Universidad Nacional Autónoma de México

Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas

Separación de la guitarra acústica de la mezcla de una canción

T E S I S

que para obtener el título de:

Maestro en Ciencia e Ingeniería de la Computación

presenta:

Edgar Rigoberto Santos Martín

Tutor:

Caleb Antonio Rascón Estebané

Mérida, Yucatán, 2022

Dedico este trabajo a mis padres,
a mis hermanos y a Dios.

Agradecimientos

Quisiera agradecer primeramente a Dios por la vida y salvación que me ha dado. También agradezco a mis padres por su amor y formación en mi vida.

Agradezco a mi tutor el Dr. Caleb Rascón, por todo su apoyo, sus valores, sus conocimientos y su experiencia, lo cual me ayudó para realizar este trabajo.

Agradezco a la Universidad Nacional Autónoma de México y al Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas por darme la oportunidad de estudiar esta maravillosa maestría. Y al Consejo Nacional de Ciencia y Tecnología por permitirme a mi, y a muchos estudiantes, tener la dicha de estudiar posgrados de alta calidad.

También agradezco al Programa de Apoyo a Proyectos de Investigación e Innovación Tecnológica (PAPIIT), con el proyecto IA100222, el cual me apoyó en las últimas fases de la maestría.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

La música es producida por la mezcla de sonidos provenientes de varios instrumentos individuales [1]. En una canción, cada instrumento representa una pista; al ensamblar todas las pistas (involucrando también producción musical) se produce la mezcla final de una canción.

En 1953, en [2], se propuso el efecto *Cocktail Party*, el cual consiste en que el cerebro humano es capaz de prestar atención a una conversación aun cuando alrededor existe mucho ruido y mucha gente hablando al mismo tiempo. Las computadoras requieren algoritmos para poder replicar esta habilidad humana.

La finalidad de la separación de fuentes musicales es obtener las pistas individuales de la mezcla final. A diferencia del problema del *Cocktail Party*, en la música no hay una sola fuente de interés, sino una amplia variedad de tonos y timbres reproduciéndose de una forma coordinada [3] [4].

Hoy en día existen una variedad de modelos de aprendizaje profundo que han logrado realizar la separación de la batería, bajo, voces y "otros"; donde "otros" son los demás instrumentos que no se separaron, o también puede verse como el residuo de la separación de la batería, bajo y las voces. Muy pocos modelos-aplicaciones separan hasta el piano [5] [6] [7] [8] y solamente unos cuantos [6] [7] [8] [9], separan la guitarra acústica y/o eléctrica. Aún no se ha creado un modelo que reconozca perfectamente y separe todos los instrumentos que existen en una canción.

1.1. Motivación

En muchos escenarios, investigadores han descubierto que es más fácil procesar las fuentes aisladas que la mezcla de ellas [10]. En el ámbito musical, obtener las pistas de cada instrumento de una canción, es esencial. Sin embargo, la separación de fuentes musicales no solo tiene aplicaciones en el ámbito musical.

En la siguiente lista se muestran varias aplicaciones de la separación de fuentes musicales:

- Para que músicos puedan escuchar solo el instrumento que les toca interpretar y puedan practicar con más facilidad.
- Para reconocimiento de voz en asistentes celulares (por ejemplo: Siri, Alexa, Google) en lugares con música a alta presión sonora.
- Cuando a los productores de música se les entrega por el artista o banda sólo la maqueta inicial de su canción (no multipista); la cual es grabada con todos los músicos tocando al mismo tiempo.
- Para que estudiantes de producción musical tengan material con el que puedan practicar.
- Para facilitar la transcripción de las notas y partituras o incluso para transcriptores de notas automáticos.
- Para arreglistas de música orquestal.
- Para facilitar el reconocimiento automático de las letras en una canción.
- En canciones donde la voz no se entiende mucho por la forma de grabación, se puede usar el sistema para aumentar el volumen o inteligibilidad de la voz.
- Para remasterizar una canción antigua.
- Para producción con *samples* (creación de música electrónica).
- Para obtener la base rítmica de una canción.

- Para mejorar la detección de actividad vocal.
- Cuando alguien quiere editar la mezcla de una canción por gusto personal.
- Para generar música para *karaoke*.
- Para dispositivos de ayuda auditiva.
- Para alineamiento de instrumentos o voces.
- Para detección automática de instrumentos.
- Como herramienta de enseñanza musical.
- Para estimación de frecuencias fundamentales.

La guitarra acústica es un instrumento muy utilizado en la música de todo tipo de géneros. En la época del Romanticismo musical, la guitarra acústica se utilizaba en algunas obras e interpretaciones [11]. También, se tenía la idea de que la guitarra acústica era como un tipo de orquesta pequeña. Desde entonces, ha sido un instrumento importante en la música a lo largo de la historia.

Desafortunadamente, como se ha mencionado anteriormente, muy pocos modelos separan este instrumento, a pesar de su gran popularidad.

1.2. Hipótesis

Las herramientas basadas en aprendizaje profundo que actualmente separan otros instrumentos musicales, así como aquellos que separan la voz, son viables para separar la guitarra acústica de la mezcla de una canción. Esto es debido a que en el estado del arte ya se ha logrado realizar separación de bajo, batería y voces mediante diferentes modelos de aprendizaje profundo.

1.3. Objetivos

1.3.1. Objetivo general

Obtener un modelo de aprendizaje profundo el cual logre separar la guitarra acústica de una canción.

1.3.2. Objetivos específicos

- Realizar una revisión del estado del arte en materia de separación de fuentes musicales.
- Generar una base de datos adecuada para el entrenamiento de los modelos.
- Buscar repositorios especializados en separación de fuentes musicales, para facilitar la programación.
- Probar diferentes arquitecturas de aprendizaje profundo.

1.4. Estructura de la Tesis

Como se ha visto, el capítulo actual es la introducción de la tesis; se presentó el planteamiento del problema, la motivación, la hipótesis y los objetivos.

En el capítulo 2 se presentan los antecedentes y el marco teórico; en el cual se explica la separación de fuentes musicales, el aprendizaje profundo para separación de fuentes, la guitarra acústica, el análisis musical y lo que es MIDI.

En el capítulo 3 se detalla toda la metodología con la cual se llevó a cabo el proyecto de esta tesis. Se explica cómo se generó la base de datos utilizada, toda la programación realizada, el entrenamiento de los modelos de aprendizaje profundo y como se obtuvieron las inferencias

En el capítulo 4 se muestran las características e hiperparámetros utilizados para cada modelo, se explica como se realizó la evaluación de los modelos, se muestran resultados obtenidos y la discusión de los mismos.

En el último capítulo, el capítulo 5, se presentan las conclusiones que se formularon al finalizar este proyecto de tesis y el posible trabajo a futuro.

Capítulo 2

Antecedentes y marco teórico

Dado el efecto *Cocktail Party* [2], se han creado algoritmos y sistemas que intentan realizar la separación de los diferentes hablantes que existen en ambientes ruidosos. Algunas de las técnicas que se utilizan para realizar esta separación son: *beamforming*, máscara de frecuencias y análisis estadísticos. Si se quiere profundizar más en estos temas se pueden consultar los artículos [12] y [13].

En una canción, no solo existe una fuente de interés como ya se ha mencionado anteriormente; todos los instrumentos son una fuente de interés y no pueden ser considerados como ruido.

También existen sistemas que realizan separación de eventos sonoros o sonidos ambientales, es decir de una grabación en la calle, por ejemplo, la finalidad es obtener las diferentes fuentes que se escuchan como puede ser automóviles, sirenas de ambulancia, personas hablando, un perro ladrando, etcétera.

En la siguiente sección se hablará más sobre la separación de fuentes musicales. Gran parte de la información se tomó del libro *Open Source Tools & Data for Music Source Separation* [10], el cual fue escrito por Ethan Manilow, Prem Seetharaman y Justin Salamon; quienes, junto con otros autores, han trabajado en la creación de modelos y herramientas para la separación de fuentes musicales.

2.1. Separación de fuentes musicales

La separación de fuentes musicales es el proceso inverso de mezclar una canción; es decir que al tener una canción, la separación de fuentes musicales se encarga de separar cada instrumento, o grupo de instrumentos, que se encuentra en la canción. En la figura 2.1 se puede ver un diagrama del funcionamiento de la separación de fuentes musicales. Del lado izquierdo de la figura se encuentra la señal de la canción con todos los instrumentos mezclados; luego pasa por el sistema de separación de fuentes musicales y se obtienen las pistas de cada instrumento (lado derecho de la figura).

2.1.1. Separación de fuentes

Se deben sentar las bases primero de la separación de fuentes, para poder hablar luego de la separación de fuentes musicales. Matemáticamente hablando, una señal de una mezcla $y(t)$ se puede descomponer en N fuentes, $x_n(t)$ para $n = 1...N$ tal que

$$y(t) = \sum_{i=1}^{N} x_i(t).$$

Dado únicamente $y(t)$ la finalidad de la separación de fuentes es obtener una o más $x(t)$'s. Las mezclas se componen de más de una señal; es por eso que la separación de fuentes es considerado un problema indeterminado; es decir, se tienen menos observaciones (la mezcla) que resultados requeridos (fuentes).

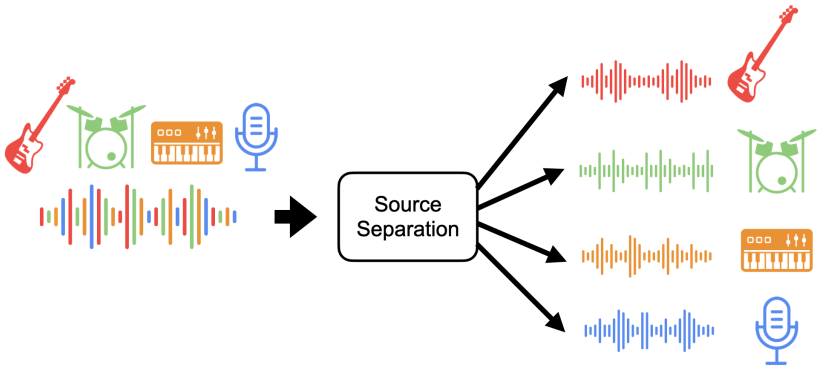

Figura 2.1: Diagrama de separación de fuentes musicales. Imagen extraída de [10].

Las señales musicales tienen distintas características que la diferencian de otros tipos de señales de audio, tiene muchos factores que la hacen desafiante [14]:

- Los instrumentos están muy correlacionados. Usualmente cambian al mismo tiempo.
- La música es mezclada y procesada de una forma no física y no lineal. A las grabaciones de los instrumentos se les agregan efectos, reverberación, filtros y otros tipos de procesamiento de señales no lineales, lo cual dificulta la separación. Este es un problema ya que rara vez se sabe qué procesamientos se aplicaron.
- Los usuarios de sistemas de separación de fuentes musicales esperan obtener una muy alta calidad en las separaciones.

2.1.2. Representación del audio

En su forma menos procesada, se asume que el audio se guarda digitalmente como una forma de onda. Algunos sistemas de separación de fuentes utilizan las formas de onda directamente, pero esto requiere mucho pre-procesamiento antes de poder separar. Cuando se genera un sonido (acústicamente hablando) la presión del aire se mueve a lo largo del tiempo. Este sonido se graba con un micrófono, el cual convierte los cambios de la presión del aire en señales eléctricas. Los voltajes de estas señales son muestreadas en intervalos de tiempo iguales, convirtiendo así la señal del sonido analógico en digital. Este arreglo digital es a lo que llamamos forma de onda. En este proceso se involucran detalles del campo de la física, acústica y procesamiento de señales. Es importante saber que la señal de tiempo continuo se discretiza tanto en tiempo como en amplitud.

En la figura 2.2 se puede ver una forma de onda de una canción de 8 segundos. La forma de onda se generó con un software de grabación y edición de audio llamado *Audacity* [15].

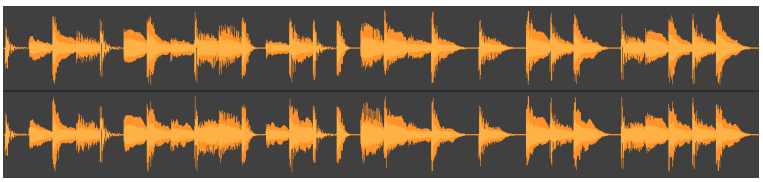

Figura 2.2: Forma de onda de una canción.

Una señal monofónica, o mono, es aquella que solo tiene un canal. Es decir, su arreglo tiene la forma: $x \in \mathbb{R}^{t\times 1}$. Una señal es estereofónica, o estéreo, si el arreglo tiene dos canales. Es decir, tiene la forma: $x \in \mathbb{R}^{t\times 2}$. La música es presentada de manera estereofónica, sin embargo, hay algunas aplicaciones en las que se presenta con más canales como el *surround* 5.1 o 7.1.

Un aspecto importante en las señales es la frecuencia de muestreo, la cual describe cuántas muestras se obtienen en un segundo y se mide en Hertz (Hz). Dada la frecuencia de muestreo de una señal sr, la frecuencia máxima que puede ser presentada es $f_N = \frac{sr}{2}$, esta es la frecuencia de Nyquist [16]. Por ejemplo, si la frecuencia de muestreo es de 44.1 kHz, la frecuencia máxima presentada sería 22.05 kHz. Muchos modelos de separación de fuentes basados en aprendizaje profundo reducen la frecuencia de muestreo de las señales de entrada para reducir la carga computacional durante el entrenamiento. Hacer esto reduce las altas frecuencias de las señales.

La mayoría de los métodos de separación de fuentes siguen tres pasos:

1. Convertir el audio a una representación fácil de separar.

2. Separar el audio manipulando su representación.

3. Convertir el audio manipulado a su representación original, para obtener las fuentes separadas.

Por lo tanto, un aspecto importante en la representación del audio es que sea reversible, es decir, que la señal que se convierte de forma de onda a una representación diferente, pueda volver a convertirse a forma de onda con ningún o muy poco error. Muchas veces al realizar la conversión de vuelta a forma de onda surgen artefactos en el audio, estos son los tipos de errores que podrían existir. Algunas representaciones pueden ser más adecuadas que otras para separar ciertas señales.

Una representación tiempo-frecuencia es una matriz de 2 dimensiones que representa las frecuencias de un audio a lo largo del tiempo [17]. A una entrada específica en esta matriz se le llama *bin* de tiempo-frecuencia. Se puede visualizar una representación tiempo-frecuencia utilizando un mapa de calor o un espectrograma, el cual tiene el tiempo a lo

largo del eje horizontal y a lo largo del eje vertical las frecuencias. Cada *bin* de tiempo-frecuencia en el espectrograma representa la amplitud de la señal en un tiempo y frecuencia en específico. Algunos de estos mapas muestran una barra de colores la cual ilustra a qué intensidad de amplitud pertenece cada color. Si no se muestra una barra de colores, se asume que los colores más brillantes indican amplitudes mayores que los colores más oscuros. Las representaciones de tiempo-frecuencia son las más usadas en separación de fuentes. Muchas de las representaciones de tiempo-frecuencia utilizan la transformada de Fourier de tiempo corto (STFT por sus siglas en inglés). La STFT se calcula a partir de la forma de onda de la señal, calculando una transformada de Fourier discreta (DFT por sus siglas en inglés) de pequeñas ventanas a lo largo de todo el arreglo. El valor absoluto de un *bin* de tiempo-frecuencia $|X(t, f)|$ en tiempo t y frecuencia f determina la cantidad de energía de la frecuencia f en el tiempo t. Cada *bin* en la STFT es un número complejo, es decir, que tiene un componente de magnitud y uno de fase. La STFT es reversible, haciendo que sea posible convertirla de nuevo a forma de onda, a esta se le llama la transformada inversa de Fourier de tiempo corto (ISTFT por sus siglas en inglés) y requiere tanto la componente de magnitud como la de fase para realizar el proceso. [10].

En la figura 2.3 se muestra el espectrograma de la misma canción de 8 segundos que se mostró en la figura 2.2. En el eje horizontal se muestran los segundos de la canción y en el eje vertical la frecuencia en kHz. La imagen se obtuvo con un software analizador de espectro

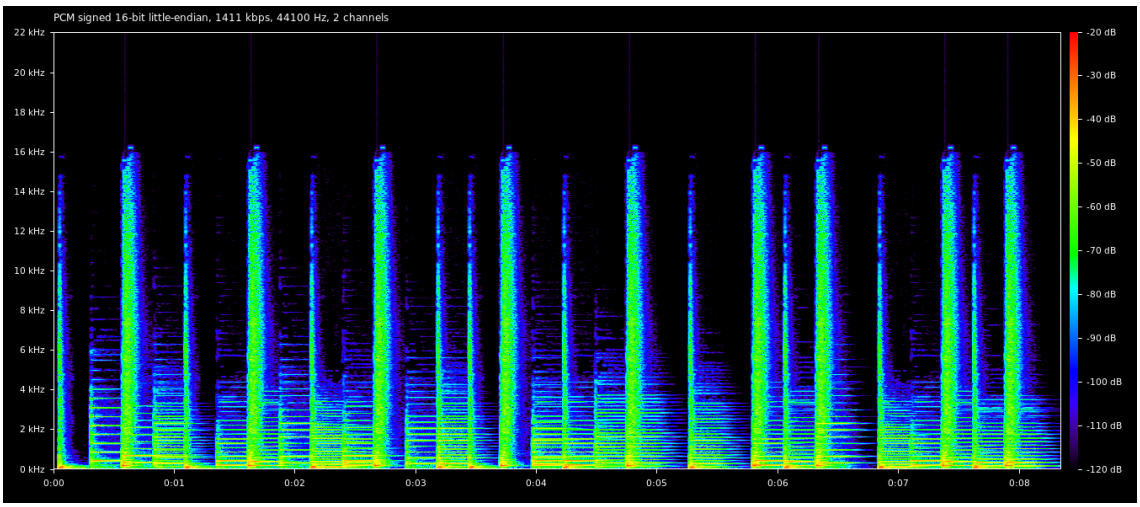

Figura 2.3: Espectrograma de una canción.

llamado Spek [18].

Algunos parámetros a considerar para calcular la STFT son:

- Tipo de ventana: El tipo de ventana determina la forma de los segmentos de audio que se tomarán, antes de aplicar la DFT. La forma de esta ventana afecta a qué frecuencias se enfatizan o cuáles se atenúan. Existen varios tipos de ventanas, como por ejemplo: Blackman, Chebwin, Coseno, exponencial, Gaussiano, Hamming, Hann, rectangular, triangular, entre otros. En la figura 2.4 se muestran algunas de las ventanas más usadas.

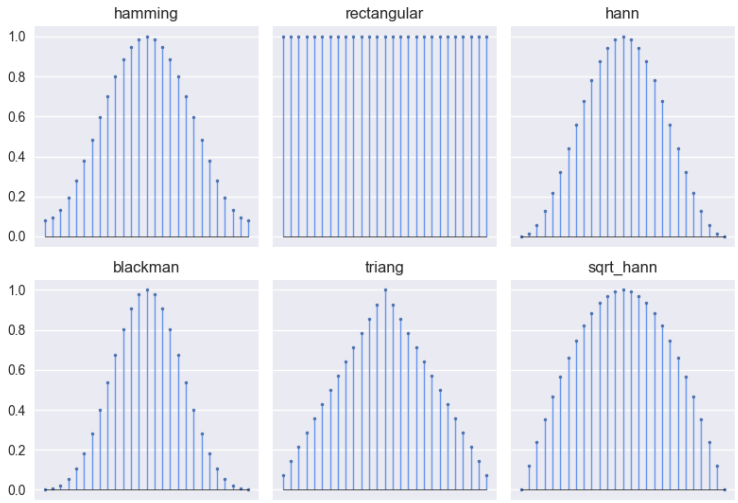

Figura 2.4: Tipos de ventanas comúnmente utilizadas. Imagen extraída de [10].

- Tamaño de la ventana: El tamaño de la ventana determina cuántas muestras se incluyen en cada ventana de tiempo corto. De acuerdo a cómo es calculada la DFT, el tamaño de la ventana, determina la resolución del eje de las frecuencias de la STFT. Si se tiene un tamaño de ventana más grande habrá mayor resolución de frecuencias, y a un tamaño de ventana pequeño, se tendrá una menor resolución de frecuencias. Pero a la vez este parámetro afecta a la resolución del tiempo de forma inversa.

- Tamaño del traslado de ventana: Este parámetro (*Hop* en inglés) determina la distancia en muestras, entre dos ventanas adyacentes, en la figura 2.5 se puede observar como el traslado de ventana puede alargar o acortar el eje del tiempo (afectar la resolución del tiempo) en la STFT dependiendo del tamaño.

