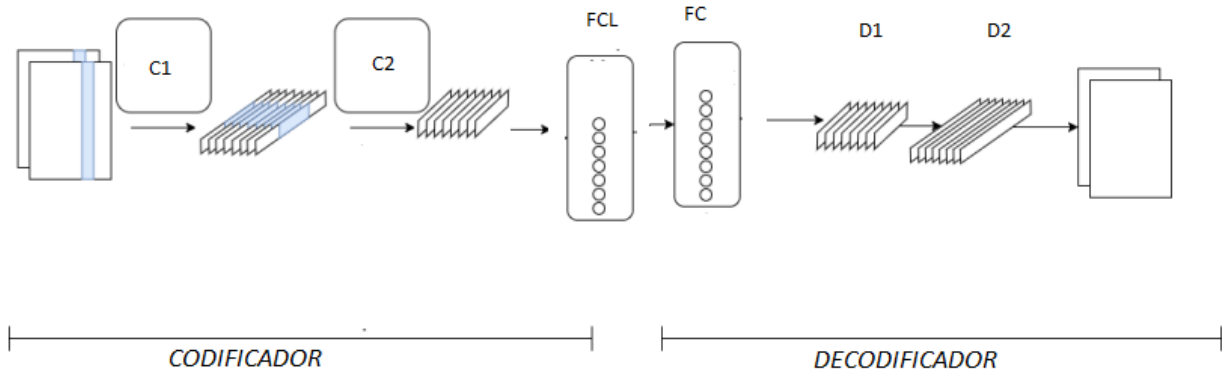


Figura 4-9. Arquitectura de la red neuronal del modelo propuesto.

Como se observa en la figura 4-9, el codificador está formado por dos capas convolucionales en serie. La primera capa (C1) realiza una convolución vertical y cuenta con 50 filtros de dimensión 2049x1. Con ella se pretende que la red capture patrones espectrales del espectrograma de la mezcla.

La segunda capa (C2) tiene 30 filtros de dimensión 1x15 y realiza una convolución horizontal con el fin de capturar la evolución temporal de los patrones espectrales aprendidos en la capa anterior.

La salida de esta segunda capa se introduce a una capa completamente conectada (FCL) de 512 neuronas, la cual forma un espacio latente representado las características relevantes.

A esta capa completamente conectada, le sigue un decodificador (recordamos que la separación de cada instrumento se realiza en entrenamientos distintos) que se encarga de transformar el espacio latente y obtener el espectrograma de la fuente que se desea separar. El decodificador consiste en una capa densa completamente conectada (FC) de 512 neuronas seguida de capas de convolución traspuesta (D1 y D2) en las que se lleva a cabo proceso de deconvolución.

En el Anexo A se incluye el código en Python para la implementación del este modelo.

5 ENTRENAMIENTO DE LOS MODELOS

En este capítulo se cubrirán los detalles relativos al entrenamiento de los modelos de redes neuronales analizados, así como del modelo propuesto. Entre otros aspectos, se describirá el ajuste realizado en los distintos parámetros e hiperparámetros que intervienen en el entrenamiento de los modelos con el objetivo de optimizar el rendimiento de estos.

5.1 Entrenamiento del modelo DeepConvSep

Para realizar el entrenamiento de este primer modelo se partió de la configuración inicial que se propone en los experimentos realizados previamente por los autores del modelo y que se recogen en [29]. En la tabla 5-1 se recogen los valores de los parámetros más destacados:

Tabla 5-1. Configuración inicial de parámetros destacados en DeepConvSep

batch_size *(tamaño de lote)*	32
time_context *(contexto temporal en frames)*	25
hopSize *(tamaño de salto)*	256
nepoch *(número de épocas)*	30
window *(ventana)*	hanning
overlap *(solapamiento)*	25

Se realizaron múltiples entrenamientos del modelo modificando el valor de sus hiperparámetros de forma empírica y finalmente se adoptaron aquellos que ofrecían una pérdida de entrenamiento menor. La configuración que nos ofreció mejores resultados se muestra en la tabla 5-2:

Tabla 5-2. Valores obtenidos de forma empírica

batch_size *(tamaño de lote)*	32
time_context *(contexto temporal en frames)*	30
hopSize *(tamaño de salto)*	512
nepoch *(número de épocas)*	40
window *(ventana)*	blackmanharris
overlap *(solapamiento)*	25

En las figuras 5-1 y 5-2 pueden apreciarse capturas del proceso de entrenamiento de nuestro experimento más destacado. En ellas se recogen datos relevantes tales como las pérdidas que resultaron de gran utilidad en el ajuste de los parámetros comentados anteriormente.

```
I 2023-09-01 09:01:33 dataset:635 found 460 files
I 2023-09-01 09:01:33 dataset:657 iteration size 2130
I 2023-09-01 09:01:42 trainer:434   Maximum:           1.110977
I 2023-09-01 09:01:42 trainer:435   Mean:              0.004026
I 2023-09-01 09:01:42 trainer:436   Standard dev:              0.017521
I 2023-09-01 09:01:42 trainer:162 Building Autoencoder
I 2023-09-01 09:02:20 trainer:238 Training...
Epoch 1 of 40 took 174.704s
  training loss:                40.465877
  training loss for vocals:             10.791901
  training loss for bass:               12.987197
  training loss for drums:              12.843502
  Beta component:               0.702503
  Beta component for voice:             0.839561
  alpha component:              0.291929
Epoch 2 of 40 took 192.329s
  training loss:                30.197542
  training loss for vocals:             7.708328
  training loss for bass:               9.624864
  training loss for drums:              9.536612
  Beta component:               0.785026
  Beta component for voice:             0.912965
  alpha component:              0.323945
Epoch 3 of 40 took 178.319s
  training loss:                27.704651
  training loss for vocals:             7.106050
  training loss for bass:               9.134791
  training loss for drums:              8.804249
  Beta component:               0.812527
  Beta component for voice:             0.948452
  alpha component:              0.333802
```

Figura 5-1. Primeras épocas del entrenamiento de DeepConvSep

```
Epoch 37 of 40 took 180.300s
  training loss:                19.084666
  training loss for vocals:             5.023946
  training loss for bass:               7.744796
  training loss for drums:              6.790521
  Beta component:               0.864690
  Beta component for voice:             1.061582
  alpha component:              0.357131
Epoch 38 of 40 took 180.163s
  training loss:                19.062164
  training loss for vocals:             5.010578
  training loss for bass:               7.783140
  training loss for drums:              6.776769
  Beta component:               0.864696
  Beta component for voice:             1.062454
  alpha component:              0.357302
Epoch 39 of 40 took 180.001s
  training loss:                18.959518
  training loss for vocals:             4.997553
  training loss for bass:               7.741503
  training loss for drums:              6.762406
  Beta component:               0.864736
  Beta component for voice:             1.062206
  alpha component:              0.357428
Epoch 40 of 40 took 180.647s
  training loss:                18.970129
  training loss for vocals:             4.987737
  training loss for bass:               7.772266
  training loss for drums:              6.760574
  Beta component:               0.864414
  Beta component for voice:             1.062736
  alpha component:              0.357244
I 2023-09-01 11:02:54 trainer:287 Separating
```

Figura 5-2. Últimas épocas del entrenamiento de DeepConvSep

5.2 Entrenamiento del modelo Open-Unmix

En el entrenamiento tanto de este modelo caben resaltar algunas diferencias fundamentales con respecto al estudiado anteriormente. De entre ellas destaca que, mientras en el modelo anterior el entrenamiento era conjunto para todas las fuentes, en Open-Unmix se entrenan por separado.