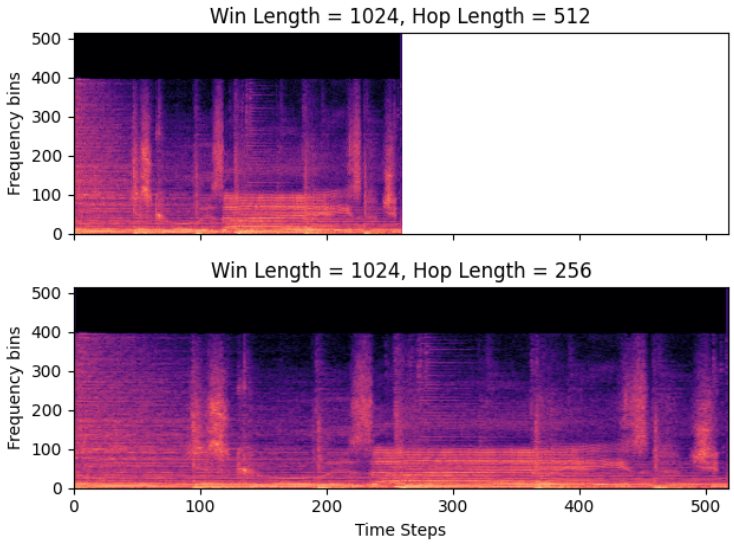

Figura 2.5: Diferentes tamaños de traslado de ventana. Imagen extraída de [10].

- Sobrelape y suma: Muchas de las configuraciones de los parámetros anteriores puede que no reconstruyan perfectamente la señal de STFT a la forma de onda de nuevo. Sin embargo, ciertas configuraciones son matemáticamente perfectas para reconstruir cualquier señal. A estos conjuntos de parámetros se les llama sobrelape y suma constante debido a que al aplicar ventanas sucesivas, se van sumando a un valor constante. En general se tiende a usar un tamaño del traslado de ventana que sea la mitad que el tamaño de la ventana, pero existen varias configuraciones.

La mayoría de los sistemas de separación de fuentes utilizan variantes de los espectrogramas. Algunos tipos de espectrogramas son:

- Espectrograma de magnitud: Este se calcula tomando el valor absoluto de cada elemento de la STFT, $|X| \in \mathbb{R}^{T \times F}$.

- Espectro de potencia: Este se calcula elevando al cuadrado cada elemento de la STFT, $X^2 \in \mathbb{R}^{T \times F}$.

- Espectrograma logarítmico: El humano escucha la amplitud de un sonido de manera logarítmica, no lineal; es decir que la percepción de intensidad se agudiza para sonidos débiles, y disminuye para sonidos fuertes. El espectrograma logarítmico se calcula tomando el logaritmo del valor absoluto de cada elemento de la STFT, $\log |X| \in \mathbb{R}^{T \times F}$.

- Espectrograma logarítmico de potencia: Este se calcula tomando el logaritmo del cuadrado de cada elemento en la STFT, $\log |X|^2 \in \mathbb{R}^{T \times F}$.

- Espectrograma de Mel: El humano también escucha de manera logarítmica las frecuencias; las frecuencias graves no son tan perceptibles como las medias agudas. La escala de Mel aproxima esta propiedad y es una forma de hacer que el eje de las frecuencias sea cuasi-logarítmico. Esto es muy usado para reducir el costo computacional para sistemas basados en aprendizaje profundo, debido a que el número de *bins* utilizando la escala de Mel es menor que cuando se usa la escala lineal. [10] (sección *Representing Audio*).

2.1.3. Enmascaramiento

Las máscaras se utilizan comúnmente con representaciones de tiempo-frecuencia. Una máscara es una matriz del mismo tamaño que un espectrograma y contiene valores en el intervalo $[0.0, 1.0]$. Cada valor en la máscara determina qué proporción de energía de la mezcla original es parte de una fuente. Un valor de 1.0 de un *bin* de tiempo-frecuencia aporta todo el sonido de la mezcla y un valor de 0.0 no aporta sonido de la mezcla. El enmascaramiento no solo se utiliza en separación de fuentes musicales, también es utilizado en general en las ciencias de la computación y aprendizaje de máquina. Si se quisiera separar solamente una fuente, se utilizaría solo una máscara. Y para separar múltiples fuentes, se deben utilizar múltiples máscaras. Para obtener una fuente se multiplica elemento a elemento la matriz de la máscara correspondiente con la matriz del espectrograma de la mezcla. Entonces si la máscara $\hat{M}_i \in [0.0, 1.0]^{T \times F}$, representa la i-ésima fuente S_i, en una mezcla representada por el espectrograma de magnitud $|Y| \in \mathbb{R}^{T \times F}$, se puede estimar la fuente realizando:

$$S_i = \hat{M}_i \odot |Y|.$$

Para una mezcla con N fuentes, la suma elemento a elemento de cada máscara, debería ser igual a una matriz de unos $J \in [1.0]^{T \times F}$, con el mismo tamaño que las máscaras.

$$J = \sum_{i=1}^{N} \hat{M}_i.$$

Esto quiere decir que al multiplicar la combinación de todas las máscaras con la mezcla, se produce la mezcla misma. Lo que se pretende buscar es una máscara que pueda realizar la separación de la mejor manera posible. También existen máscaras binarias las cuales solo toman valores de 0 o 1. Este tipo de máscaras no es muy utilizado para separación de fuentes musicales ya que introducen discontinuidades en el dominio de la frecuencia lo cual resulta en la aparición de artefactos en el resultado final [10] (sección *TF Representations and Masking*).

2.1.4. Fase

La mayoría de los sistemas de separación de fuentes que utilizan la representación de tiempo-frecuencia se centran en la componente de magnitud de la transformada de Fourier, pero otro aspecto importante del sonido es la componente de fase. Dada una señal de audio

$$y(t) = A \sin(2\pi f t + \phi)$$

se puede describir por los parámetros t, A, f y ϕ, donde t es el tiempo en segundos, A es la amplitud, f es la frecuencia en Hz, y ϕ es el desfase en radianes. Si $\phi \neq 0$ entonces la onda sinusoidal se desplaza en el tiempo por $\frac{\phi}{2\pi f}$, teniendo en cuenta que la onda sinusoidal es igual cada $2\pi f$ segundos, es decir:

$$\sin(0) = \sin(2\pi f) = \sin(2k\pi f) \ \forall \ k \in \mathbb{Z}.$$

Esto es inherente a la periodicidad de la función seno, se reinicia cada 0 o $2\pi f$ segundos. De acuerdo con Fourier, cualquier sonido puede ser representado como una suma infinita de senos, cada uno con sus propias amplitudes, frecuencias y desfases. Esto significa que cualquier

sonido puede ser representado con muchos conjuntos de (t, A, f, ϕ). En un espectrograma se involucran los valores de t, A y f, pero no la fase ϕ. La fase es crucial para poder describir una señal de audio.

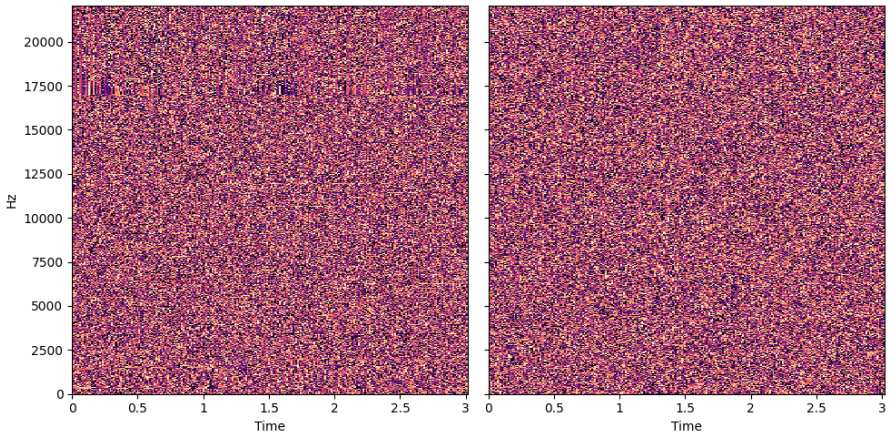

Figura 2.6: Similitudes entre la STFT con componente de fase y ruido aleatorio. Imagen extraída de [10].

En la figura 2.6 se muestran dos imágenes similares, una de ellas representa la componente de fase de una señal de audio, y la otra imagen muestra ruido blanco. El problema de incluir la fase en un espectrograma es que se vuelve complicado la forma en que la DFT captura la señal en cada paso de tiempo, como se captura la frecuencia y la fase. Las frecuencias bajas de la señal cambian más lento que las frecuencias altas. Esto significa que para dos pasos de tiempo adyacentes la diferencia de tiempo es la misma, pero el cambio de cualquier frecuencia puede no ser el mismo. El cambio de fase ocurre más rápido en frecuencias altas que en frecuencias bajas. Otra gran dificultad con la fase es que los humanos no siempre perciben las diferencias de fase. Hay formas de estimar la fase, pero muy pocos sistemas de separación de fuentes logran modelar la fase igual que la magnitud.

Una forma sencilla de manejar la fase es copiar la fase de la mezcla. Esta estrategia no es perfecta, pero investigadores han descubierto que funciona bien. Asumiendo que tenemos una mezcla STFT $Y \in \mathbb{C}^{T \times F}$ y una máscara $\hat{M}_i \in [0.0, 1.0]^{T \times F}$ de una fuente i, se puede aplicar la máscara a la magnitud del espectrograma de Y:

$$\hat{X}_i = \hat{M}_i \odot |Y|$$

donde $\hat{X}_i \in \mathbb{R}^{T \times F}$ representa un espectrograma de magnitud de una fuente (aquí puede variar el tipo de espectrograma). Ahora podemos copiar la fase de la mezcla al espectrograma $\hat{X}_i$ de la siguiente manera:

$$\tilde{X}_i = \hat{X}_i \odot e^{j \cdot \angle Y}$$

donde se usa $j = \sqrt{-1}$, "$\angle$" para representar el ángulo del STFT complejo de Y y $\tilde{X}_i \in \mathbb{C}^{T \times F}$ para indicar que la estimación de la fuente i ahora es compleja. La ecuación final queda de la siguiente forma:

$$\tilde{X}_i = (\hat{M}_i \odot |Y|) \odot e^{j \cdot \angle Y}.$$

Es posible estimar la fase aplicando la máscara estimada al espectrograma de la mezcla. Una forma es utilizar el algoritmo de Griffin-Lim [19], el cual intenta reconstruir la componente de fase de un espectrograma calculando iterativamente la STFT y su inversa. Usualmente este algoritmo converge entre 50 y 100 iteraciones, sin embargo se pueden generar artefactos en el audio y se han desarrollado métodos más rápidos. La inversión de espectrograma de entrada múltiple es una variante del algoritmo de Griffin-Lim, diseñada específicamente para separación de múltiples fuentes. Esta variante añade una restricción al algoritmo original de tal forma que todas las fuentes estimadas con sus componentes de fase estimados deben sumarse a la mezcla de entrada. [20].

Cabe destacar que el cálculo de la STFT, la ISTFT y los algoritmos de estimación de fase son diferenciables. Por lo que se pueden incorporar como parte de arquitecturas de redes neuronales para entrenar directamente.

Una forma reciente que investigadores han tomado para resolver el problema de la fase es eludirlo totalmente. Muchos modelos de aprendizaje profundo recientes se han propuesto como *end-to-end*, es decir que la entrada de los modelos son formas de onda. En estos casos, el modelo decide cómo manejar la fase. Este método puede no ser el más efectivo o eficiente, sin embargo se están investigando tácticas para reducir inconvenientes. [10] (sección *Phase*).

2.1.5. Evaluación

Existen dos formas de evaluar qué tan bien un sistema de separación de fuentes musicales realiza la separación, la manera objetiva y la subjetiva. La manera objetiva, mide la calidad de

la separación realizando cálculos que comparan las señales de salida de las separaciones contra las fuentes aisladas reales (*ground truth*). La manera subjetiva involucra personas que puedan calificar las salidas de los sistemas de separación de fuentes. La manera objetiva tiene la imperfección de que existen muchos aspectos de percepción humana que son extremadamente difíciles de capturar computacionalmente; pero es mucho más rápido y barato que la manera subjetiva. A pesar de que la manera subjetiva consume mucho tiempo y es costosa, puede ser más fidedigna ya que se involucra el oído humano real para evaluar. La forma más utilizada es la objetiva.

Las mediciones objetivas son las siguientes: Relación Señal-Distorsión (SDR por sus siglas en inglés), Relación Señal-Interferencia (SIR por sus siglas en inglés), y Relación Señal-Artefactos (SAR por sus siglas en inglés). Estas mediciones son las más utilizadas hoy en día para evaluar las salidas de un sistema de separación de fuentes.

Una fuente estimada $\hat{s}_i$ se puede descomponer en:

$$\hat{s}_i = s_{\text{target}} + e_{\text{interf}} + e_{\text{noise}} + e_{\text{artif}},$$

donde s_{target} es la fuente real y e_{interf}, e_{noise}, y e_{artif} son errores por interferencia, ruido y artefactos respectivamente. [21].

Todas las mediciones de estos valores están en decibeles (dB), mientras el valor sea mayor, mejor será el desempeño. Para calcularlos se requieren las fuentes aisladas reales (*ground truth*). Las fórmulas para calcular estos valores son:

Relación Señal-Distorsión

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

Relación Señal-Interferencia

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

Relación Señal-Artefactos

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

Existe también la Relación Señal-Ruido (SNR por sus siglas en inglés), la cual no es tan utilizada como los anteriores; la fórmula para calcularlo es la siguiente:

$$\text{SNR} := 10 \log_{10} \left(\frac{\|s_{\text{target}} + e_{\text{interf}}\|^2}{\|e_{\text{noise}}\|^2} \right)$$

Desde la publicación de [21], al pasar de los años, se han observado varios errores de implementación; uno de ellos es que la SDR es "fácil de engañar". La manera en que la SDR calcula los errores de interferencia, ruido y artefactos puede causar problemas.

En los artículos de separación de fuentes, normalmente se reportan las SDRs y solo se reporta un valor. Usualmente este número es el promedio de una distribución de SDRs calculadas de una base de datos. Esta práctica no es la mejor ya que es una estadística resumida que podría oscurecer toda una distribución.

En [10] se muestra una comparación de dos modelos de separación de fuentes musicales, Conv-TasNet [22] y Open-Unmix [23]. Al oído humano, Open-Unmix suena mucho mejor que Conv-TasNet, pero ambos presentan el mismo SDR.

La manera subjetiva de evaluación requiere de humanos para evaluar la calidad de las separaciones. Idealmente se ocuparían ingenieros de audio con oídos bien entrenados y un cuarto con tratamiento acústico para realizar esta tarea. [10] (sección *Evaluation*).

Es importante aclarar que, aunque se decidió utilizar la métrica de SIR en este trabajo por su amplia aplicación en el tema de separación de fuentes, no es la única que se pudiera haber utilizado. Por ejemplo, encontrar la distancia vectorial entre los espectros de señales en el dominio de la frecuencia o de Wavelets son otras alternativas. Pero, se optó por el SIR para facilitar la comparación de los resultados obtenidos en este trabajo ante otros trabajos en la literatura, como es presentado en las gráficas del Capítulo 4.

2.1.6. Algoritmos tradicionales de separación de fuentes musicales

Antes de hablar de la separación de fuentes musicales a través de aprendizaje profundo, es bueno conocer de algunos otros de los algoritmos de separación de fuentes musicales.

En muchos géneros musicales, una característica común es la repetición. Muchas veces se puede utilizar esta cualidad para identificar diferentes fuentes en una canción. Por ejemplo una guitarra eléctrica puede improvisar un solo sobre una base musical repetitiva o una banda de acompañamiento. En este caso podemos aprovechar la repetición de la banda para separar la guitarra eléctrica.

Con este concepto; existen tres algoritmos que intentan realizar la separación de un fondo repetitivo de un plano no repetitivo. Para ello se asume que:

1. Hay una repetición en la mezcla.

2. La repetición contiene lo que queremos separar.

Estas presuposiciones funcionan bien si se quiere separar una trompeta de una banda de fondo por ejemplo; pero no funcionaría bien si se quiere separar una batería del resto de la banda porque la batería es la que lleva casi todo el patrón repetitivo. Los tres algoritmos que utilizan el concepto de repetición trabajan con los espectrogramas de magnitud de las mezclas, intentan encontrar las partes repetitivas de la mezcla y las separan creando una máscara para la fuente y los instrumentos de fondo.

El primer algoritmo se llama Técnica de extracción de patrones repetitivos o en inglés *REpeating Pattern Extraction Technique* (REPET) [24], funciona de la siguiente manera:

1. Encuentra un periodo repetitivo, t_r segundos (por ejemplo el tiempo en el que una progresión de acordes se repite).

2. Segmenta el espectrograma en N segmentos, cada uno con t_r segundos de longitud.

3. "Sobrepone" los N segmentos.

4. Toma la mediana de los N segmentos apilados y crea una máscara de los valores de la mediana.

Otro algoritmo que aprovecha la repetición en la música es REPET-SIM [25], el cual es una variante de REPET que no depende de un periodo de repetición fijo. REPET-SIM calcula una matriz de similitud entre cada par de cuadros espectrales en una STFT, selecciona los k vecinos más cercanos y crea una máscara mediante un filtro de medianas de los *bins* para cada uno de los vecinos seleccionados.

Existe otra variante la cual utiliza una transformada de Fourier de dos dimensiones (2DFT) de un espectrograma para encontrar patrones repetitivos y no repetitivos [26]. Las secciones repetitivas se muestran como picos en la 2DFT y las no repetitivas están en todo lo demás. Se pueden escoger los picos para separar la parte repetitiva de la no repetitiva.

Existe un algoritmo más para separación de fuentes musicales. Este algoritmo no utiliza repetición y se llama Separación de fuente armónica-percusiva o en inglés *Harmonic-Percussive Source Separation* (HPSS). Si uno pasa mucho tiempo visualizando espectrogramas de señales musicales, se dará cuenta que los sonidos armónicos se ven como rayas horizontales y sonidos percusivos se ven como rayas verticales. La idea del HPSS es que se puede aplicar filtros de medianas a través de los *bins* de frecuencia (horizontalmente o armónicamente) y a través de los *bins* de tiempo (verticalmente o percusivamente) para separar las fuentes. [10] (sección *Build your own HPSS*).

2.2. Aprendizaje profundo para separación de fuentes

El aprendizaje profundo ha sido ampliamente utilizado en el estado del arte para separación de fuentes [1] [3] [5] [9] [22] [23] [27] [28] [29] [30] [31] [32] [33] [34] entre otros. En palabras simples, trabajan entrenando con grandes cantidades de datos de mezclas y de fuentes separadas. El concepto de aprendizaje profundo normalmente se implementa por medio de una serie de capas conformadas, a su vez, por pesos interconectados. El conjunto completo de pesos conforma una red que actúa como una gran función matemática: ésta produce una salida a partir de una entrada. Durante el entrenamiento de la red, ésta produce una salida para una fuente, luego esa salida se compara con fuente real (*ground truth*). Esta comparación se usa para actualizar la red, de tal forma que cuando produzca una estimación de la misma fuente de nuevo, ésta puede ser más cercana a la fuente real. Este proceso se llama

retro-propagación [10]. En la figura 2.7 se muestra un diagrama del entrenamiento por retro-propagación; donde *Estimates* es lo que produce la red y *Stems* son las fuentes reales. El proceso de entrenamiento requiere tener acceso a las fuentes separadas reales (*ground truth*). Esto quiere decir que la mayoría de los sistemas de separación de fuentes son sistemas de aprendizaje supervisado. Para obtener el gran desempeño que las redes profundas tienen, se requiere de una gran cantidad de datos para el entrenamiento.

Las redes neuronales son sistemas muy complejos con millones de parámetros entrenables llamados pesos. El cómo configurar los pesos (escoger el número de pesos, la estructura de las conexiones, cómo se entrenan, cómo se actualizan, etc.) es un proceso difícil. Cada una de estas configuraciones se llaman hiperparámetros. La configuración de solo un hiperparámetro puede hacer la diferencia de tener buenos o malos resultados.

Las redes neuronales están compuestas de capas las cuales, cada una tiene cierta cantidad de pesos que se actualizan con el descenso por gradiente. En el entrenamiento se tienen dos fases: el recorrido de la red hacia adelante y el recorrido hacia atrás o retro-propagación.

En el recorrido hacia adelante cada capa recibe como entrada la salida de la capa anterior y transforma los datos multiplicando cada componente de la entrada por unos pesos, el resultado de la multiplicación es la salida. En realidad, la forma que las entradas entran y las salidas se calculan es un poco más complicado, pero es importante saber que la entrada pasa a través de una red, pasando a través de muchas transformaciones discretas, las cuales cada una es una capa. En la fase de retro-propagación se regresa el error de estimación hacia atrás

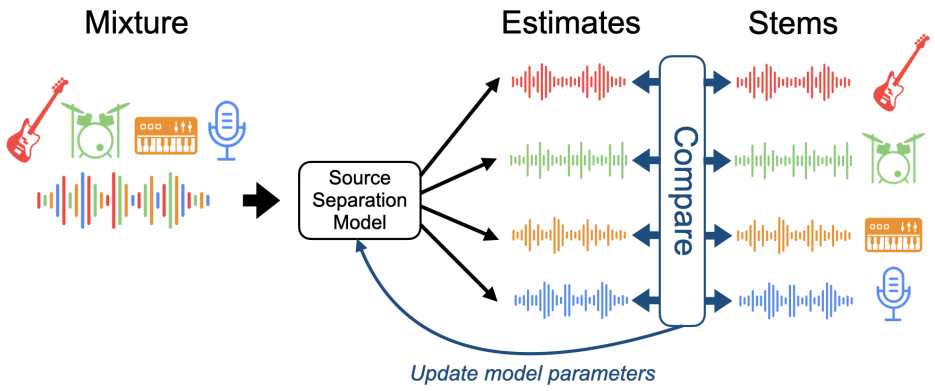

Figura 2.7: Diagrama del entrenamiento para separación de fuentes musicales. Imagen extraída de [10].

desde la capa de salida a la capa de entrada ajustando algunos hiperparámetros y pesos.