En la tabla 5-3 se han recogido los parámetros e hiperparámetros que hemos considerado más relevantes en el proceso de entrenamiento de este modelo.

Tabla 5-3. Parámetros de entrenamiento más destacados de Open-Unmix

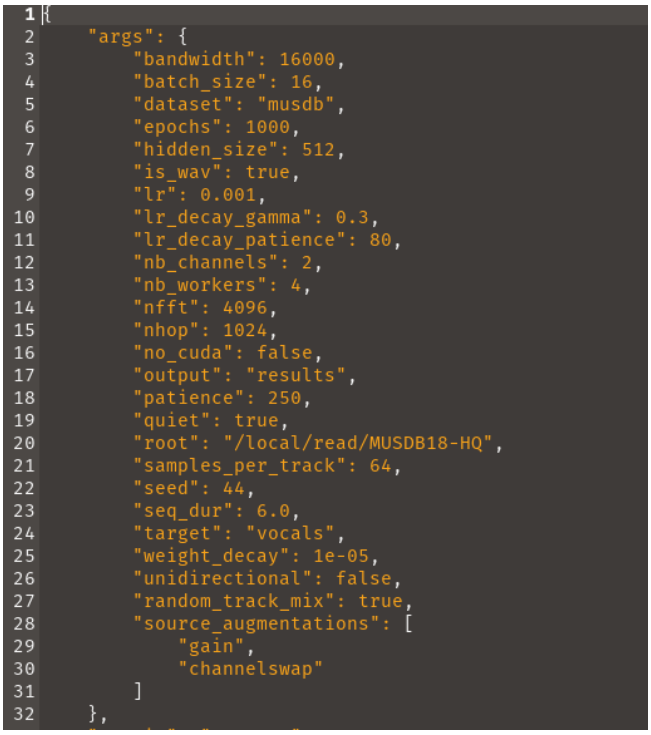
Argumento	Descripción
--batch-size <int>	*Tamaño del lote. Influye en el uso de la memoria y en el rendimiento de la capa LSTM.*
--seq-dur <int>	*Duración de la secuencia (s) de los segmentos tomados del conjunto de datos.*
--hidden-size <int>	*Parámetro del tamaño oculto de las capas densas.*
--lr <float>	*Tasa de aprendizaje (Learning rate)*
--weight_decay <float>	*Decaimiento del peso (regularización)*
--bandwidth <int>	*Ancho de banda máximo (Hz) procesado por la LSTM.*
--nfft <int>	*Tamaño de fft*

En el ajuste de los parámetros realizamos numerosos experimentos encaminados a obtener el mejor rendimiento posible.

Al entrenar el modelo con distintos tamaños de lotes observamos que, al aumentar gradualmente el valor de estos, se acelera el proceso de entrenamiento al reducirse el tiempo de cada época. Sin embargo, al emplear valores demasiado altos se producen diferencias notables entre la pérdida de entrenamiento y la pérdida de prueba.

Por otro lado, incrementar el decaimiento de peso (weight_decay) hace que se reduzca la diferencia entre la pérdida de entrenamiento y la de prueba dentro de unos márgenes. Valores demasiado altos de este parámetro reducen drásticamente la SDR obtenida.

Cabe destacar que todos los detalles de los entrenamientos realizados quedan registrados en archivos JSON. En la figura 5-3 se muestra un extracto del contenido de estos archivos que resultan de gran utilidad, puesto que además de los detalles de la configuración que pueden observarse, contienen un registro de las pérdidas de entrenamiento y validación de cada una de las épocas entrenadas.



```
{
    "args": {
        "bandwidth": 16000,
        "batch_size": 16,
        "dataset": "musdb",
        "epochs": 1000,
        "hidden_size": 512,
        "is_wav": true,
        "lr": 0.001,
        "lr_decay_gamma": 0.3,
        "lr_decay_patience": 80,
        "nb_channels": 2,
        "nb_workers": 4,
        "nfft": 4096,
        "nhop": 1024,
        "no_cuda": false,
        "output": "results",
        "patience": 250,
        "quiet": true,
        "root": "/local/read/MUSDB18-HQ",
        "samples_per_track": 64,
        "seed": 44,
        "seq_dur": 6.0,
        "target": "vocals",
        "weight_decay": 1e-05,
        "unidirectional": false,
        "random_track_mix": true,
        "source_augmentations": [
            "gain",
            "channelswap"
        ]
    },
```


Figura 5-3. Captura de JSON que contiene los detalles de entrenamiento

La tabla 5-4 recoge los valores finales que determinamos para los parámetros comentados anteriormente en el entrenamiento de nuestro modelo.

Tabla 5-4. Valores de parámetros destacados en Open-Unmix.

--batch-size	16
--seq-dur	6.0
--hidden-size	512
--lr	0.001
--weight_decay	0.00001
--bandwidth	16000
--nfft	4096

También merece especial mención la posibilidad de hacer uso de checkpoints, lo que nos permitió entrenar los modelos durante un gran número de épocas (hasta que estos dejaban de mejorar). En la tabla 5-5 se recogen el número de épocas con la que se entrenaron las distintas fuentes en los casos más exitosos.

Tabla 5-5. Número de épocas entrenadas en Open-Unmix

Instrumento	Número de épocas
Voz	450
Batería	210
Bajo	475

5.3 Entrenamiento del modelo propuesto

El proceso para realizar el entrenamiento sobre nuestro modelo ha sido muy similar al descrito para el modelo anterior ya que, como hemos comentado anteriormente, el modelo que proponemos tiene su base en él. En la figura 5-4 se muestra una captura realizada durante el entrenamiento en la que pueden observarse datos interesantes tales como la duración de las épocas (en torno a los 3 minutos), tiempos medios de iteraciones dentro de una época (en torno a 1 segundo) y las sucesivas pérdidas. Este último valor es de gran relevancia puesto que nos permite identificar en qué momento deja de mejorar el modelo.

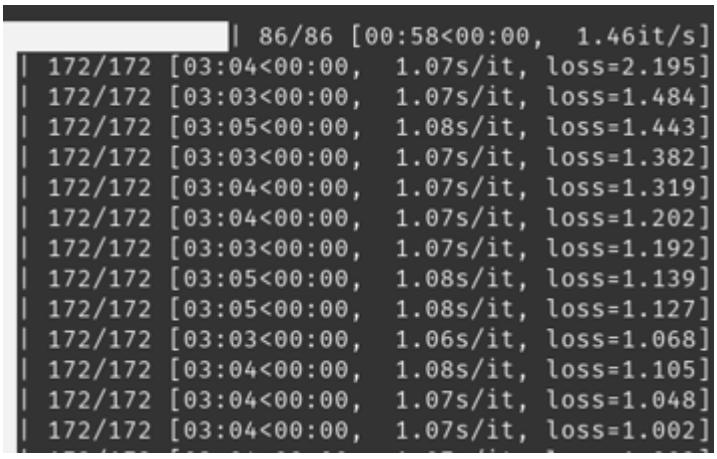


Figura 5-4. Cáptura de la consola de comandos durante el entrenamiento del modelo propuesto.

Hay que destacar que modelo propuesto se entrenó durante menos tiempo que Open-Unmix ya que la pérdida dejaba de reducirse en épocas más tempranas que el modelo anterior. En la tabla 5-6 se muestra el número de épocas entrenadas por el modelo para cada fuente.

Tabla 5-6.Número de épocas entrenadas en el modelo propuesto

Instrumento	Número de épocas
Voz	160
Batería	75
Bajo	100

Para finalizar, en la tabla 5-7 puede verse el ajuste final de los parámetros, que como hemos dicho, es muy similar al del modelo anterior.

Tabla 5-7.Valores de parámetros destacados en el modelo propuesto.

--batch-size	32
--seq-dur	6.0
--hidden-size	512
--lr	0.001
--weight_decay	0.00001
--bandwidth	16000
--nfft	4096

Conviene recalcar que los tres modelos se han entrenado con múltiples configuraciones distintas pero los valores recogidos en las tablas anteriores son los que nos han ofrecido mejores resultados. Estos resultados se presentarán a continuación en el capítulo siguiente.

6 RESULTADOS

Una vez entrenados realizado el entrenamiento de los distintos modelos, realizamos una evaluación objetiva de los resultados de separación de estos. A lo largo de este capítulo se presentan los distintos resultados obtenidos basados en las medidas que se describieron en la sección 2.4 de esta memoria.

6.1 Evaluación del modelo DeepConvSep

En primer lugar, tras el entrenamiento del algoritmo DeepConvSep, obtuvimos unos resultados que subjetivamente se alejaban de nuestras expectativas ya que, si bien en la separación de un instrumento en concreto se percibía la atenuación del resto de los instrumentos, estos permanecían siendo notablemente audibles. Esto podría ser debido a un ajuste inadecuado de los hiperparámetros, aunque lo cierto es que al realizar la evaluación objetiva los resultados no diferían en exceso a los que se presentan en [29] y cuyos resultados completos pueden observarse en la tabla 6-3.

Tal y como se observa en la tabla 6-1, este modelo presenta un mejor desempeño en la separación de la batería respecto al resto de las fuentes, lo que podría ser debido a la forma que tienen los filtros en la primera de las capas convolucionales haciéndolos eficientes en el modelado de señales percusivas.

Tabla 6-1. Mediana de las medidas de evaluación objetiva en dB.

	SDR	**SIR**	**SAR**
Voz	2.35	2.69	6.49
Bajo	2.32	1.99	6.13
Batería	2.72	7.74	5.92

En la tabla 6-2 se muestran las medidas SDR para cada uno de los géneros musicales.

Tabla 6-2. Valores de SDR por géneros musicales (modelo DeepConvSep).

	Pop/Rock	**Reggae/Rap**	**Electrónica**	**Heavy Metal**
Voz	2.94	2.88	-11.39	2.26
Bajo	2.59	1.59	3.27	0.17
Batería	2.73	2.63	2.62	5.31

Tabla 6-3. Valores de los resultados de la evaluación objetiva del modelo DeepConvSep.

ID canción	SDR			SIR			SAR		
	VOZ	BATERÍA	BAJO	VOZ	BATERÍA	BAJO	VOZ	BATERÍA	BAJO
1	4.03	1.65	3.40	6.71	1.37	3.79	8.60	3.40	7.47
2	5.74	3.40	-9.53	11.25	3.55	-7.59	9.25	10.24	5.50
3	3.26	3.31	2.58	7.25	5.73	1.81	8.71	6.33	7.07
4	1.33	2.63	3.28	3.18	5.36	2.40	5.39	4.30	9.31
5	2.05	3.01	2.80	2.55	4.50	2.49	5.99	6.17	6.70
6	-4.87	6.83	3.92	-6.25	11.41	6.12	3.25	9.22	6.50
7	4.83	2.33	3.80	9.58	3.39	4.36	5.88	4.34	7.62
8	3.51	2.03	2.21	5.71	3.64	1.32	6.41	4.19	8.92
9	2.82	5.68	3.40	4.14	10.39	3.79	6.78	8.42	6.20
10	3.26	-0.66	-6.72	4.59	0.89	-8.01	9.02	4.63	5.53
11	4.41	6.81	1.78	6.97	10.48	0.77	9.52	9.37	4.51
12	3.15	3.65	0.17	7.17	6.31	-2.69	4.47	8.10	2.13
13	4.53	5.43	3.05	6.39	8.54	2.82	8.63	7.28	6.52
14	1.38	4.03	0.68	3.09	5.12	-0.25	3.06	7.56	3.71
15	1.45	3.06	2.38	1.75	7.27	3.63	5.60	5.76	5.37
16	0.82	2.69	1.05	0.59	3.23	-0.06	4.66	4.58	5.07
17	-1.77	0.99	3.13	3.04	-0.92	3.28	5.67	6.07	7.10
18	2.43	2.05	0.17	3.40	3.51	-0.59	7.04	4.89	5.06
19	1.94	8.45	0.07	4.71	12.18	-2.98	6.33	11.57	6.12
20	4.36	1.79	3.64	10.63	14.86	6.40	10.09	11.03	4.96
21	2.19	2.74	1.59	3.25	6.09	0.98	4.98	6.51	5.85
22	6.42	-4.66	0.68	11.23	-1.64	-0.47	9.97	3.74	6.26
23	2.08	2.76	2.72	4.22	4.43	-1.05	8.57	5.13	6.58
24	3.57	5.95	2.25	4.29	9.29	1.75	9.81	9.28	5.02
25	1.08	0.44	2.96	0.55	-2.25	2.09	10.02	3.64	8.25
26	3.13	4.67	1.42	4.49	9.02	1.13	6.09	7.95	3.37
27	-0.30	2.59	2.59	0.91	4.98	2.68	8.86	6.25	6.14
28	-37.62	7.24	3.41	-36.12	11.93	3.40	-0.32	9.76	-0.61
29	-21.17	4.13	0.57	-28.73	10.47	18.64	0.01	8.51	1.72
30	3.53	4.67	4.12	6.72	5.68	10.43	5.43	9.61	4.44
31	1.84	1.64	2.70	2.83	1.49	1.30	5.39	6.59	7.43
32	2.35	-0.39	2.99	3.23	-1.35	3.89	7.29	3.68	5.59
33	1.68	1.24	0.22	1.62	0.64	-1.96	7.19	3.34	7.75
34	2.74	1.96	4.59	3.71	2.63	5.15	6.58	3.84	9.65
35	6.01	8.73	3.26	13.11	14.63	5.52	10.62	11.76	4.18
36	5.28	1.10	0.35	8.38	1.35	-1.26	8.99	4.21	7.18
37	-2.95	-0.31	-1.88	-1.70	-1.01	0.36	1.88	2.27	1.67
38	-0.18	1.10	4.29	-1.39	0.73	5.76	6.24	3.78	6.69
39	3.11	1.09	0.97	3.67	0.76	3.31	6.78	3.27	5.45
40	3.77	1.15	2.26	4.50	2.34	3.03	6.25	1.71	5.88
41	1.31	4.29	-2.23	1.68	5.23	-5.91	7.99	10.21	6.84
42	1.40	1.50	-1.24	1.62	2.15	-2.92	3.78	1.80	4.06
43	-1.79	1.18	6.33	0.24	0.92	8.67	9.26	4.77	8.93
44	2.22	1.86	3.13	3.23	2.24	1.99	5.97	4.22	6.52
45	6.21	4.37	3.58	8.45	8.24	4.53	10.11	5.60	6.59
46	1.17	0.49	-4.55	0.97	-2.58	-8.07	1.55	5.46	11.75
47	2.26	5.31	1.57	3.44	9.99	0.96	4.57	6.99	4.37
48	4.79	2.80	7.14	7.85	5.03	10.99	9.68	3.61	9.27
49	1.13	6.01	-1.89	-2.21	7.55	-6.23	3.55	11.72	6.00
50	2.94	3.95	-0.75	3.40	6.82	-2.88	8.28	6.28	5.63