Casi todas las redes profundas para separación de fuentes musicales se entrenan con una variante del descenso por gradiente estocástico; el cual pasa los datos a través de cada capa en mini lotes. [10] (sección *Deep learning approaches*).

2.2.1. Tipos de capas

Capas completamente conectadas: En una capa completamente conectada, cada nodo (neurona) se conecta con todos los nodos de la siguiente capa; esto se ilustra en la figura 2.8. Estas capas son normalmente llamadas capas lineales, ya que, sin una función de activación (ver 2.2.2), aplican una transformación lineal a los datos de entrada. Las capas completamente conectadas tienen varios usos, pero comúnmente se usan para expandir o comprimir la dimensionalidad de la entrada a la siguiente capa.

En la separación de fuentes las capas completamente conectadas se utilizan usualmente para crear máscaras, cambiando la dimensionalidad de la capa anterior para que coincida con la dimensionalidad de la salida. Las capas completamente conectadas comúnmente tienen una función de activación cuando se usan como máscara. Hay muchas formas en las que se puede aprovechar la propiedad de poder cambiar la dimensionalidad. Un ejemplo es: digamos que tenemos un sistema de separación de fuentes musicales que aplica una máscara a un espectrograma que tiene 513 componentes de frecuencia y 400 pasos de tiempo. Incluyendo la dimensión del lote, la forma (*shape*) de la salida para nuestra capa completamente conectada es: [16, 400, 513], donde la primera dimensión es el lote (16 ejemplos). Si queremos que este sistema haga dos máscaras podemos cambiar la dimensionalidad de esta capa para que quede [16, 400, 1026], donde ahora tenemos el doble de la dimensión de la frecuencia ($513 \times 2 = 1026$) para indicar que tenemos dos fuentes. Luego podemos cambiar la forma de la salida de tal forma que sea [16, 400, 513, 2] para tener en cuenta ambas fuentes. La red aprenderá que, esto significa que hay dos fuentes.

Capas convolucionales: Las capas convolucionales son similares a las completamente conectadas, excepto que cada nodo está conectado a solo algunos nodos de la capa anterior. Reducir la cantidad de conexiones hace que la red sea menos propensa a sobre-ajustarse. Las capas convolucionales se relacionan con el concepto de convolución en el sentido de que las

capas convolucionales aprenden un conjunto de filtros utilizando una ventana deslizante en la capa de entrada. Esta ventana que se desliza es el campo receptivo de la capa convolucional.

Las formas de salida de las convoluciones pueden cambiar drásticamente dependiendo de cómo se configuran las capas. Existen cuatro parámetros principales que afectan en este aspecto: tamaño del *kernel*, el paso (*stride*), relleno (*padding*), y dilación (*dilation*). El tamaño del *kernel* determina el número y forma de los nodos de la capa anterior que los nodos de la capa actual ven (en palabras más simples es la forma de la ventana), y el *stride* determina la distancia que la ventana se irá moviendo entre nodos de entrada adyacentes. El *padding* determina qué se hace en los bordes de la entrada; si no hay *padding*, las entradas solo se cubren hasta los bordes por la capa convolucional; si hay *padding* se agregan valores (comúnmente ceros) a los bordes de la entrada así la capa convolucional abarca más de los bordes de la entrada. La *dilation* determina el espaciado entre los nodos de la entrada que cada nodo convolucional ve. Esto permite entender más contexto que si no se usara *dilation*. En WaveNet [27] se utilizaron convoluciones con *dilation* de manera autoregresiva para producir una muestra de una forma de onda a la vez. Normalmente la salida de las capas convolucionales tienen menor dimensionalidad que la capa de entrada, sin embargo a veces es requerido tener una mayor dimensionalidad que la entrada.

Una parte importante al usar redes convolucionales es el *pooling*, o reducir la dimensionalidad de una capa convolucional usando alguna función no lineal como el máximo o el promedio. El *pooling* secciona la entrada en varias regiones (no sobrepuestas) y realiza la reducción en cada región. El *pooling*, técnicamente es una capa a parte, pero casi siempre se encuentra después de una capa convolucional. El tipo de *pooling* más común es el *max pooling*, pero existen otros tipos como el *average pooling*.

En separación de fuentes las convoluciones han tenido buena efectividad en el dominio del tiempo (forma de onda) y de tiempo-frecuencia. Para formas de onda se utilizan convoluciones de una dimensión y para tiempo-frecuencia de dos dimensiones. Cuando se dice que las convoluciones aprendieron un grupo de filtros (o un banco de filtros), se relaciona con el concepto de filtros en audio, como pasa altas, pasa bajas, o pasa banda.

Las capas convolucionales necesitan tener ejemplos completos con la forma de entrada exacta para poder procesar los datos, por ejemplo, si tenemos un espectrograma con 512 *bins*

de frecuencia y 1000 pasos de tiempo, pero nuestra primera capa convolucional requiere una forma de entrada de [512, 128], se debe dividir el espectrograma en 8 ventanas de 128 de tamaño e incluir el *padding* necesario en la última ventana. Las capas convolucionales pueden tener dificultades con efectos en los bordes. Por ejemplo, cuando se predice una forma de onda es posible que una red neuronal convolucional aprenda discontinuidades, lo cual podría producir artefactos audibles. Una forma de evitar esto es generar ventanas sobrepuestas, similar al sobrelape y suma. Utilizando el ejemplo anterior, se debería dividir la entrada en 16 ventanas de la misma longitud pero con sobrelape una con otra.

Capas recurrentes: Las capas recurrentes tienen conexiones tanto consigo mismas como las que le siguen. En otras palabras, algunas salidas de los nodos se conectan a las entradas de los mismos nodos, creando ciclos para que la información pueda persistir. En la figura 2.9 se puede ver una red recurrente (RNN por sus siglas en inglés). Gracias a los ciclos formados, las redes recurrentes son adecuadas para datos que varían con el tiempo, como el audio. La forma en que las capas recurrentes se usan en separación de fuentes musicales es que toman el audio a medida que cambia a lo largo del tiempo. Por ejemplo, si tenemos un espectrograma como entrada a una capa recurrente, esta tomará una columna del espectrograma por cada paso de tiempo.

El número de unidades que tienen las capas recurrentes define su campo receptivo o la cantidad de pasos de tiempo que la capa puede ver en un momento dado. Por ejemplo, si un espectrograma tiene 1000 pasos de tiempo pero la capa recurrente solo tiene 400 unidades, solo 400 pasos de tiempo se procesarán por cada capa recurrente en un tiempo. Empezará con el primer paso de tiempo y trabajará a través de cada uno de los 1000 pasos de tiempo avanzando un paso a la vez. Debido a esto, si entrenas con una capa recurrente que tiene solo 400 pasos de tiempo, puede aceptar espectrogramas de cualquier longitud.

Los dos tipos más comunes de RNN's son *Long Short-Term Memory* (LSTM) y *Gated Recurrent Unit* (GRU). Estas son capaces de retener información de manera estable; mientras que las RNN's simples (como en la figura 2.9) no retiene la información de manera estable. En el estado del arte se utiliza más las LSTM's que las GRU's.

Es común ver capas recurrentes bidireccionales, lo cual significa que tienen dos grupos de unidades: una que avanza hacia adelante en el tiempo y otra que avanza hacia atrás en el

tiempo. Utilizándolo de manera bidireccional se puede "ver el futuro", ya que inicia al final de un clip de audio. Dado esto, no es recomendable usar capas recurrentes bidireccionales si se requiere un sistema en línea ("en tiempo real" como incorrectamente se dice). Si una LSTM es bidireccional se abrevia BLSTM, y si una GRU es bidireccional se abrevia BGRU. [10] (sección *Deep learning approaches - Building blocks*).

2.2.2. Funciones de activación

Las funciones de activación cambian qué tanto una capa afecta a la siguiente. La función de activación "decide" si un nodo debe "prenderse" o "apagarse". Las funciones de activación normalmente son no lineales y siempre diferenciables. Las más utilizadas en separación de fuentes musicales son:

- Sigmoide: Esta es usada comúnmente como la salida de una red neuronal que crea máscaras. Dado que está limitada en el rango de $[0.0, 1.0]$ se puede usar para crear *Soft Masks*. Esta función se denota con σ. En la figura 2.10 se muestra la gráfica de esta función. Esta figura fue creada con Python al igual que las figuras 2.11 y 2.12

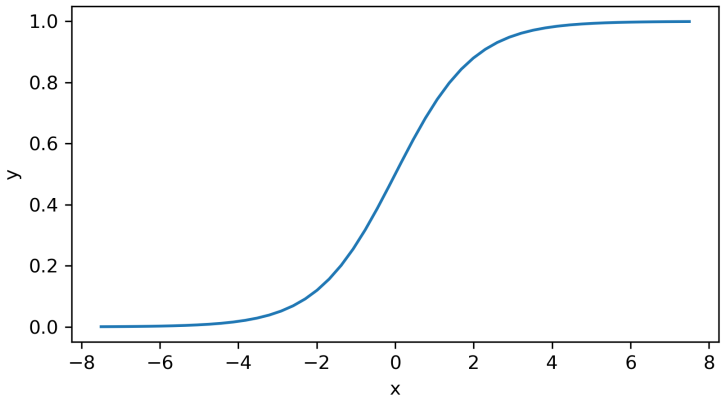

Figura 2.10: Gráfica de la función de activación Sigmoide.

- Tangencial hiperbólica (Tanh): Esta es similar a la Sigmoide pero con la diferencia que se delimita en el rango de $[-1.0, 1.0]$. Por esta razón es más común encontrarla a la mitad de una red. En la figura 2.11 se muestra la gráfica de esta función.

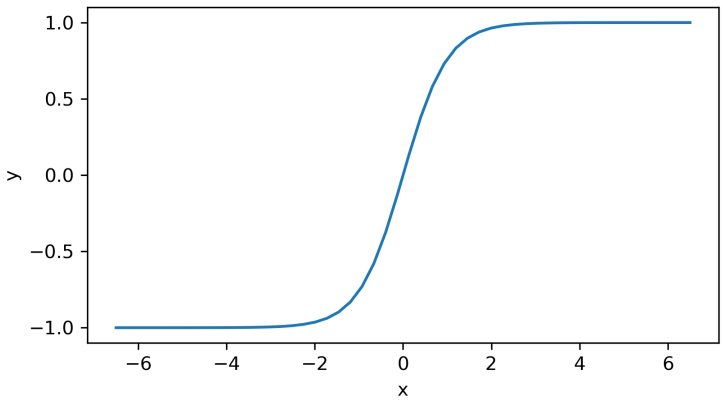

Figura 2.11: Gráfica de la función de activación tangencial hiperbólica.

- Unidad lineal rectificada (ReLU): Con esta función la salida es 0.0 si la entrada es menor que 0.0, sino, se tiene una salida lineal con pendiente de 1. La ReLU se usa a veces para crear máscaras, o también formas de onda. En la figura 2.12 se muestra la gráfica de esta función. Existen dos variantes que son la *Leaky* ReLU y la PReLU. La *Leaky* ReLu, tiene un comportamiento diferente si la entrada es menor que 0.0, se multiplica la entrada por 0.01. La PReLU tiene un parámetro entrenable (α) que define la pendiente cuando la entrada es menor que 0.0. Estas funciones de activación se utilizan en las capas medias o como activación para la capa final cuando en la salida se genera una forma de onda.

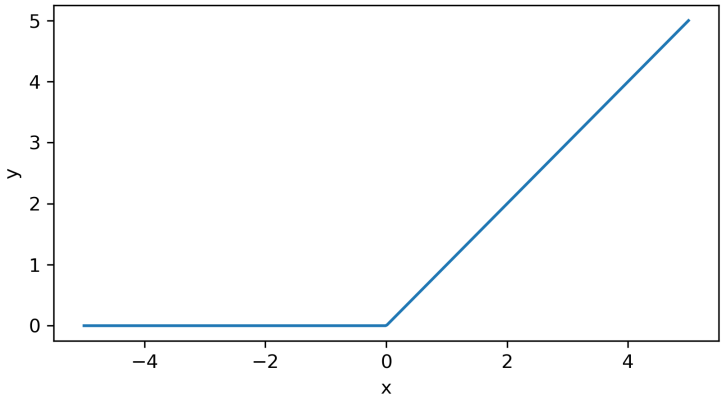

Figura 2.12: Gráfica de la función de activación ReLU.

2.2.3. Normalización

La normalización asegura que las entradas a una red o a una capa sean iguales desde el punto de vista estadístico. Prácticamente esto significa que todos los datos deben tener el mismo promedio y desviación estándar. Cambiar la media desplaza los datos, literalmente significa sumar o restar la media calculada a cada dato específico; y cambiar la desviación estándar escala los datos, literalmente significa dividir cada punto de datos por la desviación estándar calculada. Este proceso hace que el entrenamiento de las redes sea más estable. Existen algunos métodos para realizar la normalización. De acuerdo con [10] algunos de los métodos más comunes de normalización en separación de fuentes son:

- Normalización por lotes: Este método calcula el promedio y la desviación estándar de cada mini lote durante el entrenamiento y normaliza los datos utilizando esas estadísticas. Como en el *pooling*, la normalización por lotes se considera una capa más en una red neuronal y se puede encontrar en varias partes de la arquitectura, incluyendo en la capa de entrada.

- Blanqueado de datos: Algunos investigadores normalizan todo su base de datos como una etapa de pre-procesamiento. A esto se le llama blanquear los datos. El concepto es el mismo, el promedio y la desviación estándar de toda la base de datos se usa para desplazar y escalar los datos. Esto se realiza antes de entrenar la red. Este método es popular en separación del habla, pero no es muy práctico en separación de fuentes musicales.

- Normalización aprendible: Este método es hacer que la red aprenda a normalizar sus entradas por sí misma. En este caso, el sistema tiene dos parámetros a aprender para el desplazado y escalado. Estos parámetros se optimizan usando el descenso del gradiente, igual que para los pesos de la red. Cuando se aplica como una función de desplazado y escalado a las entradas de la red, ésta las usa para normalizar sus datos de entrada con base en lo que se determine mejor. Open-Unmix [23] utiliza parámetros de normalización aprendible.

- Otras técnicas de normalización: Existen también muchas otras técnicas de normaliza-

ción que se han desarrollado; algunas se han utilizado en separación de fuentes, como la normalización de instancia.

2.2.4. *Dropout*

El *Dropout* es una técnica de regularización la cual ayuda a la habilidad de la red de poder generalizar a datos no vistos. Esta es una técnica simple mediante la cual, en cada paso de entrenamiento, un porcentaje de los nodos se establece en 0. Esta técnica se utiliza mucho en sistemas de separación de fuentes.

A veces es deseable representar la información de frecuencias haciendo que la red ingrese un espectrograma en vez de utilizar una forma de onda. Los espectrogramas se calculan dada la forma de onda que se introduce a la red, y se pueden calcular en una etapa de pre-procesamiento, con todo el espectrograma guardado como caché en el disco para ser cargado durante el entrenamiento; o también los espectrogramas pueden calcularse sobre la marcha cuando la red lo necesite. Si se hace como pre-procesamiento, es usualmente más rápido pero requiere un paso por separado lo cual toma un poco de tiempo y espacio en disco para guardar los datos. Por otra parte, calcularlo sobre la marcha requiere más cálculos por cada ejemplo y puede llegar a haber un cuello de botella en el entrenamiento si no se hace eficientemente.

Muchas veces cobra sentido disminuir el tamaño del espectrograma. Esto se debe a que, al hacerlo nos permite tener redes más grandes, y se tiene mayor capacidad de aprender mejor. Una forma en la que podemos disminuir el tamaño del espectrograma y aún preservar algunas características relevantes es convertirla de escala lineal a escala de Mel. El cálculo del espectrograma es completamente diferenciable, lo que significa que podemos incorporar cálculos de STFT estáticos y no aprendibles en nuestra red para hacer modelos forma de onda a forma de onda, si así lo deseamos. Esto es útil si queremos utilizar un modelo de espectrogramas con alguna función de pérdida de formas de onda; pero puede alentar el proceso de entrenamiento notablemente. [10] (sección *Deep learning approaches - Building blocks*).

2.2.5. Funciones de pérdida

La pérdida es lo que se usa para determinar la distancia entre las estimaciones de la red y las fuentes reales. Se utiliza entonces para actualizar los parámetros de la red.

Pérdidas para espectrogramas: Cuando se calculan las pérdidas con espectrogramas, se compara el espectrograma de la fuente real con el espectrograma que tiene aplicada la máscara de la red. Para una fuente i, dada la separación real (*ground truth*) de una STFT $S_i \in \mathbb{C}^{F \times T}$, una mezcla de entrada $Y \in \mathbb{C}^{F \times T}$, y una máscara estimada $\hat{M}_i \in \mathbb{R}^{F \times T}$, podemos calcular la pérdida de la siguiente manera:

$$\mathcal{L}_{\text{spec}} = \left|\left||S_i| - \hat{M}_i \odot |Y|\right|\right|_p,$$

donde $\odot$ es el producto elemento a elemento y p es la norma del valor de pérdida. En separación de fuentes solo dos normas se utilizan ampliamente, la norma L1 donde $p = 1$ y L2 o norma euclidiana donde $p = 2$. La norma L2 es comúnmente llamada como error cuadrático medio (MSE por sus siglas en inglés). [10] (sección *Deep learning approaches - Building blocks*).

Pérdidas para agrupamiento profundo: Estas pérdidas usualmente se utilizan con espectrogramas como entrada [35]. La meta es aprender un espacio de *embeddings* de alta dimensionalidad donde todos los *bins* de tiempo-frecuencia dominados por la misma fuente estén juntos y los dominados por diferentes fuentes estén muy separados.

Para un canal, la máscara binaria de una fuente real (*ground truth*) $Y \in \{0.0, 1.0\}^{T \times F}$, la cambiamos para que tenga la forma $TF \times 1$. Luego se aprende un espacio de *embeddings* de dimensión D llamado $V \in \mathbb{R}^{TF \times D}$. Definimos una matriz afinidad binaria real $A = YY^T$, y una matriz de afinidad estimada de la red $\hat{A} = VV^T$ [36]. La pérdida de agrupamiento profundo está dada por:

$$\mathcal{L}_{DC} = ||\hat{A} - A||_F^2 = ||VV^T - YY^T||_F^2$$

donde F es la norma Frobenius. También se han propuesto otros tipos de pérdidas de agrupamiento profundo, pero la intuición es la misma: poner los *bins* de tiempo-frecuencia de la

misma fuente cerca una de otra y lejos de las de otras fuentes. Una vez que la red es entrenada, un algoritmo de agrupamiento como *K-means* se usa para crear máscaras para cada fuente. Este método es usado frecuentemente como regularizador para ayudar a aprender una máscara directamente.

Pérdidas de formas de onda: La forma simple de calcular la pérdida para el dominio de la forma de onda es tomar la pérdida L1 o el MSE entre la forma de onda real y la estimada, similar a como se hace con los espectrogramas:

$$\mathcal{L}_{\text{wvfm}} = \left\| \hat{x}_i - x_i \right\|_p = \frac{1}{T} \sum_{t=1}^{\infty} \left\| \hat{x}_{i,t} - x_{i,t} \right\|_p$$

donde $\hat{x}_i$ es la forma de onda estimada para la fuente i, x_i es la forma de onda real (*ground truth*) de dicha fuente, T es el número total de muestras en la fuente y p es la norma (1 para L1, 2 para MSE). [3].

Alternativamente podemos emular las métricas de evaluación SIR, SDR y SAR como funciones de pérdida. [28].

2.2.6. Optimizadores

Los optimizadores son los que realizan el descenso del gradiente para actualizar los pesos de un modelo. Mayormente se utiliza el optimizador Adam, pero existen muchos otros. Algunos hiperparámetros importantes en este aspecto son la tasa de aprendizaje y el *gradient clipping*. [10] (sección *Deep learning approaches - Building blocks*).

2.2.7. Tasa de aprendizaje

La tasa de aprendizaje es un hiperparámetro clave para el descenso del gradiente. Controla que tanto se avanza (o se aleja) hacia el gradiente óptimo. Si la tasa de aprendizaje es muy alta, entonces nunca se llegará a la solución óptima. Si se escoge una tasa de aprendizaje muy pequeña se tardará mucho en llegar al punto óptimo. [10] (sección *Training - Gradient descent*). Existen técnicas de optimización de la tasa de aprendizaje durante el entrenamiento; como por ejemplo: reducirla si en cierta cantidad de épocas no hay mejora de la pérdida

en la etapa de validación.