6.2 Evaluación del modelo Open-Unmix

En la tabla 6-6 se muestran los resultados completos de evaluación objetiva del modelo que entrenamos de Open-Unmix, realizada sobre el conjunto de prueba MusDB-HQ. La tabla 6-4 recoge los valores de la mediana para cada una de las medidas de evaluación.

Tabla 6-4. Medidas de evaluación objetiva en dB obtenidas en el modelo Open-Unmix.

	SDR	**SIR**	**SAR**
Voz	5.98	14.81	4.01
Batería	5.45	14.40	4.65
Bajo	4.92	16.52	2.24

En vista a los resultados obtenidos, nuestro modelo ofrece unos resultados muy similares a los que obtuvieron sus autores, por lo que creemos que el fine-tunning de este modelo, descrito en la sección 5.2, ha resultado óptimo. En la tabla 6-5 se muestran las medidas SDR para cada uno de los géneros musicales obtenidas en la evaluación de este modelo.

Tabla 6-5. Valores de SDR por géneros musicales (modelo Open-Unmix).

	Pop/Rock	**Reggae/Rap**	**Electrónica**	**Heavy Metal**
Voz	6.06	6.48	-3.78	5.6
Bajo	5.32	4.24	0.9	4.28
Batería	5.41	5.5	4.87	6.53

Open-Unmix representa una gran opción como sistema de separación, aunque en la actualidad sus resultados se han visto superados por otros modelos como se puede ver en la tabla 6-6.

Tabla 6-6. Algoritmos con mejores resultados en Musdb18-HQ [42].

Modelo	**SDR (medio)**	**SDR (bajo)**	**SDR (batería)**	**SDR (vocales)**
Sparse HT Demucs	9.20	10.47	10.83	9.37
Hybrid Transformer Demucs	9.00	10.39	10.08	9.20
Band-Split RNN	8.97	8.16	10.15	10.47
TFC-TDF-Unet	8.34	8.45	8.44	9.59
KUIELab-MDX-Net	7.47	7.83	7.20	8.97
CWS-PResUNet	6.77	5.93	6.38	8.92

Tabla 6-7. Valores de los resultados de la evaluación objetiva del modelo Open-Unmix.

ID canción	SDR			SIR			SAR		
	VOZ	BATERÍA	BAJO	VOZ	BATERÍA	BAJO	VOZ	BATERÍA	BAJO
1	7.20	4.51	-3.64	16.73	11.06	6.28	7.90	5.23	-3.19
2	6.41	3.44	5.52	10.85	4.08	16.36	5.17	1.18	2.30
3	8.33	5.32	4.46	10.53	9.01	20.41	6.63	4.08	1.17
4	5.73	5.50	6.21	-0.33	-8.36	19.42	12.68	11.41	-10.24
5	2.75	8.65	6.54	-3.91	-2.58	30.67	0.66	1.57	0.09
6	9.17	6.07	9.04	3.16	-12.56	13.03	4.05	0.17	1.36
7	2.45	6.30	6.70	18.13	16.06	16.54	2.50	5.95	6.44
8	7.39	5.49	5.56	22.17	16.05	17.81	6.63	4.72	5.32
9	1.34	7.36	6.18	-1.54	4.48	13.44	0.26	4.58	2.41
10	4.62	3.62	-1.18	21.98	11.23	1.91	3.24	3.07	-0.35
11	8.37	8.97	4.28	1.68	7.10	31.03	2.99	5.81	0.08
12	6.89	4.99	0.00	12.65	16.99	1.36	7.60	5.73	-0.05
13	8.42	0.95	3.43	27.19	-5.04	-1.29	0.22	0.16	1.06
14	5.53	8.40	4.67	25.03	17.59	19.74	4.97	8.91	2.96
15	4.11	6.44	6.52	12.67	14.21	14.26	3.08	5.89	5.34
16	3.96	6.35	2.72	24.27	12.37	10.12	3.35	6.81	2.99
17	6.44	4.54	0.69	19.94	15.11	11.67	6.19	3.95	-0.85
18	5.43	5.42	4.94	3.91	-0.37	24.24	2.54	1.91	0.17
19	7.13	9.63	1.64	23.91	24.24	17.30	7.27	9.83	-0.12
20	9.19	1.80	9.14	17.64	20.33	22.49	9.37	8.27	9.48
21	6.48	7.09	5.65	8.61	7.79	22.37	4.66	5.00	0.41
22	9.93	4.63	4.44	22.97	11.33	14.51	9.15	3.62	4.42
23	8.10	4.29	6.40	3.10	-6.08	5.11	3.59	0.39	1.61
24	8.08	7.67	4.24	18.86	17.78	11.96	7.41	8.31	3.53
25	3.45	5.30	9.28	24.94	18.50	16.39	2.38	5.29	8.52
26	8.56	12.10	6.84	2.93	7.15	24.28	3.79	6.95	0.55
27	5.15	7.99	4.80	8.88	4.65	22.59	3.96	4.17	0.29
28	-11.06	10.85	0.00	-30.82	16.56	16.63	-0.49	13.02	-1.87
29	-9.91	5.28	1.09	-25.71	16.09	20.99	2.35	8.69	1.96
30	5.74	6.33	4.57	13.90	14.64	17.69	6.38	7.50	5.84
31	4.12	4.49	5.83	-1.37	-10.71	6.54	1.03	0.21	2.18
32	10.09	3.22	8.01	24.61	15.74	16.58	9.79	1.74	8.50
33	5.06	3.82	3.73	22.50	16.32	17.01	4.43	2.83	2.30
34	6.22	5.13	9.49	18.85	15.72	18.31	6.08	4.87	10.21
35	10.68	10.49	6.79	15.66	21.65	17.31	11.62	12.49	6.42
36	7.76	2.67	-0.33	14.21	-6.53	13.58	6.33	0.15	-0.01
37	0.00	5.77	9.32	10.60	15.82	14.76	-0.40	5.64	9.63
38	2.34	4.38	10.35	7.23	17.59	20.54	3.50	6.19	10.55
39	5.65	3.12	11.94	22.00	22.16	28.29	5.59	3.62	12.03
40	4.87	6.82	1.14	6.19	-0.04	9.93	3.57	1.57	-0.27
41	2.76	7.14	2.48	13.32	16.71	19.80	1.81	7.81	2.42
42	2.49	7.42	6.27	16.68	13.36	16.49	0.90	9.03	5.02
43	7.08	4.77	4.90	24.11	15.58	14.96	6.75	3.88	3.58
44	7.84	4.15	8.23	24.87	15.00	14.85	8.40	1.85	8.98
45	9.35	0.50	5.33	18.52	14.74	6.56	9.61	-1.49	6.80
46	2.82	3.00	1.63	12.35	12.33	7.65	1.90	1.31	0.13
47	7.10	11.93	4.28	0.44	-4.31	17.86	2.78	1.38	0.08
48	6.98	3.80	9.72	16.14	11.61	16.30	6.87	3.52	11.98
49	4.26	9.25	1.24	15.41	14.60	13.80	3.22	9.23	-1.23
50	5.66	6.53	1.09	25.07	15.50	17.14	5.37	6.76	-3.60

6.3 Evaluación del modelo propuesto

Para finalizar, se realizó una evaluación del modelo propuesto. A nivel subjetivo, percibimos una separación aceptable, en la que, si bien en algunas canciones la eficiencia de la separación es mejor que otras, a nivel general se detecta claramente la predominancia del instrumento a separar.

En la tabla 6-10 se muestran las medidas de evaluación objetivas obtenidas sobre el conjunto de prueba de MusDB18-HQ.

La tabla 6-8 contiene los valores de la mediana para cada una de las medidas objetivas.

Tabla 6-8. Medidas de evaluación objetiva del modelo propuesto.

	SDR	**SIR**	**SAR**
Voz	3.41	9.65	2.05
Batería	3.97	8.64	3.09
Bajo	3.37	13.51	0.98

En la tabla 6-8 puede observarse que los resultados son inferiores a los ofrecidos por Open-Unmix (tabla 6-4) pero si representan una mejora significativa con respecto a los que obtuvimos con el modelo de DeepConvSep.

A su vez se observa que obtenemos un mejor SDR en la separación de la batería. Esto puede ser debido a lo comentado anteriormente en la evaluación de DeepConvSep ya que la arquitectura del modelo que proponemos está claramente inspirada en la de este modelo de separación.

La tabla 6-9 recoge los valores de SDR obtenidos para cada estilo musical.

Tabla 6-9. Valores de SDR por géneros musicales (modelo propuesto).

	Pop/Rock	**Reggae/Rap**	**Electrónica**	**Heavy Metal**
Voz	3.97	3.10	-14.37	3.85
Bajo	3.55	2.54	1.55	3.54
Batería	3.96	2.2	3.72	7.15

Tabla 6-10. Valores de los resultados de la evaluación objetiva del modelo propuesto.

ID canción	SDR			SIR			SAR		
	VOZ	BATERÍA	BAJO	VOZ	BATERÍA	BAJO	VOZ	BATERÍA	BAJO
1	5.51	3.93	-3.66	9.78	7.75	3.82	6.69	5.41	-2.69
2	4.86	2.28	3.41	6.01	0.34	12.78	3.45	0.83	1.32
3	6.03	3.63	2.25	5.55	2.94	18.53	3.58	2.01	-0.04
4	2.35	0.74	3.13	-0.20	-11.83	15.52	1.10	-0.09	0.11
5	1.99	7.38	4.23	-9.40	-0.55	26.48	0.24	2.10	0.09
6	6.25	4.42	7.11	4.46	-12.84	6.12	3.93	0.16	1.07
7	1.44	5.27	5.09	16.74	11.74	14.65	1.02	5.55	4.69
8	4.87	3.84	4.08	15.92	11.47	13.59	4.22	3.19	3.62
9	1.87	8.02	4.24	-1.81	4.85	9.62	-0.01	4.79	1.26
10	3.03	3.37	-0.26	16.39	9.75	0.19	2.10	3.14	0.00
11	6.17	8.96	2.96	-3.01	-0.78	28.47	1.25	2.36	0.04
12	3.47	5.36	0.06	9.51	12.00	-14.56	5.76	6.48	-0.07
13	4.85	1.76	-0.56	25.97	-7.49	-8.77	0.10	0.06	0.89
14	3.37	7.40	2.00	22.48	13.17	14.70	2.41	8.41	0.55
15	3.05	4.67	4.29	8.02	10.63	13.42	1.66	4.06	3.11
16	1.78	3.98	2.18	14.59	5.45	3.73	0.60	6.17	2.95
17	4.51	2.50	-0.90	16.83	9.12	7.79	3.92	1.20	-1.60
18	3.85	4.01	3.84	0.21	-5.03	19.32	1.08	0.63	0.08
19	4.43	6.83	0.33	17.72	18.48	16.51	3.85	7.59	-0.37
20	1.99	0.06	1.20	7.34	1.46	0.76	1.01	-1.34	2.39
21	3.27	6.09	3.98	3.32	6.96	20.11	1.45	4.34	0.34
22	8.12	3.73	3.84	21.54	10.11	13.26	7.50	2.75	3.19
23	4.99	3.94	5.31	1.08	-8.10	-1.01	2.00	0.38	0.59
24	3.10	5.38	2.54	10.82	10.58	11.64	2.99	6.44	1.14
25	0.61	3.97	5.94	19.40	17.02	11.36	0.71	4.22	5.50
26	4.45	9.57	5.02	-5.60	-1.88	20.64	0.79	2.06	0.13
27	4.80	6.69	3.63	2.57	0.64	20.35	2.39	2.50	0.17
28	-44.21	8.26	0.27	-40.33	12.45	3.88	-0.44	11.69	-0.96
29	-18.93	4.94	0.92	-26.78	12.53	20.12	0.00	8.70	1.80
30	3.45	3.18	2.32	10.44	5.03	18.73	4.21	7.86	3.13
31	2.56	3.56	3.53	-7.24	-0.66	8.03	0.00	1.58	0.34
32	6.92	1.91	6.24	17.42	8.51	15.05	6.28	1.78	6.83
33	2.53	2.37	0.05	16.84	10.72	9.00	1.84	0.80	-0.05
34	3.30	3.82	7.28	14.49	11.29	13.79	2.77	3.41	7.78
35	7.43	8.15	3.34	11.51	15.79	13.62	9.63	10.49	3.18
36	5.41	2.20	0.00	11.70	-5.85	16.76	4.04	0.26	-0.05
37	0.50	3.24	6.88	3.92	9.80	13.02	-3.02	3.04	7.21
38	0.93	2.44	6.63	7.16	9.86	18.79	1.19	3.88	7.15
39	4.09	1.67	11.12	15.29	11.11	23.01	4.03	2.58	12.05
40	2.73	4.78	0.10	1.95	0.44	18.12	0.75	1.80	-0.05
41	1.16	5.45	2.48	7.51	11.85	14.26	2.18	6.13	2.53
42	0.82	6.26	4.60	9.13	10.74	18.10	0.44	6.99	3.36
43	4.81	2.71	3.81	18.67	8.79	11.84	4.20	1.60	2.89
44	5.63	4.23	7.12	19.95	11.67	11.78	5.43	4.18	8.37
45	7.56	1.37	1.59	13.91	5.28	0.10	7.88	0.89	9.21
46	1.40	2.19	-3.49	7.59	7.82	1.61	-0.36	1.06	-5.42
47	5.15	10.28	3.54	-7.71	-5.19	16.71	0.56	1.10	0..06
48	5.18	2.89	7.57	14.68	6.36	15.33	4.80	3.24	9.72
49	1.75	7.15	0.42	14.47	11.29	6.75	0.56	8.00	-4.54
50	3.23	5.78	1.09	18.77	12.52	6.21	2.31	6.09	-1.28