2.2.8. *Gradient clipping*

En muchos entrenamientos, al visualizar la trayectoria de la pérdida, se observa que se puede tener ruido o saltos e imperfecciones. Estas imperfecciones pueden producir picos en el gradiente, lo cual puede desestabilizar el proceso de aprendizaje. Esto se puede atenuar con el *Gradient clipping*. En pocas palabras consiste en recortar los gradientes si este excede cierto umbral, luego se renormaliza de tal forma que la norma del gradiente sea igual al umbral. Si la norma esta debajo del umbral, el gradiente no se modifica. [10] (sección *Training - Gradient descent*).

2.2.9. Tamaño del lote

El tamaño del lote es la cantidad de ejemplos de entrenamiento que se utilizan para calcular los gradientes. Comúnmente es mejor tener un mayor tamaño del lote dependiendo de la memoria disponible, ya sea de la GPU o la memoria RAM si se entrena en CPU. Y es recomendable utilizar un número que sea potencia de 2. [10] (sección *Deep learning approaches - Building blocks*).

2.2.10. Épocas de entrenamiento

Este es el número de veces que se realiza un ciclo de entrenamiento a través de todos los datos. También existe un hiperparámetro útil en este aspecto, el paro temprano. Este hiperparámetro hace que el entrenamiento se detenga cuando la pérdida en la validación deja de disminuir (esto es un signo que el modelo está sobre-ajustándose). [10] (sección *Deep learning approaches - Building blocks*).

2.2.11. Algunas bases de datos usadas para la separación de fuentes musicales

Las siguientes bases de datos son útiles para entrenar o evaluar modelos de separación de fuentes musicales.

BASS-dB: BASS-dB [37] es una pequeña base de datos que contiene grabaciones de música multipista para la evaluación de separación de fuentes musicales. Algunos de sus datos se utilizaron como material de evaluación en la primera campaña de evaluación de separación de fuentes de audio en estéreo [38]. BASS-dB se compone de 20 canciones con instrumentos variados para cada una de sus canciones; como batería, bajo, voces, guitarras eléctricas, guitarra acústica, piano, teclados o sintetizadores y otros. No en todas sus canciones se encuentran todos los instrumentos mencionados. Todas las canciones incluyen las pistas separadas, así como la mezcla de la canción en formato MP3.

MASS: *Music Audio Signal Separation* [39] es una base de datos aún más pequeña que BASS-dB, la cual se creó para evaluar algoritmos de separación de señales musicales. Se compone de 2 pares de fragmentos de 10 a 40 segundos de 6 canciones, con sus mezclas e instrumentos por separado. Los audios se presentan en formato WAV a 44.1Khz y 24 bits; adicionalmente las mezclas se presentan en formato MP3. Algunas de las canciones incluyen las mezclas y pistas sin efectos agregados, y algunas de las canciones se presentan solamente sin efectos. Algunos instrumentos que se pueden escuchar en estas canciones son: batería, bajo, voces, guitarras eléctricas, guitarra acústica, piano, teclados, harmónica entre otros. No en todas las canciones se encuentran todos los instrumentos mencionados y en las canciones con efectos se proporciona una descripción de los efectos utilizados.

MedleyDB: MedleyDB [40] es una base de datos de 122 canciones multipista, la cual fue creada primordialmente para hacer investigación en extracción de melodía, pero también es útil para reconocimiento automático de instrumentos, mezclado automático y separación de fuentes musicales. En toda la base de datos se puede encontrar diferentes instrumentos musicales como: batería, bajo, piano, voces, sintetizadores, guitarras eléctricas, guitarras acústicas, percusiones, violines, cellos, entre otros. De las 122 canciones, 108 contienen anotaciones de la melodía y activación de instrumentos. Un par de años más tarde salió MedleyDB

2.0 [41] la cual mejora la calidad de los archivos de audio y agrega 132 nuevas canciones.

DSD100: DSD100 [42] es una base de datos de 100 canciones de diferentes estilos con sus respectivas pistas de batería, bajo, voces y otros; donde otros son el resto de instrumentos diferentes de la batería, bajo y voces. Los conjuntos se dividen en 50 canciones para entrenamiento y 50 para prueba. Todas las canciones son en formato WAV estéreo a 44.1 kHz.

MUSDB18: MUSDB18 [43] es una base de datos de 150 canciones de diferentes estilos con sus respectivas pistas de batería, bajo, voces y otros igual que DSD100. Esta base de datos es la que más se utiliza en el estado del arte para separación de fuentes musicales. Los conjuntos se dividen en 100 canciones para entrenamiento y 50 para prueba. MUSDB18 esta conformado por las 100 canciones de DSD100, más 46 canciones tomadas de MedleyDB, más 2 canciones proporcionadas por *Native Instruments* (empresa que fabrica software y hardware para crear música) y 2 canciones de una banda llamada *The Easton Ellises*. Los archivos de MUSDB18 estan condificados en el formato de *Native Instruments* MP4 el cual es un formato multipista compuesto de 5 señales estéreo codificadas en AAC a 256 kbps. Las 5 señales corresponden a la mezcla, la batería, el bajo, el resto del acompañamiento (otros) y las voces. Ahora bien, existe también MUSDB18-HQ [44] que son las mismas canciones pero en formato WAV sin compresión a 44.1 kHz pudiendo tener el ancho de banda audible hasta poco más de 22 kHz; diferente de MUSDB18 normal cuyo ancho de banda audible es hasta 16 kHz.

Slakh2100: *Synthesized Lakh* (Slakh) [45] es una base de datos de 2100 canciones multipista para realizar investigación en separación de fuentes musicales y transcripción automática de instrumentos. Está generada a partir de la base de datos MIDI Lakh (LMD, también se utilizaron algunos de estos MIDI's para la base de datos de este proyecto). Slakh2100 no contiene pistas de voces, es meramente instrumental debido a que está hecho a base de MIDI; entre los instrumentos que hay en las diferentes canciones se encuentran: piano, bajo, batería, guitarra eléctrica y acústica, ensambles de cuerdas, sintetizador, ensambles de metales, órgano, percusiones, entre otros; siendo el piano, el bajo, la batería y la guitarra los instrumentos con más ocurrencia en las canciones. Slakh2100 es la base de datos para separación de fuentes musicales más grande que existe actualmente. Los archivos están separados

en 1500 canciones para el entrenamiento, 375 para validación y 225 para prueba. Todos los archivos de audio de Slakh2100 están en formato mono a una calidad de 44.1 kHz y 16 bits.

Base de datos interna de Deezer: En [5] se menciona que los datos que se utilizaron para los entrenamientos, fueron bases de datos internas de Deezer. Deezer es un servicio de *streaming* de música. Lamentablemente no encontré algún artículo que mencione las características de esta base de datos privada. En [46] se menciona que la base de datos es de aproximadamente 200 horas de audio, se intuye que la diversidad de las canciones es variada y son canciones reales. En esta base de datos se intuye que contiene las pistas de: la batería, el bajo, las voces, el piano y otros; ya que [5] se separan dichos instrumentos.

Ninguna de estas bases de datos está enfocada al entrenamiento de modelos para separación solamente de la guitarra acústica. La base de datos más utilizada en el estado del arte es la de MUSDB18.

2.2.12. Algunas arquitecturas de separación de fuentes musicales

Todas estas arquitecturas que se presentarán a continuación, se dividen en: sistemas que crean máscaras que se aplican al espectrograma de la mezcla y en aquellas que estiman la forma de onda directamente.

U-Net: La red U-Net fue originalmente creada para segmentación de imágenes biomédicas. Pero en [29] se utilizó para separación de la voz en la música. Hoy en día U-Net se ha vuelto una arquitectura muy popular en la separación de fuentes musicales.

[29] requiere como entrada un espectrograma y realiza una serie de convoluciones 2D las cuales, cada una codifica representaciones más pequeñas que sus entradas. Luego, la representación más pequeña, al centro, se escala de nuevo, decodificando con el mismo número de capas de-convolucionales 2D. Cada una de las capas codificadoras se concatenan con las capas decodificadoras correspondientes.

Utiliza seis capas convolucionales codificadoras 2D con kernel de 5x5 y *stride* de 2. Después de cada capa codificadora se realiza normalización por lotes con función de activación ReLU. Se aplica un *Dropout* del 50 % a las tres primeras capas codificadoras. Luego de la sexta capa codificadora le siguen cinco capas decodificadoras con el mismo tamaño de kernel

y *stride*, y también con normalización por lotes y ReLU. La capa final tiene una función de activación Sigmoide la cual crea la máscara. Utilizan el optimizador Adam y la norma L1 como función de pérdida.

La máscara se multiplica por la mezcla de entrada y se obtiene la pérdida comparando el espectrograma real con el generado por la máscara. Dado que U-Net es convolucional, debe procesar un espectrograma con un tamaño fijo. Es decir, la señal de audio debe separarse en espectrogramas con las mismas dimensiones de tiempo y frecuencia de los que la U-Net fue entrenada.

La base de datos que utilizan es una creada por los mismos autores y contiene 3 señales por canción: la mezcla, las voces y los instrumentos. Por lo que las pistas que se separan son: voces e instrumentos.

Spleeter: Spleeter [5] es una red basada en U-Net con la modificación que al final se puede obtener las separaciones ya sea por las máscaras generadas o por filtros de Wiener y al tener varias fuentes a separar (batería, bajo, voces, piano y otros) utiliza una red U-Net para cada fuente. Al igual que su antecesor, Spleeter utiliza como entrada el espectrograma de la canción y la norma L1 como función de pérdida. Fue creado gracias a la empresa de *streaming* de música Deezer. Esta red ha sido utilizada en softwares y aplicaciones comerciales para separación de fuentes musicales. Spleeter utiliza una base de datos interna de Deezer.

Open-Unmix: Open-Unmix [23] es una arquitectura la cual utiliza capas BLSTM y en su época fue la primera red *open source* que alcanzó resultados del estado del arte. Open-Unmix trabaja en el dominio de tiempo-frecuencia para hacer las predicciones. El espectrograma de entrada se estandariza usando el promedio global y la desviación estándar para cada *bin* de frecuencia a través de todos los *frames*. Luego en la red se tiene una capa completamente conectada con normalización por lotes y función de activación Tanh; le siguen tres capas de BLSTM; dada la recurrencia, el modelo puede entrenarse y evaluarse con señales de audio de longitudes arbitrarias; dado que el modelo toma información del pasado y futuro simultáneamente, no puede usarse para aplicaciones en línea. Luego siguen dos capas completamente conectadas más con normalización por lotes y función de activación ReLU.

Se aplica un *Dropout* de 40 % a las primeras dos capas BLSTM. Toda esta arquitectura es solo para la separación de una fuente; para múltiples fuentes utiliza múltiples modelos como

el mencionado, que se entrenan con cada fuente en particular. Cada uno de estos modelos predice el espectrograma de magnitud de una fuente a separar (como por ejemplo el bajo) del espectrograma de magnitud de la mezcla de entrada. Para obtener una fuente separada se aplica la máscara aprendida al espectrograma de entrada. Toda esta arquitectura se muestra en la figura 2.13.

Hay tres detalles a notar acerca de esta arquitectura. La dimensión de frecuencia de la salida de las capas completamente conectadas antes de las capas BLSTM, es más pequeña que la de la entrada. En el primer paso después de la normalización, la red aprende a comprimir el eje de frecuencia y canal del modelo para reducir redundancia y hacer que el modelo converja más rápido. Lo segundo son las conexiones saltadas en las capas BLSTM, lo cual, permite a la red aprender si es útil usar esas capas o no. Y por último, son las normalizaciones en la arquitectura. Recordemos que la normalización ayuda en el aprendizaje ya que las entradas siempre quedan en una región bien definida.

La base de datos utilizada es la de MUSDB18 y las pistas que se separan son: batería, bajo, voces y otros.

Mask Inference: A pesar de que muchas de las arquitecturas de aprendizaje profundo generan máscaras, *Mask Inference* se refiere a una arquitectura en específico para separación de fuentes. Esta red tiene como entrada un espectrograma la cual se alimenta a un puñado de capas de redes neuronales recurrentes conectadas a una capa completamente conectada que genera la máscara. Es común utilizar la función de activación Sigmoide en la capa completamente conectada.

Un estándar es aplicar normalización por lotes antes de las capas recurrentes y aplicar *Dropout* de 30 % a las tres primeras capas recurrentes. También es común utilizar la norma L1 como pérdida entre el espectrograma original de la fuente de interés y el que resulta al aplicar la máscara al espectrograma de la mezcla.

Un ejemplo de esta técnica se puede observar en [47].

Agrupamiento profundo: Aquí se mapea cada *bin* de tiempo-frecuencia a un espacio de *embeddings* de alta dimensionalidad de los cuales los *bins* de de tiempo-frecuencia dominados por la misma fuente están cerca entre ellos y los que son dominados por fuentes diferentes están apartados. Se dice que es dominado por una fuente S_i si la mayor parte de la energía

en esa fuente es de S_i.

La red se conforma con una entrada de espectrograma, luego normalización por lotes, luego unas capas de BLSTM y por último una capa completamente conectada. La cuestión es que se requiere proyectar cada *bin* de tiempo-frecuencia a un espacio dimensional alto. La pérdida se aplica a la salida de alta dimensión del espacio de *embeddings*.

Una vez que la red es entrenada para formar el espacio de *embeddings*, se aplica un algoritmo de agrupamiento como *K-means*, para crear máscaras.

Un ejemplo de esta técnica se puede observar en [48].

Chimera: Chimera [49] combina *Mask Inference* y agrupamiento profundo en una red neuronal multi tarea. La red se entrena para optimizar ambas funciones de pérdida al mismo tiempo. Tiene una cabeza separada para cada pérdida (cabeza se refiere a una rama final de la red en la que se resuelve una tarea en específico, una red puede tener varias cabezas). Cada una de estas cabezas es su propia capa completamente conectada con un conjunto de pesos BLSTM. Aquí no se usa el agrupamiento profundo para generar las máscaras si no la cabeza de *Mask Inference*. Durante el entrenamiento, la cabeza del agrupamiento profundo actúa como regularizador que ayuda a la red a generalizar para cualquier mezcla. En esta arquitectura se utilizó como base de datos para el entrenamiento y evaluación, una versión remezclada de DSD100, creada por los mismo autores; y para complemento de la evaluación se utilizó la base de datos iKala. Las pistas que se separan son la voz y el acompañamiento.

TaSNet: TaSNet [28] es una arquitectura para separación de la voz hablada, la cual se estructura de manera similar a *Mask Inference*, con capas LSTM en el centro. La principal diferencia es que TaSNet utiliza un par de capas convolucionales para que la entrada y la salida sean formas de onda directamente. Adicionalmente, debido a que genera formas de onda, no requiere el paso adicional de multiplicar la mezcla STFT para obtener la información de la fase. La base de datos utilizada fué la de WSJ0-2mix y las pistas separadas fueron la de las voces.

Conv-TasNet: Conv-TasNet [22] es una variante de TaSNet, también es para separación del habla. En Conv-TasNet en vez de la LSTM del centro, se utilizan capas convolucionales 1D que separan la señal de entrada. Conv-TasNet se ha utilizado también en separación de fuentes musicales. Las bases de datos utilizadas fueron la de WSJ0-2mix y WSJ0-3mix. Las

pistas separadas son las voces.

Wave-U-Net: Wave-U-Net [30] es como una extensión de U-Net que trabaja directamente con las formas de onda. En vez de las convoluciones-deconvoluciones 2D, Wave-U-Net tiene una serie de convoluciones-deconvoluciones 1D que operan directamente con el audio. Las capas de codificación se concatenan con las capas decodificadoras correspondientes igual que en U-Net. Wave-U-Net utiliza MSE como pérdida entre las formas de onda originales y las estimadas. La base de datos utilizada fue la de MUSDB18 y para el entrenamiento también se agregó la base de datos de CCMixter. Las pistas separadas fueron: batería, bajo, voces y otros.

Demucs: Demucs [3] es similar a Wave-U-Net y TaSNet. Tiene conexiones saltadas y en el centro tiene dos capas BLSTM. Los autores también han sacado otras versiones de esta arquitectura ya sea para utilizarlo en tiempo real o como una mejora a la arquitectura original. Demucs utiliza la norma L1 como pérdida entre la forma de onda real y la estimada, escalada por la longitud de las señales. La base de datos que se utilizó es la de MUSDB18 y las pistas que se separan son: batería, bajo, voces y otros.

SudoRMRF: *SUccessive DOwnsampling and Resampling of Multi-Resolution Features* [31] tiene la configuración de codificador - separador - decodificador; similar a muchas arquitecturas. La entrada es la señal de la mezcla en el dominio del tiempo, el cual entra al codificador. Luego la representación latente de la mezcla entra al módulo separador, el cual estima las máscaras correspondientes para cada una de las fuentes de la mezcla. En el separador se tiene una capa de normalización, luego siguen varios bloques convolucionales llamados *UConvBlocks*, los cuales cada uno son bloques similares a la arquitectura U-Net; luego se tiene una convolución 1D por cada fuente a separar; después cada una de esas salidas se combinan con una operación *Softmax*, y luego se multiplican la mezcla con las máscaras elemento a elemento, para poder obtener la representación latente de las estimaciones. Por último, cada fuente separada se obtiene utilizando el decodificador para transformar las estimaciones del espacio latente al dominio del tiempo. El modelo con el codificador (*encoder*) - separador (*separator*) y decodificador (*decoder*) se muestra en la figura 2.14.

El modelo más a detalle es el siguiente: La arquitectura del codificador consiste en una convolución unidimensional con *Stride* igual a la mitad del *kernel*. A la salida de la convo-

lución se le aplica una función de activación ReLU. En el separador se proyecta la representación de la mezcla a un nuevo espacio de canales aplicando una normalización por capas seguido de una convolución punto a punto con *kernel* de 1x1; luego se realizan unas transformaciones no lineales repetitivas dado por los bloques U-convolucionales (*UConvBlocks*), cada uno de estos bloques extrae y agrega información de múltiples resoluciones, dejando la resolución temporal intacta; la representación visual de estos *UConvBlocks* se muestra en la figura 2.15; luego se agrega información sobre múltiples canales aplicando una convolución unidimensional para cada fuente en la representación de características transpuestas; después se combinan los códigos latentes para todas las fuentes usando una operación *Softmax* para poder obtener las máscaras estimadas; por último se estiman las representaciones latentes de cada fuente multiplicando elemento a elemento la representación codificada de la mezcla con la máscara correspondiente. El decodificador es el paso para transformar la representación del espacio latente de cada fuente al dominio del tiempo; para ello se alimenta dichas representaciones a diferentes decodificadores de convolución transpuesta.

SudoRMRF se caracteriza por ser más amigable con la memoria computacional y con el tiempo de cómputo que Conv-TasNet y Demucs tanto en el entrenamiento como en las inferencias.

Para las experimentaciones, se utilizaron las bases de datos WSJ0-2mix y ESC50; la primera es una base de datos para separación de voz y la segunda es de sonidos que se encuentran en el medio ambiente. Por lo que las pistas que se separaron en sus experimentaciones son: voces y sonidos de animales, naturaleza, aplausos, puertas, sonidos urbanos, etc.

DCUNet: *Deep Complex U-Net* [50] es como una versión refinada de U-Net, para mejora de la voz. Aunque la voz y la guitarra presentan diferentes características acústicas, ambos comparten un sobrelape considerable en sus respectivos rangos de frecuencias. Por tal razón, se prevé que sea viable para la separación de guitarra. DCUNet maneja operaciones en el dominio de los números complejos y da mucha importancia a la fase de las señales de audio.

Se puede obtener un vector de la convolución compleja dado un filtro convolucional complejo y dos matrices de números reales. En la práctica, las convoluciones complejas se pueden implementar como dos operaciones de convolución de valores reales diferentes con

filtros de convolución compartidos [50]. Las funciones de activación también fueron adaptadas al dominio de los complejos. El modelo general de DCUNet se muestra en la figura 2.16.

DCUNet es una arquitectura modificada de U-Net aplicada en el domino STFT. Dichas modificaciones comprenden: Las capas convolucionales de U-Net se reemplazaron por capas convolucionales complejas. Los *kernels* son independientes unos de otros para mejor generalización y aprendizaje más rápido. La normalización por lotes también es compleja y se aplica después de cada capa convolucional (con excepción de la última capa de la red) seguido de la función de activación. En la etapa de codificación se reemplaza el *max pooling* por operaciones complejas deconvolucionales para prevenir pérdidas de información espacial. Se modifica la función de activación a *Leaky* ReLU compleja.

En esta arquitectura se tienen cuatro variantes de las cuales cambia la cantidad de sus capas convolucionales. Las variantes son: DCUNet-10, DCUNet-16, DCUNet-20; las cuales tienen 10, 16 y 20 capas convolucionales respectivamente; y existe también la Large-DCUNet-20 que tiene también 20 capas convolucionales pero cuenta con más número de canales por capa. en la figura 2.17 se muestra la arquitectura con 20 capas convolucionales y en la figura 2.18 se muestra lo que conforma un bloque codificador (izquierda) y un bloque decodificador (derecha), en el que se observa que cada bloque conforma la convolución (o deconvolución), con sus respectivos tamaños de filtro y *stride*, seguido de la normalización por lotes y la función de activación *Leaky* ReLU; todas estas operaciones en el dominio de los números complejos.