7 CONCLUSIONES Y FUTUROS PASOS

En este último capítulo se expondrán las distintas conclusiones que se han ido obteniendo durante la realización de este trabajo. Además, para finalizar, se propondrán algunas líneas de investigación.

7.1 Conclusiones

A lo largo de este trabajo se ha analizado la arquitectura, el funcionamiento y rendimiento de diversos algoritmos destacados del estado del arte de la separación automática de fuentes musicales. Se decidió centrar el estudio en dos de ellos: DeepConvSep y Open-Unmix.

En un primer lugar se optó por el algoritmo DeepConvSep ya que nuestro trabajo se inició estudiando la tesis doctoral *"Source Separation Methods for Orchestral Music: Timbre-Informed and Score-Informed Strategies"* elaborada por Marius Miron de la Universidad Pompeu Fabra [42] y en ella se proponía este interesante modelo de separación.

Una vez analizado, entrenado y evaluado este modelo que hace uso de redes convolucionales en su tarea de separación, se decidió profundizar en otro algoritmo que usase otro tipo de redes neuronales y así poder comparar el desempeño de unas y otras redes. Para ello, se escogió Open-Unmix debido a su facilidad de reproducción, modificación y ampliación además de que ofrecía unos buenos resultados de separación.

Si bien el estudio y reproducción de DeepConvSep fue un buen punto de partida para adentrarnos en el conocimiento de las redes neuronales, desde su publicación ha pasado unos años, con el consiguiente desarrollo de nuevos modelos que ofrecen unos resultados superiores. Además, este algoritmo está implementado en Python 2.7, el cual se encuentra desactualizado en la actualidad.

Es por ello por lo que se consideró interesante implementar un modelo inspirado en las redes convolucionales que se proponen en DeepConvSep aprovechando el software de código abierto que Open-Unmix ofrece como punto de partida para la investigación de esta temática. Esto contribuyó a la consecución de los objetivos iniciales que se establecieron en la etapa inicial de este trabajo.

Si bien con nuestro modelo propuesto no conseguimos igualar los resultados alcanzados por Open-Unmix, el uso de un software actualizado o el empleo de funciones de activación ReLU a la salida de las capas convolucionales, entre otras cosas, supusieron una mejora en los resultados en comparación a los que se obtuvieron con DeepConvSep.

Como hemos comentado, durante todo el trabajo hemos optado por implementaciones de código abierto lo que ha supuesto una gran ayuda en términos de avance en la investigación. Además de los modelos expuestos a lo largo de esta memoria se han analizado otros, los cuales han sido finalmente descartados debido a su complejidad o a inconvenientes surgidos en el entrenamiento de estos tales como disponer de una capacidad computacional insuficiente.

7.2 Futuros pasos

Para obtener una separación de buena calidad mediante el uso de algoritmos de aprendizaje máquina, se necesita disponer de una gran cantidad de datos. Trabajos futuros podrían centrarse en la creación de bases de datos más amplias y de estilos musicales concretos que permita a los modelos especializarse en ellos.

Si bien el uso de la STFT simplifica enormemente las arquitecturas de redes empleadas en los sistemas de separación de fuentes, no resulta adecuado para todas las señales que están presentes en las composiciones musicales. Además, descartando la información de fase podría perderse información relevante. Es por ello por lo que optar por otras técnicas como las ventanas adaptativas para representar el audio podrían mejorar los sistemas de separación.

REFERENCIAS

[1] E. C. Cherry, "Some Experiments on the Recognition of Speech, with One and with Two Ears," *J Acoust Soc Am*, pp. 975–979, 1953.

[2] D. Ward, R. D. Mason, C. Kim, F.-R. Stöter, A. Liutkus, and M. D. Plumbley, "SiSEC 2018: State of the art in musical audio source separation - subjective selection of the best algorithm," in *WIMP: Workshop on Intelligent Music Production*, Huddersfield, United Kingdom, Sep. 2018. [Online]. Available: https://inria.hal.science/hal-01945362

[3] Ethan Manilow, Prem Seetharman, and Justin Salamon, *Open Source Tools & Data for Music Source Separation*. 2020.

[4] Mathworks, "¿Qué es Machine Learning?" https://es.mathworks.com/discovery/machine-learning.html#-por-qu%C3%A9-es-importante (accessed Jul. 19, 2023).

[5] AWS, "¿Qué es el aprendizaje automático?" https://aws.amazon.com/es/what-is/machine-learning/ (accessed Jul. 22, 2023).

[6] H. Abdi and L. Williams, "Principal Component Analysis," *Wiley Interdiscip Rev Comput Stat*, vol. 2, pp. 433–459, Jul. 2010, doi: 10.1002/wics.101.

[7] S. Sra and I. Dhillon, "Generalized Nonnegative Matrix Approximations with Bregman Divergences," in *Advances in Neural Information Processing Systems*, Y. Weiss, B. Schölkopf, and J. Platt, Eds., MIT Press, 2005. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2005/file/d58e2f077670f4de9cd7963c857f2534-Paper.pdf

[8] IBM, "¿Qué son las redes neuronales?" https://www.ibm.com/es-es/topics/neural-networks (accessed Jul. 23, 2023).

[9] L. Guesmi, H. Fathallah, and M. Menif, "Modulation Format Recognition Using Artificial Neural Networks for the Next Generation Optical Networks," 2018. doi: 10.5772/intechopen.70954.

[10] L. Pepino, "Separación de fuentes musicales mediante redes neuronales convolucionales." 2019.

[11] Jaime Durán, "Técnicas de Regularización Básicas para Redes Neuronales," Oct. 08, 2019. https://medium.com/metadatos/t%C3%A9cnicas-de-regularizaci%C3%B3n-b%C3%A1sicas-para-redes-neuronales-b48f396924d4#:~:text=Normalizaci%C3%B3n%20por%20lotes%20(Batch%20normalization)&text=La%20normalizaci%C3%B3n%20en%20lotes%20consiste,normalizar%20las%20activaciones%20de%20salida. (accessed Jul. 24, 2023).

[12] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014, [Online]. Available: http://jmlr.org/papers/v15/srivastava14a.html

[13] IBM, "¿Qué son las redes neuronales convolucionales?" https://www.ibm.com/es-es/topics/convolutional-neural-networks (accessed Jul. 24, 2023).

[14] M. Yani, Irawan S, and C. Setianingsih, "Application of Transfer Learning Using Convolutional Neural Network Method for Early Detection of Terry's Nail," *J Phys Conf Ser*, vol. 1201, p. 12052, Jul. 2019, doi: 10.1088/1742-6596/1201/1/012052.

[15] IBM, "What are recurrent neural networks?" https://www.ibm.com/topics/recurrent-neural-networks (accessed Jul. 24, 2023).

[16] "Understanding LSTM Networks," Aug. 27, 2015. https://colah.github.io/posts/2015-08-Understanding-LSTMs/ (accessed Jul. 24, 2023).

[17] A. Ali *et al.*, "Speaker Gender Recognition Based on Deep Neural Networks and ResNet50," *Wirel Commun Mob Comput*, vol. 2022, pp. 1–13, Jul. 2022, doi: 10.1155/2022/4444388.

[18] H. Jeon, Y. Jung, S. Lee, and Y. Jung, "Area-Efficient Short-Time Fourier Transform Processor for Time–Frequency Analysis of Non-Stationary Signals," *Applied Sciences*, vol. 10, p. 7208, Jul. 2020, doi: 10.3390/app10207208.

[19] Z. Shi, H. Lin, L. Liu, R. Liu, and J. Han, "Is CQT more suitable for monaural speech separation than STFT? an empirical study." 2019.

[20] F. Stoter, A. Liutkus, R. Badeau, B. Edler, and P. Magron, "Common Fate Model for Unison Source Separation," Jul. 2016. doi: 10.1109/ICASSP.2016.7471650.

[21] F. Pishdadian and B. Pardo, "Multi-Resolution Common Fate Transform," *IEEE/ACM Trans Audio Speech Lang Process*, vol. 27, no. 2, pp. 342–354, 2019, doi: 10.1109/TASLP.2018.2878616.

[22] A. Narayanan and D. Wang, "Ideal ratio mask estimation using deep neural networks for robust speech recognition," *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 7092–7096, 2013.

[23] E. Vincent, R. Gribonval, and C. Fevotte, "Performance measurement in blind audio source separation," *IEEE Trans Audio Speech Lang Process*, vol. 14, no. 4, pp. 1462–1469, 2006, doi: 10.1109/TSA.2005.858005.

[24] P. Smaragdis, C. Févotte, G. J. Mysore, N. Mohammadiha, and M. Hoffman, "Static and Dynamic Source Separation Using Nonnegative Factorizations: A unified view," *IEEE Signal Process Mag*, vol. 31, no. 3, pp. 66–75, 2014, doi: 10.1109/MSP.2013.2297715.

[25] P. Magron and T. Virtanen, "Complex ISNMF: a Phase-Aware Model for Monaural Audio Source Separation," *CoRR*, vol. abs/1802.03156, 2018, [Online]. Available: http://arxiv.org/abs/1802.03156

[26] E. M. Grais, M. U. Sen, and H. Erdogan, "Deep neural networks for single channel source separation," *CoRR*, vol. abs/1311.2746, 2013, [Online]. Available: http://arxiv.org/abs/1311.2746

[27] S. Uhlich, F. Giron, and Y. Mitsufuji, "Deep neural network based instrument extraction from music," in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2015, pp. 2135–2139. doi: 10.1109/ICASSP.2015.7178348.

[28] S. Uhlich *et al.*, "Improving music source separation based on deep neural networks through data augmentation and network blending," *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 261–265, 2017, [Online]. Available: https://api.semanticscholar.org/CorpusID:9823207

[29] M. and J. J. and G. E. Chandna Pritish and Miron, "Monoaural Audio Source Separation Using Deep Convolutional Neural Networks," in *Latent Variable Analysis and Signal Separation*, M. and M. O. J. J. and T.-M. N. Tichavský Petr and Babaie-Zadeh, Ed., Cham: Springer International Publishing, 2017, pp. 258–266.

[30] J.-Y. Liu and Y.-H. Yang, "Denoising Auto-encoder with Recurrent Skip Connections and Residual Regression for Music Source Separation," *CoRR*, vol. abs/1807.01898, 2018, [Online]. Available: http://arxiv.org/abs/1807.01898

[31] A. A. Nugraha, A. Liutkus, and E. Vincent, "Multichannel music separation with deep neural networks," in *2016 24th European Signal Processing Conference (EUSIPCO)*, 2016, pp. 1748–1752. doi: 10.1109/EUSIPCO.2016.7760548.

[32] F.-R. Stöter, S. Uhlich, A. Liutkus, and Y. Mitsufuji, "Open-Unmix - A Reference Implementation for Music Source Separation," *J Open Source Softw*, vol. 4, no. 41, p. 1667, 2019, doi: 10.21105/joss.01667.

[33] Z. Rafii, A. Liutkus, F.-R. Stöter, S. I. Mimilakis, and R. Bittner, "MUSDB18 - a corpus for music separation," Dec. 2017, doi: 10.5281/ZENODO.1117372.

[34] N. Takahashi and Y. Mitsufuji, “D3Net: Densely connected multidilated DenseNet for music source separation.” 2021.

[35] R. Hennequin, A. Khlif, F. Voituret, and M. Moussallam, “Spleeter: a fast and efficient music source separation tool with pre-trained models,” *J Open Source Softw*, vol. 5, p. 2154, Aug. 2020, doi: 10.21105/joss.02154.