En esta arquitectura se realiza mejora de la voz pero al fin al cabo es una red profunda cuyo entrenamiento es similar a la de separación de fuentes; se tiene una mezcla y se tiene los audios que se desean obtener; es por eso que a pesar de que sea una red para mejora de voz puede entrenarse para realizar separación de la guitarra acústica. Para las experimentaciones se utilizaron dos bases de datos, la primera se llama DEMAND (*Diverse Environments Multichannel Acoustic Noise Database*) la cual es de grabaciones de voz con y sin ruido; la segunda base de datos es un corpus de banco de voces, mas detalles en [51].

Multi-channel U-Net: *Multi-channel U-Net* [32] es como la U-Net pero de varios canales, lo cual hace que funcione como un modelo separador para cada instrumento. Utiliza una función de pérdida multitarea, ya sea el promedio ponderado dinámico (DWA por sus

siglas en inglés) o la ponderación basada en energía (EBW por sus siglas en inglés). Esta arquitectura promete costos de entrenamiento menores que las U-Net's dedicadas. La base de datos utilizada fue la de MUSDB18 y las pistas que se separan son: batería, bajo, voces y otros.

DPTNet: *Dual Path Transformer Network* [33] es una red con la configuración de codificador - separador - decodificador; para separación de la voz hablada en audios monoaurales. El codificador se usa para convertir los segmentos de la mezcla de forma de onda a un espacio de características. Luego las características se alimentan a la capa de separación para construir una máscara para cada fuente. Por último el decodificador reconstruye las formas de onda de las fuentes. En el separador se cuenta con conjuntos de Transformers duales. Transformer es una arquitectura relativamente nueva en el aprendizaje profundo, la cual ha sido muy poco explotada en el audio.

Los Transformers utilizan un mecanismo de atención lo cual promete generar mejores resultados que las redes recurrentes. Los Transformers utilizados en [33] tienen una mejora del Transformer original, agregando una RNN en vez de la capa lineal que va antes de la función ReLU. La arquitectura general se muestra en la figura 2.19.

El codificador puede ser visto como un banco de filtros ya que se aplica una convolución unidimensional seguido de una función ReLU. La capa de separación se compone de 3 etapas: la segmentación, el procesamiento con el transformer dual y el sobrelape y suma. En la segmentación se divide la señal en partes iguales las cuales se concatenan formando un tensor 3D. El transformer dual está compuesto por un codificador y un decodificador. La parte codificadora comprende 3 módulos: módulo de atención de producto punto escalado, módulo de atención multicabeza y red neuronal de posicionamiento. Además de estos 3 módulos, el transformer contiene capas de normalización. El primer transformer le llaman intra-transformer y el segundo inter-transformer. La salida del inter-transformer se usa para aprender una máscara para cada fuente utilizando una capa convolucional 2D. Las máscaras se convierten a secuencias utilizando sobrelape y suma; estas se multiplican elemento a elemnto con las segmentaciones para obtener las características codificadas de cada fuente. En el decodificador se usa una convolución transpuesta para reconstruir las señales de la voz y se aplica sobrelape y suma para obtener las formas de onda finales. La estructura del

decodificador es simétrica al codificador.

Las bases de datos utilizadas fueron la WSJ0-2mix y la LS-2mix. Y las pistas separadas son las voces.

D3Net: *Densely connected Dilated DenseNet* [34] es una red que involucra una convolución multi dilatada, el cual tiene diferentes factores de *dilation* en una capa, para modelar diferentes resoluciones al mismo tiempo. También se logra evadir el problema del *aliasing*. La entrada de la red es el espectrograma de la canción y la salida de la red se utiliza para calcular el filtro de Wiener multi-canal para obtener las separaciones finales. La base de datos utilizada fue la de MUSDB18 y las pistas separadas fueron: batería, bajo, voces y otros.

Meta-Tasnet: Meta-Tasnet [52] es una red con la configuración de codificador - separador - decodificador. Propone un modelo inspirado en el meta aprendizaje jerárquico donde se usa un generador de parámetros para predecir los pesos de modelos de extractores individuales. El modelo del extractor se basa en Conv-TasNet. El generador utiliza un vector *one-hot* para codificar los atributos de los instrumentos a lo largo de múltiples ejes. La base de datos que se utilizó es la de MUSDB18 y las pistas que separa son: batería, bajo, voces y otros.

SamsNet: *Sliced Attention-based Neural Network* [1] es una red neuronal con atención repartida. En cada uno de sus N módulos de su arquitectura, se tiene una capa de normalización, luego el módulo de atención multi cabeza, luego otra capa de normalización y por último una capa convolucional. La base de datos utilizada fue la de MUSDB18 y las pistas que se separan son: batería, bajo, voces y otros.

2.3. La guitarra acústica

La guitarra es un instrumento musical de cuerda, el cual se utiliza en muchos géneros musicales. En la música moderna existen la guitarra acústica y eléctrica pero en esta sección se hablará más de la guitarra acústica.

2.3.1. Antecedentes históricos

Desarrollos similares a los de la escuela de fabricantes de violines de Cremona ocurrieron con los fabricantes de guitarras a finales de los años 1700s. Se hicieron cambios para mejorar la proyección y el *sustain* de la guitarra acústica, la cual en ese tiempo tenía cinco pares de cuerdas. A inicios de los años 1800s, las guitarras en Paris y Viena, tenían seis cuerdas que tendían a ser más anchas y robustas que las de su predecesor de cinco pares de cuerdas. En esos años se crearon cuantiosas obras con guitarra acústica. El instrumento y el estilo de interpretación se desarrolló en los años 1800s, a este tiempo se le llamó la época dorada de la guitarra, y se adaptaba perfectamente a la era clásica de Viena. Sin embargo, con el crecimiento del Romanticismo vinieron demandas de contrastes más fuertes y un rango dinámico más amplio que la guitarra podía producir. El piano y el violín se convirtieron en los instrumentos más famosos en la era del Romanticismo, mientras que la guitarra acústica retuvo un círculo pequeño de compositores y músicos. Sin embargo no se perdieron las semillas plantadas en Viena y Paris, ya que se siguieron creando obras con guitarra como óperas y sinfonías.

En la primera mitad del siglo XX, a pesar que la guitarra sufrió de un repertorio mediocre, la guitarra pasó por cambios similares a los que pasó a principios del siglo XIX. En este siglo existía un tipo de guitarra llamada Terz la cual era afinada más aguda que la normal y de dimensiones un poco más pequeñas que la guitarra actual. También, en ese tiempo se empezó a desarrollar la guitarra Quinte-Basse.

La guitarra clásica moderna nació del compositor/guitarrista Francisco Tarrega y el lutier Antonio de Torres Jurado. El instrumento de Torres tenía un cuerpo más largo y una mayor longitud de la escala que sus predecesores, y utilizaba un nuevo diseño interior tipo abanico de mano (aún utilizado hoy en día) el cual resultó en una mejora considerable de la proyección y la respuesta tonal de la guitarra. Tarreaga no solo ayudó a desarrollar el instrumento sino también a extender las técnicas adoptando la técnica de mantener la mano derecha perpendicular a las cuerdas y apoyando los dedos en las cuerdas siguientes que las que se tocan. También ayudó a extender el repertorio componiendo un vasto número de piezas y creando transcripciones de composiciones para tocarse en guitarra, especialmente

de Bach y Chopin, mostrando que la guitarra puede tener un argumento polifónico, lo cual ayudó a ampliar el vocabulario del instrumento y a hacerlo más atractivo para la composición. Los logros y sus métodos de enseñanza de Tarreaga inspiraron una nueva generación de guitarristas. Incluyendo a Andrés Segovia. La guitarra clásica como la conocemos hoy en día, seguramente se debe a los esfuerzos de Segovia en la primera mitad del siglo XX. Él, constantemente se esforzó por extender el instrumento y su música. Su dedicación, interpretaciones y grabaciones han sido un factor primordial para que la guitarra clásica capte la atención del público.

A inicios de los años 1900s hasta la segunda guerra mundial, el repertorio de la guitarra solista incrementó gracias a Segovia y algunos guitarristas poco conocidos de Europa y Sudamérica. También empezó a haber repertorios para no solistas. Justo después de la segunda guerra mundial, cuando la búsqueda de nuevos timbres y contrastes instrumentales se volvió de suprema importancia, hubo una avalancha de composiciones queriendo la guitarra, usualmente en ensambles pequeños. Los compositores empezaron a trabajar con intérpretes individualmente para ver que tipo de sonidos podrían producirse. Con el espíritu experimental como el de estar en un laboratorio de sonido, los escritores y músicos trabajaron para extender el rango y carácter de la expresión musical y nuevos sonidos.

En los años 1960s la personalidad musical de la guitarra pareció dividirse. El vanguardismo de los años 1950s continuó la exploración de los instrumentos musicales como fuentes de material sonoro no tradicional, y la guitarra, que fue ganando popularidad en este período de exploración, no fue excluida de esta experimentación. Se introdujo la guitarra eléctrica, pero los compositores tradicionales encontraron en la guitarra acústica una paleta satisfactoria para sus necesidades expresivas. El repertorio en este periodo fue aún más diverso que la década anterior. El año 1963 fue un pivote para el repertorio de la guitarra solista debido a la creación de tres diferentes piezas maestras modernas: *Nocturnal* de Benjamin Britten, *Si le jour parait...* de Maurice Ohana y *Las Seis Cuerdas* de Alvaro Company.

Para los años 1970s la guitarra (al igual que la eléctrica) fue finalmente aceptada por la comunidad musical ya que se componían obras que mostraban los atributos de la guitarra en vez de acentuar sus debilidades. La guitarra clásica de cuerdas de nailon que era considerada capaz de producir sonidos suaves se benefició enormemente del desarrollo de la tecnología

con la amplificación de alta calidad, portátil y de bajo costo. Lo que fue controversial ahora es considerado algo común. El último cuarto del siglo XX presenció el sueño de Segovia de la aceptación de la guitarra en institutos de educación superior. Ahora, miles de programas para guitarra existen alrededor del mundo, brindando nuevas generaciones de no solo guitarristas si no, más importante, músicos que tocan guitarra. Los conservatorios y universidades generaron una avalancha de talentosos y entusiastas intérpretes que continuaron en la búsqueda de nuevos repertorios. Formando así, nuevos ensambles profesionales o dúos, tríos y cuartetos de guitarra. [11].

2.3.2. Partes de la guitarra acústica

En la figura 2.20 se muestra una guitarra acústica, con indicaciones de sus partes principales. Las partes que conforman una guitarra acústica clásica son las siguientes:

- Clavijero: Es la parte más extrema del mástil, aquélla que más lejos se encuentra del cuerpo de la guitarra. A veces recibe el nombre de pala. Es en esta parte de la guitarra donde se localizan encajadas las clavijas que permiten afinar las cuerdas variando su tensión hasta lograr la altura del sonido deseada. Además, es el lugar dónde el fabricante suele poner su logotipo y/o marca.

- Clavijas: Las clavijas son cada una de las piezas giratorias de pequeño tamaño situadas a cada lado de la cabeza que nos permiten modificar la tensión de las cuerdas a través del giro de un tornillo. Con las clavijas se puede aumentar o disminuir la tensión de las cuerdas para poder afinarlas más agudas o más graves [53].

- Mástil: Es una pieza larga de madera que sobresale del cuerpo y llega hasta el clavijero [54]. En el mástil se encuentra el clavijero, la cejuela, el diapasón y los trastes. La calidad de la madera y la forma de mástil son importantes para definir el sonido de la guitarra. Un mástil diseñado incorrectamente, hará que el sonido "trastee" [53]. Las guitarras electroacústicas y eléctricas tienen una varilla dentro del mástil llamado alma, esta se usa para hacer que el brazo cambie su forma, más cóncavo o más convexo.

- Diapasón: Es una delgada tabla de madera localizada en la parte frontal del mástil;

se extiende desde la boca de la guitarra hasta la cejilla. A lo largo del diapasón se encuentran incrustados los trastes, entre los cuales, los dedos aprietan las cuerdas de la guitarra para cambiar de nota. [54].

- Trastes: Son las delgadas barras metálicas insertadas a lo largo del diapasón, van colocadas de forma perpendicular a las cuerdas [54]. Los trastes dividen el mástil en intervalos, en los que cada uno equivale a medio tono. También se le llama trastes al espacio que existe entre dos barras continuas. Estas son las zonas sobre las que los dedos del guitarrista presiona las cuerdas para obtener una nota en específico. Así, el guitarrista define las notas musicales en la guitarra, cambiando la vibración efectiva de una cuerda. [54].

- Cuerpo o caja: En la guitarra acústica, el sonido es definido por el cuerpo; el cual También se conoce como caja de resonancia [53]. El tipo de madera con que esté hecha la caja define un sonido particular a la guitarra. El cuerpo proporciona un anclaje para el mástil y el puente. La caja funciona como la cámara de sonido de la guitarra. Esta parte influye en el volumen y es donde se crea el efecto acústico. Las partes que componen al cuerpo son: el fondo, la tapa y los aros laterales. Las funciones principales de la tapa son: aguantar el puente, el cual soporta la tensión de las cuerdas, y la generación del sonido mediante las vibraciones. [54].

- Boca: La boca de la guitarra es un agujero en el centro de la tapa. La boca de la guitarra es la base para proyectar el sonido ya que es la salida y entrada del aire. El sonido es producido por las vibraciones del aire dentro de la caja, y es por eso que conforma una parte importante para el sonido de la guitarra acústica. [54].

- Puente: El puente es la parte que sostiene a las cuerdas y se transmite las vibraciones a la caja [53]. Esta parte comunica las cuerdas al cuerpo [54]. En las guitarras acústicas, por lo general, los puentes están hechos de madera [53].

- Cejuela: La cejuela o cejilla mantiene las cuerdas en su posición correspondiente. La cejuela tiene 6 hendiduras, en la que cada se coloca a una cuerda [53]. Comúnmente se le llama hueso por el material que con que está hecho, que por lo general es de marfil.

- Selleta: Esta pieza va en el puente y sobre ella se apoyan las cuerdas [54]. La función de esta pieza es levantar las cuerdas a un nivel deseado y transmitir la vibración de las cuerdas a la tapa armónica [53]. También recibe el nombre de hueso. Pueden utilizarse otros materiales como la melamina o materiales sintéticos. Las variaciones en el material de la elaboración de la selleta influirá en el sonido puesto que variará el matiz por sus diferentes capacidades de vibración. Es importante que este ajustado a la altura indicada para que el sonido suene correctamente y evitar que la guitarra trastee.

2.3.3. El sonido en una guitarra acústica

La guitarra acústica utiliza las vibraciones de las cuerdas para generar su sonido; la tapa y la caja de resonancia también juegan un papel importante en el sonido de la guitarra acústica.

La forma en que una cuerda vibra se determina por su longitud, tensión y su material. Una cuerda vibrando genera varias frecuencias audibles simultáneamente y estas frecuencias son 2, 3, 4, 5 ... veces, la frecuencia fundamental de la cuerda. Por ejemplo, si la frecuencia fundamental es 110 Hz, cuando la cuerda vibra en 2 secciones iguales, su frecuencia será de 220 Hz, cuando vibre en 3 secciones iguales, la frecuencia será de 330 Hz. En la figura 2.21 se ilustra el número de secciones que vibra una cuerda.

Estas relaciones de la vibración se llaman armónicos. Al tocar las cuerdas en diferentes trastes, el guitarrista suprime algunas frecuencias y resalta otras cambiando así el contenido armónico de las vibraciones. Cuando una cuerda se toca, se envían dos pulsos u ondas viajando direcciones opuestas a lo largo de la cuerda. Cuando cada una de estas ondas llega a su final, se refleja de nuevo y viaja en dirección inversa. Luego viaja al otro final de la cuerda y el proceso se repite. En la figura 2.22 se ilustra la reflexión de las ondas de una cuerda.

Estas dos ondas que viajan en sentido contrario a través de la cuerda, se cruzan e interfieren entre ellas, sumando sus amplitudes. Si ambas ondas son positivas el valor sumado será más grande que cualquiera de las dos si las ondas están en fase; si una es positiva y otra negativa se cancelaran y sumadas darán cero.

La frecuencia y amplitud del sonido producido por una cuerda depende de su grosor, flexibilidad y material. Una guitarra acústica puede tener cuerdas de nailon o de metal, la diferencia de tono de cada una se debe a la poca elasticidad del nailon. Algo que también afecta el sonido es la no uniformidad de las cuerdas dado el desgaste por el uso y por el contacto con los trastes, cambiando el desempeño de la cuerda.

El sonido se transmite con el cambio de la presión del aire. Dado que una cuerda no es muy eficiente para mover el aire, se necesita un medio de comunicación de las vibraciones de la cuerda al aire. Esta comunicación se logra con la tapa resonante del cuerpo de la guitarra y la caja. La tapa es más eficiente para irradiar las vibraciones que las cuerdas por si solas. Lo que une las cuerdas con la tapa es el puente el cual además de sostener las cuerdas, ayuda a determinar el sonido de acuerdo a que tantas vibraciones se transmiten a la tapa.

La tapa puede verse como una membrana vibratoria. Una membrana rectangular vibra en modos inarmónicos, las frecuencias no estarán en proporciones simples como lo están en una cadena; mientras que una membrana circular vibra en muchos modos, como se puede observar en la figura 2.23.

Es por eso que no es común ver una guitarra acústica con tapa cuadrada. Debido a que la tapa debe resistir la gran tensión de las cuerdas y vibrar de manera adecuada, es importante lo que hay detrás de la tapa. Se colocan pedazos de madera en ciertas formas y medidas para que la tapa tenga fuerza suficiente para soportar la tensión de las cuerdas pero sin inhibir las vibraciones. El trabajo artístico de un lutier cumple con estos dos objetivos de manera balanceada. En la figura 2.24 se muestran los diseños de tapas de Robert Bouchet y Antonio de Torres Jurado.

Investigadores han logrado grabar los patrones de vibración de la guitarra utilizando técnicas holográficas. Esto se puede observar en la figura 2.25 para diferentes frecuencias. Como se puede observar, los patrones de las franjas en las diferentes partes de la guitarra indican el desplazamiento de tapa.

Cada uno de los hologramas en la figura representan la vibración de una sola frecuencia en específico, pero como al tocar una cuerda se generan muchas ondas simultáneamente, la tapa vibra de manera compleja.

El aire que se encuentra dentro de la caja es otro factor importante que afecta a la

resonancia de la tapa pero también al sonido de la guitarra en general. La resonancia del aire se forma de una onda estacionaria creada dentro de la caja de resonancia. La frecuencia del aire dentro de la caja ronda al rededor de los 100 Hz. Cuando un resonador como el cuerpo de la guitarra se excita por una cuerda vibrando, la curva de respuesta del resonador dará forma al espectro de la señal de la cuerda y, por lo tanto, afectará el espectro del sonido final. Dos guitarras pueden diferir en sus respuestas de frecuencia, incluso si tienen las mismas cuerdas y se tocan por el mismo guitarrista.

Los movimientos de la cuerda determinan los movimientos de la tapa, pero a la vez, la flexibilidad de la tapa determina también los movimientos de la cuerda; ambos sistemas se afectan uno al otro, es un sistema acoplado y el guitarrista controla la cantidad y calidad de la fuerza aplicada a la tapa por la manera de tocar las cuerdas. Cuando se toca una cuerda la tapa irradia la mayor parte de la energía en sus frecuencias resonantes de manera eficiente, pero una parte de la energía en esas frecuencias se retro-alimentará a la cuerda y también a las otras 5 cuerdas a través del movimiento del puente. Una forma de dar un mayor *sustain* a notas altas de la primera cuerda (las cuales resuenan menos) es duplicar (casi silenciosamente) una nota en la(s) octava(s) de abajo, permitiendo que la resonancia similar cree un efecto *sustain* más largo.

Cuando el movimiento de una cuerda hace vibrar a la tapa, la cuerda toma cierto tiempo para "convencer" a la tapa para cambiar su estado de "reposo". Esto crea interacciones complejas entre ambos sistemas, y estas interacciones resultan en las complejidades de la forma de onda que se puede escuchar al principio del sonido. Este sonido se llama transiente debido a que ocurre muy rápido al inicio del tono y desaparece tan pronto la cuerda convence a la tapa de vibrar a la frecuencia determinada por la cuerda. En la guitarra acústica y los demás instrumentos de cuerda, en realidad, la tapa no inicia en un estado de reposo, si no que tiene una deformación debido a la tensión de las cuerdas.

En resumidas cuentas la guitarra acústica es un instrumento de cuerda con tres sistemas para producir su sonido: las cuerdas, la tapa resonante y el cuerpo. Al tocar una cuerda se transmiten las vibraciones a la tapa a través del puente, al tener vibraciones en la tapa, la frecuencia del aire dentro del cuerpo cambia y se produce el sonido característico de la guitarra acústica. [11].

2.4. Análisis musical

La música puede considerarse como el conjunto de sonidos producidos por instrumentos musicales (o virtuales) lo cual genera un efecto expresivo y agradable al oído humano. La música cumple con ciertas características y sigue ciertas normas.

2.4.1. Nota musical

En la música existe algo llamado la escala diatónica, lo cual está formada por las notas musicales básicas: Do, Re, Mi, Fa, Sol, La, Si. Estas notas tienen una nomenclatura norteamericana: C, D, E, F, G, A, B, respectivamente. Esto es conocido como el alfabeto musical.

En la música existen intervalos entre notas llamados tonos, también existen los semitonos, el cual es la mitad de un tono. De Mi a Fa y de Si a Do hay un semitono, para todos los demás intervalos de notas hay un tono o dos semitonos. Adicionalmente existen unas figuras que indican si una nota se aumenta o se disminuye un semitono, respectivamente: Sostenido # y Bemol ♭. A cada una de las notas, con excepción de Mi a Fa y de Si a Do, se les puede agregar un # y a cada una de las notas, con excepción de Fa a Mi y de Do a Si, se les puede agregar un ♭. Aquí hay que hacer una observación, las notas sostenidas son lo mismo que las bemoles de la nota siguiente, por ejemplo, La# es lo mismo que Si♭. Al final se obtienen 12 notas en la música: Do, Do# (o Re♭), Re, Re# (o Mi♭), Mi, Fa, Fa# (o Sol♭), Sol, Sol# (o La♭), La, La# (o Si♭), Si. Esta es la escala cromática en la música. Estas 12 notas son un ciclo de notas en la música, este ciclo se repite ya sea a mayores o menores tonalidades. Cuando tocas una misma nota en un tono mayor o menor se dice que cambiaste de octava. Entonces, cada ciclo de notas se le llama octava. En la figura 2.26 se puede observar la forma de onda y espectrograma de las notas de la escala diatónica tocadas en una guitarra acústica de cuerdas de metal.

En la forma de onda se pueden ver 8 figuras o partes en color naranja, las cuales corresponden a las notas: Do, Re, Mi, Fa, Sol, La, Si, Do. Y en el espectrograma se observan diferentes líneas verticales alineadas a cada inicio de nota en la forma de onda. En el espectrograma en cada nota se observan diferentes lineas horizontales las cuales corresponden

a los armónicos de la frecuencia fundamental de la nota, ya que cada instrumento tiene su propio timbre y la guitarra acústica no es la excepción.

2.4.2. Acordes

Los acordes son la combinación de dos o más notas tocadas al mismo tiempo. Los acordes se crean a partir de una nota base. Se pueden crear acordes a partir de las 12 notas de la escala cromática. Pero también, en la música, existen cuatro tipos básicos de acordes: mayores. menores, disminuidos y aumentados. Cada uno tiene sus diferentes características musicales en la manera de formarlos. Existen muchas otras variantes que se les puede agregar a cada uno de estos acordes para cada nota base, como por ejemplo las séptimas o los suspendidos. Existe mucha más teoría musical sobre los acordes pero para fines de este trabajo no se entrará en detalle; lo esencial a saber es qué acordes se forman de dos o más notas tocadas al mismo tiempo.

2.4.3. Ritmo

En la música, el ritmo es la distribución de las duraciones de los sonidos a lo largo del tiempo. El ritmo se relaciona con cualquier movimiento repetitivo. Es la parte dinámica, organizativa y repetitiva en la música. Es la distribución temporal de las notas y silencios. [55] [56].

2.4.4. Melodía

La melodía es el arreglo significativo y coherente de una serie de notas, este arreglo se realiza según la tonalidad en la cual se diseña la melodía [55]. La melodía es la esencia de la canción y lo que la hace reconocible [56]. Es un conjunto de sonidos, dentro de un ámbito musical, que suenan sucesivamente uno después de otro; también puede tener un significado emocional, combinando alturas, velocidades y otros elementos en la música [55].

2.4.5. Armonía

La armonía tiene la función de acompañamiento, armazón y base de las melodías [56]. Es la superposición de sonidos producidos simultáneamente. En la armonía se utilizan mucho los acordes. Es ciencia y arte a la vez; ciencia porque enseña a combinar los sonidos siguiendo las reglas musicales con la finalidad de formar acordes; y arte porque la pieza musical resultará de la habilidad y el gusto de la formación de voces armónicas [55].

2.4.6. Tiempo

Para marcar el tiempo en la música se tienen diferentes velocidades que se miden en *Beats* por minuto (BPM). Algunas velocidades comunes en la música son 85, 100, 120, 140 BPM. Mientras más alto sea este valor, más rápido será una canción y mientras más bajo sea el valor más lento será una canción.

Un *Beat* equivale a una nota negra o 1 tiempo. Una nota negra es una figura de tiempo en la música. Las figuras de tiempo son:

- Redonda: Equivale a 4 tiempos.
- Blanca: Equivale a 2 tiempos.
- Negra: Equivale a 1 tiempo.
- Corchea: Equivale a 1/2 tiempo.
- Semicorchea: Equivale a 1/4 de tiempo.
- Fusa: Equivale a 1/8 de tiempo.
- Semifusa: Equivale a 1/16 de tiempo.

Un compás en la música es una métrica compuesta por varias figuras (o unidades) de tiempo organizadas en grupos. Algunos compases más comunes son 4 cuartos, 3 cuartos, 2 cuartos y 6 octavos.

2.5. MIDI

MIDI es la abreviación de *Musical Instrument Digital Interface*. MIDI es un sistema universal usado para comunicar de forma estandarizada instrumentos musicales electrónicos y otros dispositivos, independientemente de quién sea el fabricante [57]. También puede verse como un protocolo o lenguaje universal para dispositivos digitales. Los productores musicales utilizan MIDI para generar bases musicales, agregando los instrumentos virtuales que ellos deseen.

Cuando las compañías de música comenzaron a fabricar instrumentos digitales desarrollaron sus propios protocolos para controlarlos; con esto, la música digital obtuvo una gran variedad de protocolos diferentes e incompatibles entre sí. El MIDI se presentó en 1983 para tratar de traer orden al caos que se había formado y beneficiar a todos los fabricantes. El MIDI es la forma más sencilla de poder manejarnos en un contexto musical digital. Con el MIDI es fácil enviar y recibir señales, hay innumerables instrumentos y controladores MIDI; y está muy presente en la música moderna. [57].

2.5.1. Archivos MIDI

Un músico puede utilizar una computadora con un DAW (*Digital Audio Workstation*) y un controlador MIDI para crear archivos MIDI que pueden ser utilizados para crear una canción. Estos archivos MIDI, en su interior tienen varias señales con diferentes parámetros; también se tiene la información para cada instrumento grabado.

Cada señal MIDI tiene una serie de parámetros que le dan el toque más musical, como son la intensidad y duración de cada nota, además del tono (si la nota es Do, Re, etc.). Los archivos MIDI pueden ser modificados tanto en el DAW como con programación utilizando ciertas librerías que puedan manejar la información de un MIDI.

Los archivos MIDI pueden ser de dos tipos: tipo 0 y tipo 1. En el tipo 0 todos los instrumentos se encuentran en una misma pista. Y en el tipo 1, cada instrumento se encuentra en una pista por separado. En un archivo MIDI la pista 0 es una pista con información necesaria para el tempo de la canción.

La información dentro de una pista de un archivo MIDI se separa en mensajes y se

organizan de manera secuencial en el tiempo que dura la canción. Existen diferentes tipos de mensajes, los cuales contienen distinta información como el tono de una nota, el tiempo de cada nota (si una nota se toca o se deja de tocar), el instrumento de la pista (a este parámetro o término en MIDI se le llama *Pogram Change*), cuando se inicia y cuando se termina de tocar la canción, la presión con que se toca cada nota, entre otros. Todos estos mensajes cambian de valor o no a lo largo de la canción.

Para determinar el instrumento de una pista se modifica el *Pogram Change*. Existen varios instrumentos a escoger con este parámetro, desde diferentes tipos de piano, cuerdas hasta percusiones y efectos de sonido, en total son 128 instrumentos. La guitarra acústica de cuerdas de nailon ocupa el número 25 y la de cuerdas de metal ocupa el número 26 en la lista.

El tiempo general en un archivo MIDI se rige por *ticks* y *beats*. Un *beat* (como vimos en la sub-sección de tiempo) equivale a una nota negra o un tiempo. Los *beats* a su vez se dividen en *ticks*, lo cual es la unidad más pequeña en MIDI. La cantidad de *ticks* se define en *ticks* por *beat*. [58].

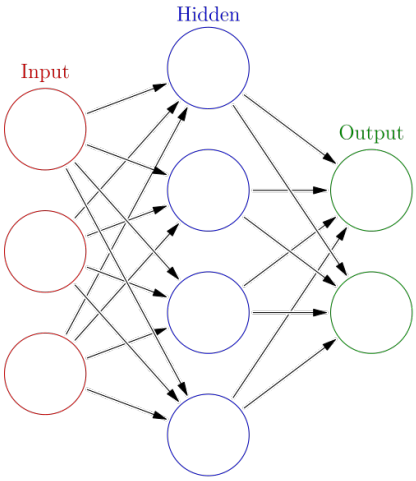

Figura 2.8: Red de completamente conectada. Imagen extraída de [10].

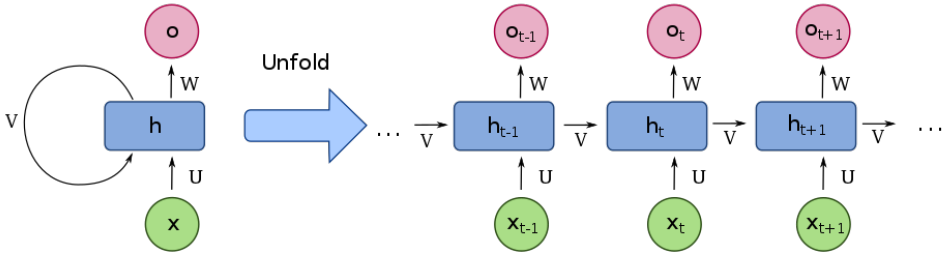

Figura 2.9: Red recurrente. Imagen extraída de [10].

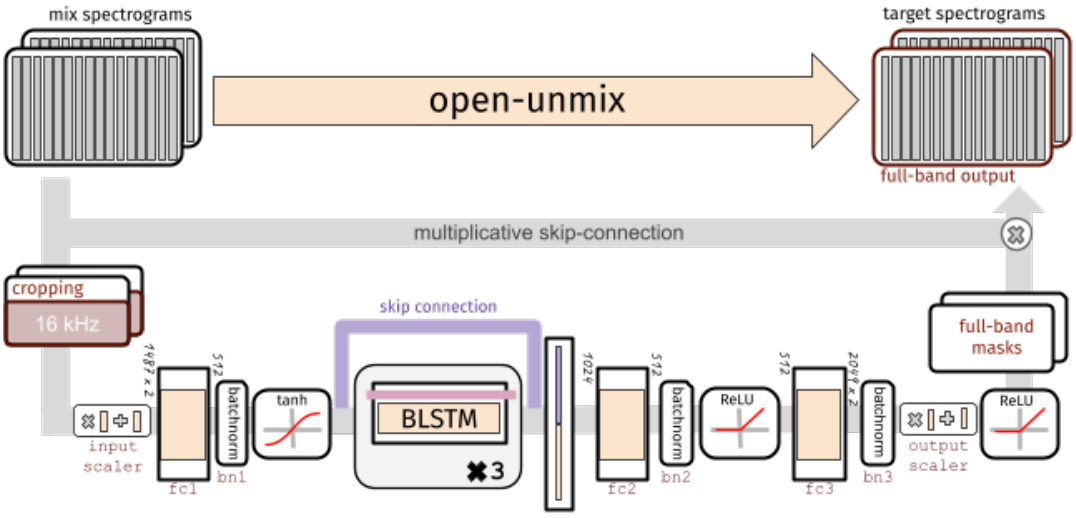

Figura 2.13: Diagrama del modelo de Open-Unmix para una fuente. Imagen extraída de [23].

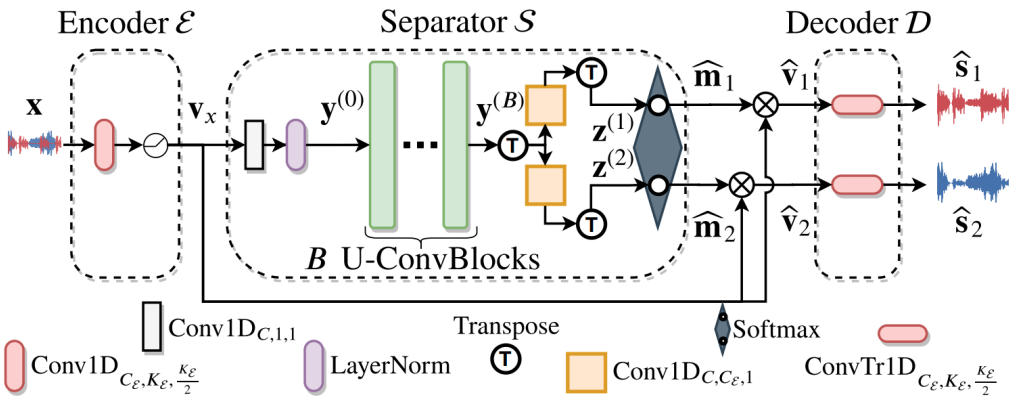

Figura 2.14: Diagrama del modelo de SudoRMRF para una fuente. Imagen extraída de [31].

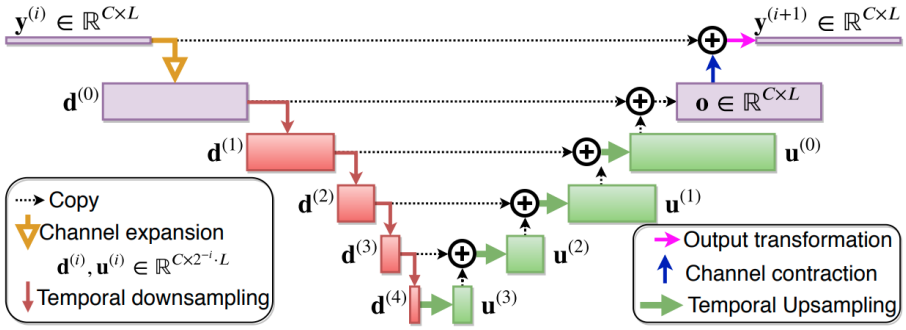

Figura 2.15: Diagrama de un bloque U-convolucional. Imagen extraída de [31].

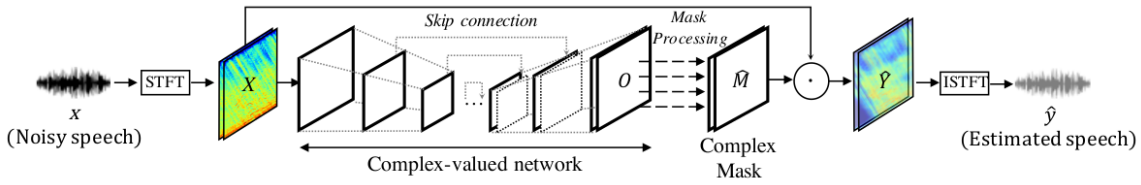

Figura 2.16: Diagrama general de la arquitectura DCUNet. Imagen extraída de [50].

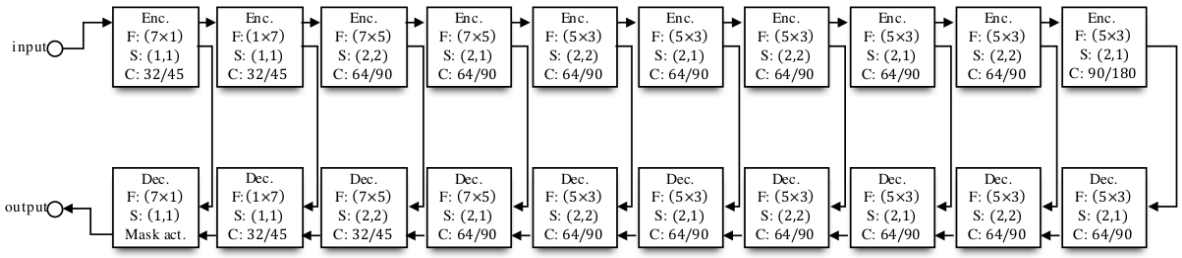

Figura 2.17: Modelo DCUNet de 20 capas. Imagen extraída de [50].

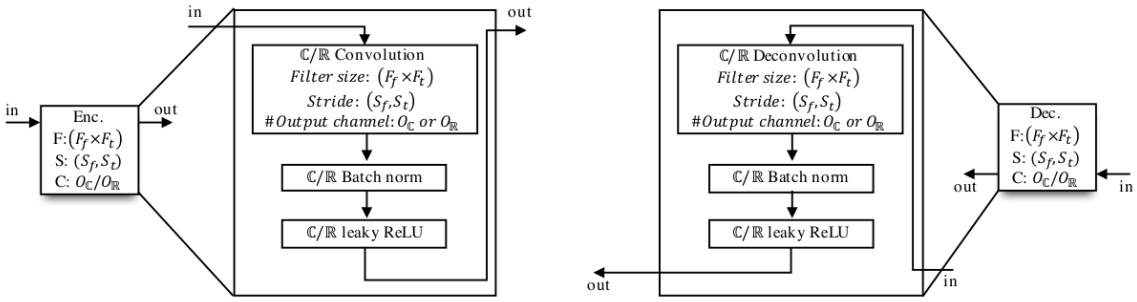

Figura 2.18: Bloques codificador (izquierda) y decodificador (derecha) a detalle. Imagen extraída de [50].

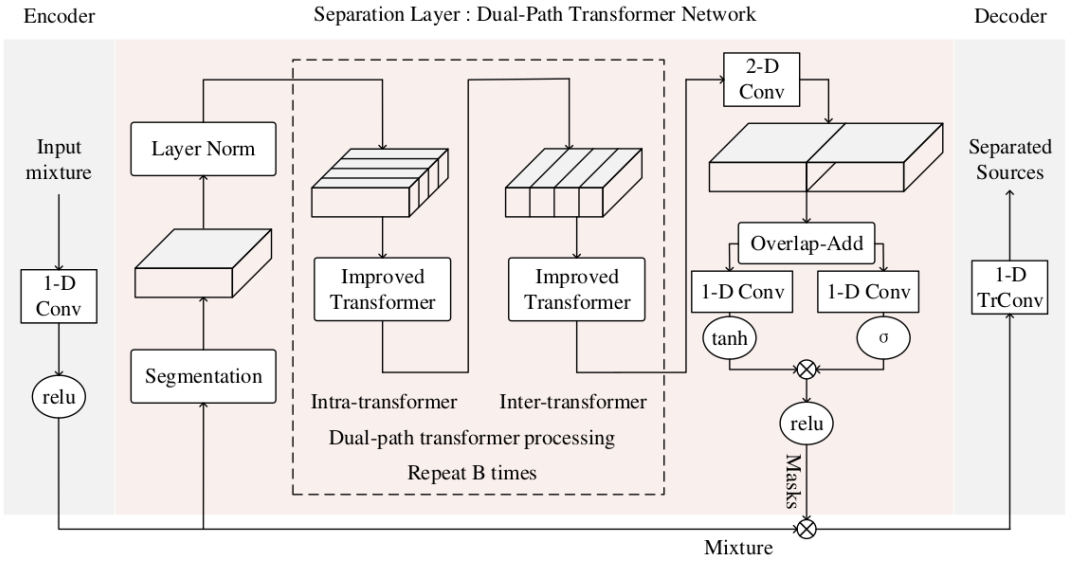

Figura 2.19: Diagrama general de la arquitectura DPTNet. Imagen extraída de [33].

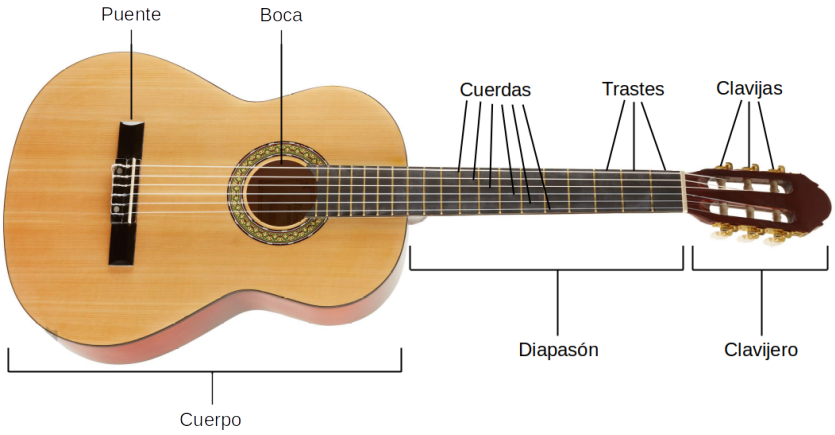

Figura 2.20: La guitarra acústica.