[36] D. Stoller, S. Ewert, and S. Dixon, “Wave-U-Net: A Multi-Scale Neural Network for End-to-End Audio Source Separation,” *CoRR*, vol. abs/1806.03185, 2018, [Online]. Available: http://arxiv.org/abs/1806.03185

[37] Y. Luo and N. Mesgarani, “TaSNet: Time-Domain Audio Separation Network for Real-Time, Single-Channel Speech Separation,” Aug. 2018, pp. 696–700. doi: 10.1109/ICASSP.2018.8462116.

[38] Y. Luo and N. Mesgarani, “TasNet: Surpassing Ideal Time-Frequency Masking for Speech Separation,” *CoRR*, vol. abs/1809.07454, 2018, [Online]. Available: http://arxiv.org/abs/1809.07454

[39] A. Défossez, N. Usunier, L. Bottou, and F. R. Bach, “Music Source Separation in the Waveform Domain,” *CoRR*, vol. abs/1911.13254, 2019, [Online]. Available: http://arxiv.org/abs/1911.13254

[40] A. Liutkus *et al.*, “The 2016 Signal Separation Evaluation Campaign,” in *Latent Variable Analysis and Signal Separation - 12th International Conference, LVA/ICA 2015, Liberec, Czech Republic, August 25-28, 2015, Proceedings*, P. Tichavský, M. Babaie-Zadeh, O. J. J. Michel, and N. Thirion-Moreau, Eds., Cham: Springer International Publishing, 2017, pp. 323–332.

[41] F.-R. Stöter, “Open Unmix.” https://github.com/sigsep/open-unmix-pytorch (accessed Aug. 31, 2023).

[42] M. Miron, “Source Separation Methods for Orchestral Music: Timbre-Informed and Score-Informed Strategies,” Pompeu Fabra University, Barcelona, 2018. doi: 10.5281/zenodo.1163675.

GLOSARIO

ANN:	Artificial Neural Network
BPTT:	Backpropagation through time
CFT:	Common Fate Transform.
COLA:	Constant-Overlap-Add.
CPU:	Central Processing Unit.
CQT:	Constant-Q Transform.
DFT:	Discrete Fourier Transform.
DL:	Deep Learning.
GPU:	Graphics Processing Unit.
LSTM:	Long Short-Term Memory.
MCFT:	Multi-resolution Common Fate Transform.
MIR:	Music Information Retrieval.
ML:	Machine Learning.
MSS:	Music Source Separation.
NMF:	Non-negative Matrix Factorization.
PCA:	Principal Component Analysis.
ReLU:	Rectified Lineal Unit.
RNN:	Recurrent Neural Network.
SAR:	Source-to-Artifact-Ratio.
SDR:	Source-to-Distortion-Ratio.
SIR:	Source-to-Interference-Ratio.
SISSEC:	Signal Separation Evaluation Campaign.
SSS:	Sound Source Separation.
STFT:	Short-Time Fourier Transform.

ANEXO A: CÓDIGO IMPLEMENTADO

Este anexo contiene el código del modelo de red neuronal implementado.

```
from typing import Optional, Mapping
import numpy as np
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch import Tensor
from torch.nn import BatchNorm1d, Linear, Parameter
from .filtering import wiener
from .transforms import make_filterbanks, ComplexNorm



class OpenUnmix_conv(nn.Module):

    """
    Args:
        nb_bins (int): Number of input time-frequency bins (Default: `4096`).
        nb_channels (int): Number of input audio channels (Default: `2`).
        hidden_size (int): Size for bottleneck layers (Default: `512`).
        input_mean (ndarray or None): global data mean of shape `(nb_bins, )`.
            Defaults to zeros(nb_bins)
        input_scale (ndarray or None): global data mean of shape `(nb_bins, )`.
            Defaults to ones(nb_bins)
        max_bin (int or None): Internal frequency bin threshold to
            reduce high frequency content. Defaults to `None` which results
            in `nb_bins`
    """
```

```
def __init__(
    self,
    nb_bins: int = 4096,
    nb_channels: int = 2,
    hidden_size: int = 512,
    input_mean: Optional[np.ndarray] = None,
    input_scale: Optional[np.ndarray] = None,
    max_bin: Optional[int] = None,
):
    super(OpenUnmix_conv, self).__init__()
    self.nb_output_bins = nb_bins
    if max_bin:
        self.nb_bins = max_bin
    else:
        self.nb_bins = self.nb_output_bins
    self.hidden_size = hidden_size



    self.conv_hor = (2049, 1)
    self.conv_ver = (1, 15)
    self.num_ch_out_hor=50
    self.num_ch_out_ver=30

    ### ENCODER
    self.encoder = nn.Sequential(
        nn.Conv2d(2, self.num_ch_out_hor, self.conv_hor, stride = 1, padding = 0, bias = True, groups = 1),
        nn.Conv2d(self.num_ch_out_hor, self.num_ch_out_ver, self.conv_ver, stride = 1, padding = 0, bias = True,
                  groups = 1)
    )

    ### DECODER
    self.decoder = nn.Sequential(
        nn.ConvTranspose2d(self.num_ch_out_ver, self.num_ch_out_hor, self.conv_ver, stride = 1, padding = 0,
                           bias = True, groups = 1),
        nn.ConvTranspose2d(self.num_ch_out_hor, 2, self.conv_hor, stride = 1, padding = 0, bias = True, groups = 1),

    ### CAPAS COMPLETAMENTE CONECTADAS
```

```
        self.layer = nn.Sequential(
            nn.Linear(30, 512),
            nn.ReLU(),
            nn.Linear(512,30),
            nn.ReLU(),
        )


        if input_mean is not None:
            input_mean = torch.from_numpy(-input_mean[: self.nb_bins]).float()
        else:
            input_mean = torch.zeros(self.nb_bins)

        if input_scale is not None:
            input_scale = torch.from_numpy(1.0 / input_scale[: self.nb_bins]).float()
        else:
            input_scale = torch.ones(self.nb_bins)

        self.input_mean = Parameter(input_mean)
        self.input_scale = Parameter(input_scale)

        self.output_scale = Parameter(torch.ones(self.nb_output_bins).float())
        self.output_mean = Parameter(torch.ones(self.nb_output_bins).float())



    def freeze(self):
        # set all parameters as not requiring gradient, more RAM-efficient
        # at test time
        for p in self.parameters():
            p.requires_grad = False
        self.eval()
```

```
    def forward(self, x: Tensor) -> Tensor:

"""
        Args:
            x: input spectrogram of shape
                `(nb_samples, nb_channels, nb_bins, nb_frames)`

        Returns:
            Tensor: filtered spectrogram of shape
                `(nb_samples, nb_channels, nb_bins, nb_frames)`
"""

        mix = x.detach().clone()
        x = self.encoder(x)
        x = self.layer(x.view(-1,30))
        x = x.view(nb_samples,30,1,-1)
        x = self.decode(x)
        x = F.relu(x) * mix
        return
```