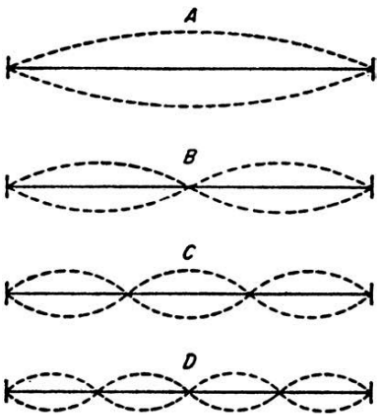

Figura 2.21: Primeros cuatro modos de vibración de una cuerda. Imagen extraída de [11].

Figura 2.22: Reflexión de las ondas en una cuerda. Imagen extraída de [11].

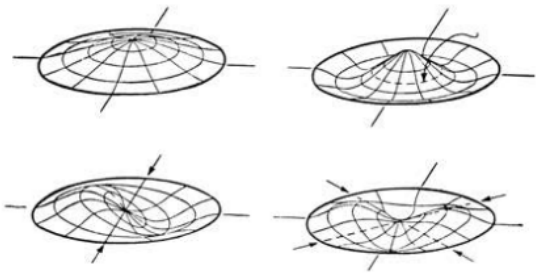

Figura 2.23: Modos de vibraciones de una membrana circular. Imagen extraída de [11].

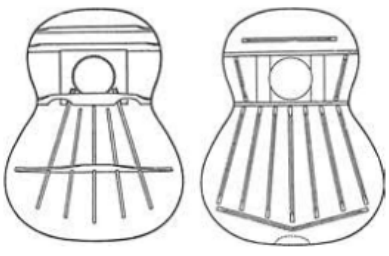

Figura 2.24: Dos diseños de la parte trasera de la tapa de la guitarra acústica. Imagen extraída de [11].

a. 103 Hz b. 213 Hz c. 268 Hz d. 436 Hz e. 553 Hz f. 731 Hz

Figura 2.25: Vibraciones en la tapa de la guitarra acústica a diferentes frecuencias (vista desde arriba). Imagen extraída de [11].

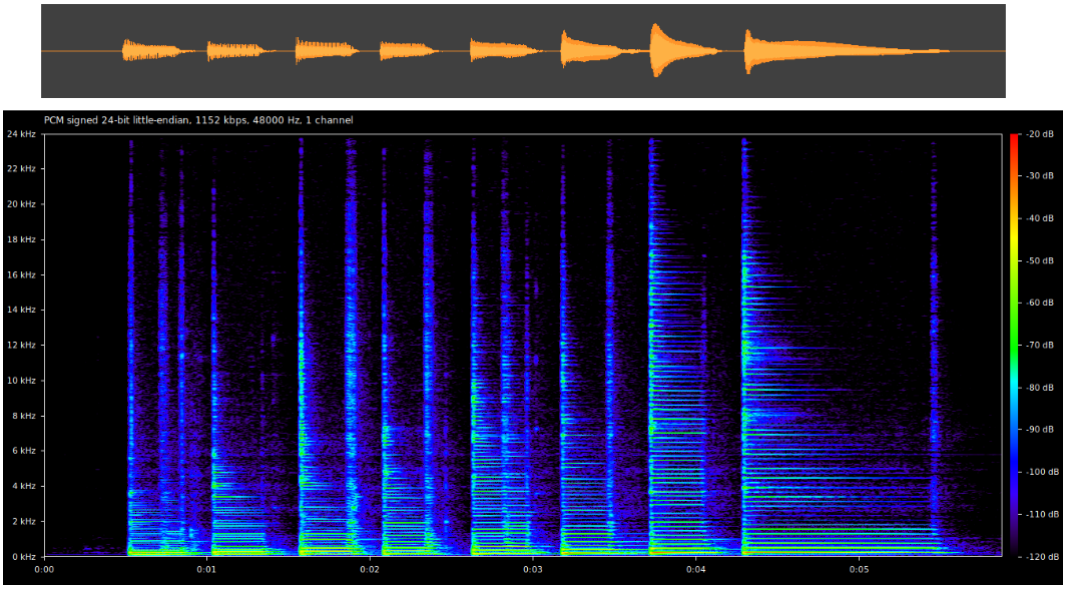

Figura 2.26: Forma de onda (arriba) y espectrograma (abajo) de la grabación de la escala diatónica tocada en una guitarra de cuerdas de metal.

Capítulo 3

Metodología

En las siguientes secciones se explican la metodología que se llevó a acabo para realizar el proyecto.

3.1. Generación de la base de datos

La base de datos requerida para el entrenamiento de los modelos de aprendizaje profundo, es aquella que tenga canciones (en formato de audio) suficientes para el entrenamiento y validación; se debe incluir tanto la mezcla final como su correspondiente pista de guitarra acústica por separado, eso para cada canción en la base de datos. Desafortunadamente no se encontró una base de datos adecuada en internet, por lo que se procedió a crear una con las características necesarias.

La base de datos se creó a partir de archivos MIDI que tengan pista de guitarra acústica. Se decidió utilizar archivos MIDI debido a que de esta forma se obtendría toda la base de datos de manera más rápida que si se buscara en internet canción por canción con sus respectivas pistas de guitarra acústica. Además, al utilizar MIDI's, se evitan problemas de piratería o copyright; y se logró realizar un método semi automatizado para obtener los archivos de audio a partir de los datos de los archivos MIDI.

La primera fase para la creación de las canciones a partir de MIDI's es obtener los archivos MIDI. En internet se encontraron varias bases de datos de archivos MIDI, de las cuales, sumando todos los archivos, habían aproximadamente 43,559 archivos MIDI. Sin

embargo para la base de datos final no se requiere esa cantidad grande de archivos de audio.

Inicialmente se empezó a trabajar los MIDI's con un DAW llamado Ableton Live, donde se cargaba el MIDI y de manera manual se asignaban los instrumentos virtuales a cada pista, se editaban las pistas, se añadían procesamientos y efectos y al final se exportaban los audios (en formato WAV, estéreo, a 24 bits de resolución, a una frecuencia de muestreo de 48 kHz) tanto de la mezcla final como de la guitarra acústica. Este proceso manual se realizó para 23 canciones, pero se observó que se tardaría mucho para completar toda la base de datos. Entonces se ideó un proceso semi automático para pasar de MIDI hasta obtener los WAV's de mezcla y guitarra acústica.

Lo primero es seleccionar los MIDI's que contengan una guitarra acústica, extraer esa o esas pistas y guardarlas en un MIDI a parte. Para ello se creó un programa en Python (con una biblioteca llamada Mido, que sirve para el manejo de archivos MIDI) que primero carga un archivo MIDI, luego identifica qué tipo de archivo MIDI es. Si era de tipo 0, se descartaba el archivo en cuestión y se continua con otro archivo, ya que en el tipo 0 todos los instrumentos se encuentran en una misma pista. Si el MIDI era tipo 1, se procede a buscar si el *Pogram Change* de alguna pista es 24 o 25. Como se mencionó en la sección 2.5, en la lista de instrumentos MIDI, la posición 25 corresponde a la guitarra acústica de cuerdas de nailon y el 26 corresponde a la guitarra acústica de cuerdas de metal, pero la biblioteca Mido maneja 0 como el primer instrumento de la lista, así que 24, en el programa, vendría siendo la guitarra acústica de cuerdas de nailon y 25 la de cuerdas de metal. Si se identifica que alguna de las pistas tiene 24 o 25 en su *Pogram Change*, se crea un MIDI nuevo, luego se copia la cantidad de *ticks* por *beat* del MIDI original al nuevo, se agregan las pistas 0 y de la o las guitarras acústicas al nuevo MIDI y se guarda el archivo MIDI nuevo. Todo este proceso se podía hacer de manera manual con un DAW llamado Reaper, pero con el programa de Python se volvió un proceso más automatizado.

Luego de tener el archivo MIDI correspondiente a la o las guitarras acústicas separadas, se convierte dicho archivo a formato WAV; y lo mismo se hace con el MIDI original. Para ello se utilizaron las herramientas en línea *Online-Convert* [59] y *Free-Convert* [60]. Las características de los audios fueron igual que como se exportaba de Ableton Live: en formato WAV, estéreo, a 24 bits de resolución, a una frecuencia de muestreo de 48 kHz.

Todo este proceso, desde el programa de Python hasta la conversión a WAV, se repitió para varios archivos MIDI. Adicionalmente se agregaron 23 canciones con instrumentos reales para que los modelos sean más robustos ante canciones reales y no se generalice solo para canciones hechas a base de archivos MIDI. La base de datos final se conformó entonces por las 23 canciones hechas con Ableton, las canciones obtenidas con el proceso semi automático y las 23 canciones con instrumentos reales. En total se obtuvieron 205 canciones las cuales se separaron en 160 canciones para el entrenamiento y 45 para la validación, procurando que las canciones con instrumentos reales estén en el subconjunto de entrenamiento. Todas estas canciones se convirtieron a una frecuencia de muestreo de 44.1 kHz utilizando el *framework* de multimedia FFMPEG. Así se obtuvieron 2 bases de datos con las mismas canciones pero de diferentes frecuencia de muestreo, de 48 kHz y de 44.1 kHz.

Las 160 canciones del subconjunto de entrenamiento se guardaron en una carpeta llamada *train* y las 45 canciones del subconjunto de validación se guardaron en una carpeta llamada *valid*. A su vez en cada una de estas carpetas se organizaron las canciones de la siguiente manera:

/train/canción1/Mix.wav
/train/canción1/AG.wav
/train/canción2/Mix.wav
/train/canción2/AG.wav
...
/valid/canción1/mixture.wav
/valid/canción1/vocals.wav
/valid/canción2/mixture.wav
/valid/canción2/vocals.wav
...

Donde "canción1" es el nombre de la primera canción dentro de la carpeta, "canción2" es el nombre de la segunda canción y así para todas las demás canciones; Mix.wav es la mezcla y AG.wav es la pista de la guitarra acústica.

Esta organización fue necesaria para que se carguen de manera correcta ambos subconjuntos de la base de datos en los programas para el entrenamiento de los modelos.

3.2. Programación

Luego de tener lista la base de datos, lo que sigue en la metodología es realizar la programación necesaria para obtener los modelos de aprendizaje profundo que sean capaces de realizar la separación de la guitarra acústica.

3.2.1. Recursos computacionales utilizados

Mi asesor me proporcionó el acceso a un servidor para utilizar para la programación y los entrenamientos, con las siguientes características:

- Procesador Intel core i7 8700k.
- 2 tarjetas gráficas NVIDIA GTX 1080 Ti.
- 64 GB de RAM.

Para la programación del entrenamiento de los modelos se utilizaron dos repositorios de separación de fuentes que facilitaron la codificación; el primero es el repositorio de Open-Unmix [23] y el segundo es Asteroid [61]. El de Open-Unmix solo maneja la arquitectura propuesta en su artículo [23], pero se puede entrenar con una base de datos propia y obtener las inferencias dada las canciones y los modelos. Asteroid es un repositorio de separación de fuentes más completo; de hecho Asteroid es un acrónimo para *Audio Source Separation on Steroids*. Asteroid maneja varias arquitecturas, diferentes bases de datos (pero se pueden implementar clases donde se cargue una base de datos propia), varias funciones de pérdidas y se pueden obtener las inferencias, dado los modelos entrenados y las canciones.

Ambos repositorios se clonaron al servidor y se creó un ambiente específico para cada uno, instalando los requerimientos necesarios para cada repositorio; pudiendo así trabajar con cada recurso en su propio ambiente.

Toda la programación se hizo con el lenguaje de Python y con los *frameworks* de aprendizaje profundo Pytorch [62] y Pytorch Lightning [63]. Para el manejo de las señales de audio se utilizaron las bibliotecas Torchaudio [64] en los códigos del entrenamiento y librosa [65] en los códigos de la evaluación de las inferencias.

Para la programación utilizando Open-Unmix, primero se leyó toda la documentación necesaria para poder realizar el entrenamiento de modelos utilizando una base de datos propia. En el repositorio de Open-Unmix hay una sección de entrenamiento donde se explica todo lo necesario que se debe realizar para poder entrenar modelos propios. Los demás detalles se explicarán en la siguiente sección. Este modelo puede trabajar con audios en estéreo, es decir con dos canales (izquierdo y derecho).

Para la programación utilizando Asteroid, primeramente, se leyó también toda la documentación necesaria. Luego se empezó a programar el código para realizar los entrenamientos de los modelos, se tuvieron que hacer varias pruebas y hacer algunas preguntas en el foro del repositorio para aclarar dudas de la codificación e ir mejorando y depurando el código. Los modelos en Asteroid trabajan solamente en formato mono, es decir con un solo canal; por lo que en la clase que carga la base de datos se utilizaba solo el primer canal de los audios

Para obtener inferencias, cada repositorio tiene implementado un comando específico para esa tarea; pero este proceso puede llegar a ser muy tedioso cuando se tienen muchos modelos entrenados y varias canciones a separar; entonces para acelerar los procesos se creó un código en Python que obtiene inferencias de varias canciones por cada modelo entrenado. Los demás detalles de como se obtuvieron las inferencias, se explican en la sección 3.4.

También se programó el código en Python para la obtención de las métricas de evaluación utilizando la librería Museval [66], para obtener (para cada inferencia de cada modelo) los valores de:

- Relación Señal-Interferencia (*Signal to Interference Ratio*, SIR)
- Relación Señal-Distorsión (*Signal to Distortion Ratio*, SDR)
- Relación Señal-Artefactos (*Signal to Artifacts Ratio*, SAR)

Todos los códigos se corrieron en el servidor utilizando la conexión remota a través de una terminal en Ubuntu.

3.3. Entrenamiento de los modelos de aprendizaje profundo

Al tener los programas listos, lo que sigue es el entrenamiento de los modelos de aprendizaje profundo. Para ello fue de mucho provecho las tarjetas gráficas que el servidor tiene.

3.3.1. Entrenamiento de modelos utilizando Open-Unmix

La codificación y entrenamiento utilizando Open-Unmix fue sencilla ya que el repositorio tiene casi todo preparado para el entrenamiento. Primero se activa el ambiente correspondiente a Open-Unmix, luego se corre el código *"train.py"* que está en la carpeta *"scripts"* del repositorio, agregando los argumentos necesarios. Entonces en la terminal, posicionándose dentro de la carpeta del repositorio se escribía algo como lo siguiente:

"python3 scripts/train.py --dataset aligned --root /home/ar/Data/EdgarSantos/DB44-1kHz24bitsAlignd/ --input-file Mix.wav --output-file AG.wav --output ./Model6 --epochs 1500 --batch-size 24 --patience 200 --seq-dur 16 --lr 0.001 --bandwidth 20000 --nb-workers 8"

El significado de cada argumento se muestra en la tabla 3.1. El número de trabajadores le indica al *Dataloader* cuantos subprocesos realizar para cargar los datos y generar los lotes en paralelo; mientras mayor sea este número, la CPU cargará los datos de una manera más eficiente; pero el valor se limita a la cantidad de núcleos que tiene el CPU.

Adicionalmente se debe escoger si se quiere trabajar con una o ambas tarjetas gráficas, para este caso, esto se logra agregando el comando *"cuda_visible_devices"* antes de escribir los comandos anteriores. Si debe escribir específicamente:

"*cuda_visible_devices*=0"

"*cuda_visible_devices*=1"

"*cuda_visible_devices*=0,1"

si se quiere utilizar (respectivamente) la primera tarjeta gráfica, la segunda tarjeta gráfica o ambas tarjetas.

Durante el entrenamiento, en la terminal se va mostrando el avance transcurrido a lo largo del tiempo. Al terminar el entrenamiento de un modelo, se generan cuatro archivos

Argumento	descripción
--dataset	Define la organización de archivos en la carpeta de la base de datos
--root	Ruta en disco donde se guardó la base de datos
--input-file	Archivo de audio de la mezcla de entrada a la red
--output-file	Archivo de audio de la fuente separada *ground truth*
--output	Ruta de la carpeta donde se guardarán los archivos generados
--epochs	Número de épocas de entrenamiento
--batch-size	Tamaño del lote
--patience	Número de épocas de validación para el paro temprano
--seq-dur	Duración en segundos de los extractos tomados de las canciones
--lr	Factor de aprendizaje
--bandwidth	Ancho de banda en Hz que procesarán las capas BLSTM
--nb-workers	Número de trabajadores paralelos para el *Dataloader*

Tabla 3.1: Significado de los argumentos para el entrenamiento con Open-Unmix.

dentro de la carpeta especificada en el argumento --output:

- El archivo con los Checkpoints.
- El archivo que contiene información del modelo como son los argumentos, épocas entrenadas, e historiales.
- El archivo del modelo.
- El archivo que contiene información del separador.

3.3.2. Entrenamiento de modelos utilizando Asteroid

Inicialmente en el código debe escogerse la arquitectura a entrenar y modificar las características correspondientes a los diferentes modelos. Para iniciar el entrenamiento, primero se activa el ambiente correspondiente a Asteroid, luego se corre el código que se creó para el entrenamiento ("TrainWAsteroid.py"), agregando los argumentos requeridos. Entonces en la terminal, posicionándose dentro de la carpeta del repositorio se escribía algo como lo siguiente:

"python3 scripts/TrainWAsteroid.py –segs 5 –sampRate 44100 –batchSize 2 –nWorkers 10 –lr 0.01 –patienc 250 –epochs 800 –modlNam DPTNetM7"

El significado de cada argumento se muestra en la tabla 3.2.

Para escoger con cuantas tarjetas gráficas se quiere trabajar, en el código se especifica "gpus=1" o "gpus=2" como argumento del objeto *Trainer* de *Pytorch Lighting*.

Durante el entrenamiento, en la terminal se va mostrando el avance transcurrido a lo largo del tiempo. Al terminar el entrenamiento de un modelo, dentro de la carpeta especificada en el argumento --modlNam, se genera lo siguiente:

- Una carpeta con los archivos de los k Checkpoints.
- El archivo que contiene los argumentos especificados.
- El archivo que indica la ruta y valor de pérdida de los k mejores modelos.
- El archivo del mejor modelo entrenado.

Donde k es una cantidad especificada en el código.

Argumento	descripción
--segs	Duración en segundos de los extractos tomados de las canciones.
--sampRate	Frecuencia de muestreo en Hz.
--batchSize	Tamaño del lote.
--nWorkers	Número de trabajadores paralelos para el *Dataloader*.
--lr	Factor de aprendizaje.
--patienc	Número de épocas de validación para el paro temprano.
--epochs	Número de épocas de entrenamiento.
--modlNam	Nombre del modelo y ruta donde se guardarán los archivos generados.

Tabla 3.2: Significado de los argumentos para el entrenamiento con Asteroid.

En el objeto *Trainer* también se tiene un argumento llamado *accelerator* con el cual se escoge la configuración del entrenamiento utilizando múltiples GPU's. En este argumento se fue variando entre DDP y DP.

DDP quiere decir *Distributed Data Parallel*. Con este paradigma, cada GPU recibe una copia del modelo y un subconjunto de los datos, cada GPU entrena con su propio subconjunto pero al final de cada época se sincronizan los gradientes a través de todas las GPU's. DDP se recomienda utilizar cuando se entrena con diferentes máquinas con varias GPU's cada una.

DP quiere decir *Data Parallel*. Con este paradigma, cada GPU recibe una copia del modelo y toda la base de datos se divide entre las GPU's disponibles, luego se realiza el proceso hacia adelante del entrenamiento en cada GPU y para actualizar los pesos del modelo,

las salidas se procesan hacia atrás en una sola GPU. En la siguiente época se copia el modelo actualizado en las GPU's de nuevo. Debido a los movimientos de las salidas y las copias de los modelos, DP es más lento que DDP pero es recomendado cuando se entrena con solo una máquina con múltiples GPU's; y en la práctica, DP consume menos memoria de la GPU que DDP.

3.4. Obtención de inferencias

Para obtener las inferencias, los repositorios tienen implementado un comando específico para esa tarea.

En el caso de Open-Unmix, después del comando se debe dar como argumentos la ruta de la canción a separar, la ruta donde se encuentra la carpeta del modelo, qué pista se quiere separar, especificar si se quiere guardar el audio residual y la ruta donde se guardarán las separaciones. Después de correr el comando para la separación, se genera una carpeta con dos archivos de audio (en formato WAV a 44.1 kHz de frecuencia de muestreo): El audio de la separación de la guitarra acústica y el audio residual (la canción sin la guitarra acústica). El nombre de la carpeta generada es el mismo que el nombre del archivo de audio original; luego, manualmente, esta carpeta se debe cambiar de nombre al nombre de la canción más el nombre del modelo para que no se confunda con futuras inferencias. Por último, esta carpeta se mueve a una carpeta con el nombre de la arquitectura.

En el caso de Asteroid, después del comando de inferencia, se debe dar como argumentos la ruta donde se encuentra el archivo del mejor modelo entrenado, la ruta de la canción a separar, especificar si se quiere reescribir la inferencia, especificar si se quiere resamplear el archivo de entrada (en caso de tener diferente frecuencia de muestreo), el tamaño de la ventana para el sobrelape y suma, la ruta donde se guardará la separación y el dispositivo para realizar la separación (cuda:0 o cuda:1 si se quiere usar, respectivamente, la primera tarjeta gráfica o la segunda). Después de correr el comando, se genera el archivo de audio de la separación de la guitarra acústica en formato WAV a 44.1 kHz de frecuencia de muestreo, con el nombre de la mezcla original más "_est1". El nombre de este archivo se debe cambiar manualmente al nombre de la canción más el nombre del modelo para que no se confunda

con futuras inferencias. Así mismo, el archivo de audio debe moverse a una carpeta con el nombre del tipo de arquitectura entrenada; se recuerda que con Asteroid se entrenaron varias arquitecturas y que trabajan solo con audios en formato mono, por lo que las inferencias solo son de un canal.

Todos estos procesos pueden llegar a ser muy tediosos cuando se tienen muchos modelos entrenados y varias canciones a inferir; entonces para acelerar los procesos se creó un código en Python para cada repositorio. Estos códigos obtienen las inferencias de varias canciones de varios modelos entrenados y realizan el cambio de nombre y movimiento de carpetas, todo completamente automatizado, y se puede escoger qué canciones, de las disponibles, se pueden inferir y también se puede escoger de qué modelos de todos los entrenados se quiere inferir.

Capítulo 4

Resultados

Los resultados del proyecto comprenden: los modelos entrenados, las inferencias y las evaluaciones de acuerdo a las inferencias. Ya se ha explicado cómo se entrenaron los modelos de aprendizaje profundo y también cómo se obtuvieron las inferencias.

Se entrenaron 84 modelos en total, de cuatro arquitecturas diferentes, con diversos hiperparámetros. Las arquitecturas utilizadas son las siguientes:

- Open-Unmix [23].
- DCUNet [50].
- SudoRMRF [31].
- DPTNet [33].

Se seleccionaron estas cuatro arquitecturas por una combinación de practicidad (ya que estos modelos entrenados están públicamente disponibles), simplicidad en recursos computacionales (ya que para los entrenamientos no se contó con servidores de gran capacidad computacional como en el estado del arte) y su relación al tipo de fuente que están separando. Así mismo se escogieron estos 4 modelos ya que se quería probar arquitecturas desarrolladas recientemente, las cuales están basadas en capas BLSTM, capas convolucionales y Transformers. También se optó por modelos que ya pueden separar instrumentos musicales, por lo que ya consideran los cambios tempo-frecuenciales de éstos, lo cual se prevé que pueda ayudar a la separación de la guitarra. Y se optó por modelos que ya pueden

separar la voz, ya que están enfocados a una región frecuencial que tiene un gran sobrelape con la región que ocupa la guitarra.

Los dos primeros modelos de Open-Unmix fueron entrenados con la base de datos de 48 kHz, al generar unas cuantas inferencias de estos modelos se obtuvieron muy malos resultados por lo que no se incluyeron como parte de las evaluaciones. Después de entrenar estos modelos, se decidió entrenar todos los demás modelos con la base de datos de 44.1 kHz ya que se ocupa menos memoria computacional y el ancho de banda audible está dentro del rango auditivo humano.

En el caso de las inferencias, en total se obtuvieron 656 de todos los modelos entrenados. Estas inferencias se utilizaron para las evaluaciones de los modelos. En total por cada modelo se infirieron 8 canciones.

En la siguiente sección se hablará de las características e hiperparámetros que se utilizaron para entrenar los modelos.

4.1. Características e hiperparámetros

Las arquitecturas entrenadas tuvieron diversas características, especificaciones e hiperparámetros de entrenamiento. Por ejemplo: se variaba el tamaño del lote, la tasa de aprendizaje y/o la frecuencia de muestreo en cada entrenamiento. Para Open-Unmix se entrenaron modelos de 3, 5 y 7 capas de BLSTM's . Las tablas se muestran por cada arquitectura utilizada. Algunas tablas contenían demasiados datos, por lo que se dividieron en 2 partes.

En las tablas 4.1 se muestran los hiperparámetros de entrenamiento de los modelos con la arquitectura Open-Unmix de 3 capas de BLSTM's.

En la tabla 4.2 se muestran los hiperparámetros de entrenamiento de los modelos con la arquitectura Open-Unmix de 5 capas de BLSTM's.

En la tabla 4.3 se muestran los hiperparámetros de entrenamiento de los modelos con la arquitectura Open-Unmix de 7 capas de BLSTM's.

En las tablas 4.4 y 4.5 se muestran los hiperparámetros de entrenamiento y características de los modelos con la arquitectura DCUNet.

En las tablas 4.6 y 4.7 se muestran los hiperparámetros de entrenamiento de los modelos

con la arquitectura SudoRMRF.

En las tablas 4.8 y 4.9 se muestran los hiperparámetros de entrenamiento de los modelos con la arquitectura DPTNet.

Los modelos que inician con la letra P fueron modelos de prueba para conocer con qué hiperparámetros se pueden entrenar de acuerdo a las capacidades del servidor.

Modelo	Frec. de muestreo de BD (kHz)	Épocas de entrenamiento	Paro temprano	Tamaño del lote	Segs. por canción	Tasa de aprendizaje	Ancho de banda (kHz)	No. de trabajadores	Ventana de sobrelape y suma para inferencia
P1	48	500	140	1	6	0.002	16	3	4096
P2	48	1000	2400	1	6	0.001	16	4	4096
P3	44.1	1000	2000	8	10	0.001	20	4	4096
P4	44.1	1000	2000	16	10	0.001	20	4	4096
M5	44.1	1500	1000	16	15	0.001	20	4	4096
M6	44.1	1500	1000	24	16	0.0001	20	4	4096
M7	44.1	1500	1000	32	16	0.002	20	4	4096
M8	44.1	2500	1000	32	16	0.0005	20	5	4096
M9	44.1	2500	1000	8	16	0.0005	20	8	4096
M10	44.1	2500	800	24	16	0.0005	20	4	4096
M11	44.1	2500	650	32	16	0.001	20	8	4096
M12	44.1	1000	650	32	16	0.005	20	8	4096
M13	44.1	2500	850	32	16	0.0001	20	8	4096
M14	44.1	1500	650	32	16	0.002	20	8	4096
M15	44.1	2500	850	32	16	0.0003	20	8	4096

Tabla 4.1: Hiperparámetros de entrenamiento de modelos con arquitectura Open-Unmix de 3 capas de BLSTM's.

Modelo	Épocas de entrenamiento	Paro temprano	Tamaño del lote	Segs. por canción	Tasa de aprendizaje	Ancho de banda (kHz)	No. de trabajadores	Ventana de sobrelape y suma para inferencia
M1	2500	100	32	15	0.001	20	8	4096
M2	2500	150	32	15	0.005	20	8	4096
M3	2000	600	32	16	0.0005	20	8	4096
M4	2500	800	32	16	0.0001	20	8	4096
M5	1000	250	32	16	0.01	20	8	4096

Tabla 4.2: Hiperparámetros de entrenamiento de modelos con arquitectura Open-Unmix de 5 capas de BLSTM's.

Modelo	Épocas de entrenamiento	Paro temprano	Tamaño del lote	Segs. por canción	Tasa de aprendizaje	Ancho de banda (kHz)	No. de trabajadores	Ventana de sobrelape y suma para inferencia
M1	1000	200	32	16	0.01	20	8	4096
M2	1500	300	32	16	0.005	20	8	4096
M3	2000	600	32	16	0.001	20	8	4096
M4	2500	650	32	16	0.0005	20	8	4096
M5	2500	800	32	16	0.0001	20	8	4096

Tabla 4.3: Hiperparámetros de entrenamiento de modelos con arquitectura de Open-Unmix de 7 capas de BLSTM's.

Modelo	Tipo de modelo DCUNET	Épocas de entrenamiento	Paro temprano	Tamaño del lote	Segs. por canción	Tasa de aprendizaje	No. de trabajadores	Función de pérdida
P1	DCUNet-10	1000	200	16	6	0.001	8	pairwise neg sisdr
P2	DCUNet-16	1000	150	8	11	0.001	8	pairwise mse
P3	DCUNet-20	1000	160	4	4	0.001	8	pairwise mse
P4	DCUNet-20	1000	400	4	4	0.001	8	pairwise neg sisdr
P5	Large DCUNet-20	1000	450	2	4	0.0001	10	pairwise neg sisdr
P6	DCUNet-20	1000	150	6	3	0.001	8	pairwise neg sisdr
M1	DCUNet-20	1000	250	2	4	0.001	10	pairwise mse
M2	DCUNet-16	800	240	4	6	0.001	10	pairwise neg sisdr
M3	DCUNet-16	650	140	4	6	0.001	10	pairwise mse
M4	DCUNet-20	1000	450	4	3	0.001	10	pairwise mse
M5	Large DCUNet-20	1000	450	2	5	0.001	10	pairwise mse
M6	Large DCUNet-20	1000	400	8	3	0.001	10	pairwise mse
M7	DCUNet-20	1000	400	12	3.8	0.001	10	pairwise mse
M8	DCUNet-20	1000	400	6	3.8	0.001	10	pairwise mse
M9	DCUNet-20	800	240	6	3.5	0.01	10	singlesrc mse
M10	Large DCUNet-20	850	240	4	2.8	0.01	10	singlesrc mse
M11	DCUNet-20	850	240	2	4.8	0.01	10	singlesrc mse

Tabla 4.4: Hiperparámetros y características de modelos con arquitectura DCUNet parte 1.

Modelo	Frecuencia de muestreo (kHz)	No. de filtros stft	Tamaño del kernel stft	*Stride* stft	Acelerador	Ventana de sobrelape y suma para inferencia
P1	44.1	512	1024	256	DDP	4096
P2	44.1	512	1024	256	DDP	4096
P3	44.1	512	1024	256	DDP	4096
P4	44.1	512	1024	256	DDP	4096
P5	44.1	512	1024	256	DDP	4096
P6	16	512	1024	256	DDP	4096
M1	44.1	1024	1024	256	DDP	4096
M2	40	2048	1024	256	DDP	4096
M3	40	2048	2048	256	DDP	4096
M4	40	1024	1024	256	DP	5200
M5	36	1024	1024	256	DP	5200
M6	32	1024	1024	512	DP	5200
M7	32	1024	1024	512	DP	5200
M8	32	2048	1024	512	DP	5200
M9	35	2048	1024	512	DP	5000
M10	34.1	1024	1024	256	DP	5000
M11	39.5	2048	1024	256	DP	5000

Tabla 4.5: Hiperparámetros y características de modelos con arquitectura DCUNet parte 2.

Modelo	Épocas de entrenamiento	Paro temprano	Tamaño del lote	Segs. por canción	Tasa de aprendizaje	No. de trabajadores	Función de pérdida	Frecuencia de muestreo (kHz)	Canales de entrada de los *UConvBlocks*	Cantidad de bloques convolucionales
P1	500	150	8	6	0.001	10	pairwise mse	44.1	32	4
P2	800	240	4	7	0.001	10	pairwise mse	44.1	64	4
P3	1000	400	4	4	0.001	10	pairwise mse	44.1	64	8
M1	1000	360	4	4	0.001	10	pairwise mse	44.1	128	8
M2	1000	360	2	4	0.001	10	pairwise mse	44.1	64	8
M3	1000	360	4	4	0.001	10	pairwise mse	44.1	128	4
M4	1000	360	8	4	0.001	10	pairwise mse	44.1	128	4
M5	1000	400	4	4	0.001	10	pairwise mse	40	128	4
M6	1000	400	8	4	0.001	10	pairwise mse	40	128	4
M7	1000	400	6	4	0.001	10	pairwise mse	40	64	16
M8	1000	400	16	5	0.001	10	pairwise mse	40	64	4
M9	1000	450	2	4	0.001	10	pairwise mse	42	128	16
M10	1000	450	2	5.2	0.001	10	pairwise mse	32	128	16
M11	1000	400	6	3.6	0.001	10	pairwise mse	32	128	8
M12	850	450	2	4.4	0.01	10	singlesrc mse	32	128	8
M13	850	400	6	3	0.01	10	singlesrc mse	34	512	8
M14	850	240	2	4.6	0.01	10	singlesrc mse	36	128	8

Tabla 4.6: Hiperparámetros y características de modelos con arquitectura SudoRMRF parte 1.

Modelo	Profundidad de resampleado	Tamaño del kernel	No. de filtros	*Stride*	Acelerador	Ventana de sobrelape y suma para inferencia	*Gradient clipping*
P1	2	21	256	25	DDP	5000	None
P2	2	21	256	10	DDP	5000	None
P3	2	21	256	10	DDP	5000	None
M1	2	21	256	10	DDP	5000	None
M2	2	21	512	10	DDP	5000	None
M3	2	21	512	10	DDP	5000	None
M4	4	21	512	25	DDP	5200	None
M5	4	21	512	10	DDP	5200	None
M6	2	21	256	10	DDP	5000	None
M7	2	21	256	25	DDP	5000	None
M8	2	21	256	25	DDP	5000	None
M9	4	21	512	10	DP	5200	None
M10	4	21	512	10	DP	5200	None
M11	4	21	512	10	DP	5200	None
M12	4	21	1024	10	DP	5200	5
M13	4	21	512	10	DP	5200	5
M14	4	21	1024	10	DP	5200	None

Tabla 4.7: Hiperparámetros y características de modelos con arquitectura SudoRMRF parte 2.

Modelo	Épocas de entrenamiento	Paro temprano	Tamaño del lote	Segs. por canción	Tasa de aprendizaje	No. de trabajadores	Función de pérdida	Frecuencia de muestreo (kHz)	No. de cabezas	No. de neuronas en la RNN	Ventana de sobrelape y suma en la red
P1	600	150	6	5	0.001	10	pairwise_mse	44.1	4	256	100
P2	600	150	4	3.5	0.001	10	pairwise_mse	44.1	4	256	100
P3	600	150	2	2.8	0.001	10	pairwise_mse	44.1	4	256	100
M1	600	200	2	2.8	0.001	10	pairwise_mse	36	4	128	100
M2	600	240	4	4	0.001	10	pairwise_mse	36	6	240	100
M3	800	250	2	3	0.001	10	pairwise_mse	41	4	256	100
M4	800	250	2	3.4	0.001	10	pairwise_mse	36.2	4	256	250
M5	800	250	4	3.6	0.1	10	pairwise_mse	38	4	256	100
M6	800	250	4	3.8	0.01	10	pairwise_mse	36.5	4	250	100
M7	800	250	2	4.8	0.01	10	pairwise_mse	38	4	250	100
M8	800	250	2	3.3	0.01	10	pairwise_mse	38	4	250	100
M9	800	250	2	3.5	0.01	10	pairwise_mse	36	4	240	100
M10	800	250	2	3.5	0.01	10	pairwise_mse	36	4	240	100
M11	800	250	4	3.6	0.01	10	pairwise_mse	35.5	4	240	100
M12	800	200	4	3.6	0.001	10	singlesrc_mse	35.5	4	240	100
M13	800	200	2	4.3	0.001	10	singlesrc_mse	36.5	5	240	100
M14	800	200	2	4	0.001	10	singlesrc_mse	36.5	6	240	100
M15	650	250	4	3.6	0.001	10	singlesrc_mse	36	8	256	100
M16	650	250	2	3.8	0.001	10	singlesrc_mse	36.6	8	256	100
M17	650	250	2	3	0.001	10	singlesrc_mse	34.2	8	256	100
M18	400	150	4	3.6	0.000015	10	singlesrc_mse	36	8	256	100
M19	400	150	2	3.8	0.000015	10	singlesrc_mse	36.7	8	256	100
M20	400	150	2	3.8	0.000015	10	singlesrc_mse	36.7	8	256	100
M21	400	150	2	3.8	0.0001	10	singlesrc_mse	36.7	8	256	100
M22	400	150	2	3.8	0.0001	10	singlesrc_mse	36.7	8	256	100

Tabla 4.8: Hiperparámetros y características de modelos con arquitectura DPTNet parte 1.

Modelo	Tamaño del traslado de ventana	No. de repeticiones del Transformer dual	Función de activación del enmascaramiento	No. de canales de entrada	Tamaño del kernel	No. de filtros	*Stride*	No. de *bins* de salida de la máscara	*Gradient clipping*	Acelerador	Ventana de sobrelape y suma para inferencia
P1	50	2	sigmoid	64	32	64	16	64	5	DP	5000
P2	50	2	sigmoid	64	16	64	8	64	5	DP	5000
P3	None	6	relu	None	16	64	8	None	None	DP	5000
M1	None	6	sigmoid	None	10	64	5	None	None	DP	5000
M2	None	2	relu	None	16	64	8	None	None	DP	5000
M3	None	6	relu	64	16	64	8	64	None	DP	5000
M4	None	6	relu	None	16	64	8	None	None	DP	5000
M5	None	4	relu	None	24	64	12	None	5	DP	5000
M6	None	4	relu	None	24	64	12	None	5	DP	5000
M7	None	6	sigmoid	None	24	60	12	None	5	DP	5000
M8	None	6	sigmoid	None	16	60	8	None	None	DP	5000
M9	None	6	relu	64	16	64	8	64	None	DP	5000
M10	None	6	sigmoid	64	16	64	8	64	None	DP	5000
M11	50	4	sigmoid	None	24	140	12	None	5	DP	5000
M12	50	4	sigmoid	140	24	140	12	140	5	DP	5000
M13	None	6	relu	150	24	150	12	150	5	DP	5000
M14	None	4	relu	None	24	150	12	None	None	DP	5000
M15	None	2	relu	None	16	64	8	None	None	DP	5000
M16	None	4	relu	None	16	64	8	None	None	DP	5000
M17	None	6	relu	None	16	64	8	None	None	DP	5000
M18	None	2	sigmoid	None	16	64	8	None	5	DP	5000
M19	None	4	sigmoid	None	16	64	8	None	5	DP	5000
M20	None	4	relu	None	16	64	8	None	5	DP	5000
M21	None	4	sigmoid	None	16	64	8	None	None	DP	5000
M22	None	4	relu	None	16	64	8	None	5	DP	5000

Tabla 4.9: Hiperparámetros y características de modelos con arquitectura DPTNet parte 2.

En la siguiente sección se hablará de cómo se realizaron las evaluaciones de los modelos.

4.2. Evaluación de los modelos entrenados

Como se mencionó en el capítulo anterior, se programó un código para la obtención de las métricas de evaluación (SIR, SDR y SAR) [21]. Específicamente se creó una libreta de Jupyter para los modelos entrenados con Open-Unmix y otra libreta para los modelos entrenados con Asteroid. Esto debido a la organización de rutas donde se guardaron las inferencias de ambos repositorios.

Para leer los archivos de audio se utilizó la función *load* de una paquetería de análisis de audio llamada Librosa [65] con la cual, los audios cargados se guardan en un arreglo de de dimensiones [2, n] cuando se trata de un audio en estéreo, donde n es el tamaño total de *bins* de tiempo (este valor es la duración en segundos de la canción, multiplicado por su frecuencia de muestreo) y cuando la canción cargada es monofónica el arreglo es de dimensión [n].

Para obtener las métricas (SIR, SDR y SAR) se utiliza una librería llamada Museval [66]. En esta librería se tiene la función "bss_eval" la cual lleva por argumentos las referencias (arreglo de fuentes originales), las estimaciones (arreglo de inferencias) y el tamaño de ventana (duración de la canción, en nuestro caso se colocó [n]). En el arreglo de referencias, en el primer renglón debe estar un canal de la pista de la guitarra acústica original y en el segundo renglón debe ir un canal de la mezcla de la canción original sin la guitarra acústica. En el arreglo de estimaciones, en el primer renglón debe estar un canal de la inferencia (la guitarra acústica estimada) y en el segundo canal debería ir la inferencia sin la guitarra acústica (el audio residual de la inferencia). Asteroid no genera el audio residual al obtener las inferencias, por lo que en éste último renglón se utilizó la mezcla original. Se recuerda que los valores de SIR, SDR y SAR se presentan en decibeles.

La función "bss_eval" se debe correr por cada una de las ocho canciones por cada modelo entrenado. Por lo que se utilizó un ciclo *for* dentro de otro para evaluar cada una de las canciones para cada modelo. Al final los resultados son organizados y desplegados en tablas utilizando la librería Pandas [67].

4.3. Resultados de las evaluaciones

A continuación se muestran las tablas con todos los resultados de las evaluaciones de 8 canciones por cada modelo. Así como en las tablas de los hiperparámetros, los resultados se muestran por cada arquitectura; pero también se dividen por tipo de canción. Las tres primeras canciones son hechas a base de MIDI (como la mayoría de canciones en la base de datos). Y las demás canciones fueron grabadas con instrumentos reales.

En las tablas 4.10 y 4.11 se muestran los resultados de los modelos con la arquitectura Open-Unmix de 3 capas de BLSTM's.

En las tablas 4.12 y 4.13 se muestran los resultados de los modelos con la arquitectura Open-Unmix de 5 capas de BLSTM's.

En las tablas 4.14 y 4.15 se muestran los resultados de los modelos con la arquitectura Open-Unmix de 7 capas de BLSTM's.

En las tablas 4.16, 4.17, 4.18 y 4.19 se muestran los resultados de los modelos con la arquitectura DCUNet.

En las tablas 4.20, 4.21, 4.22 y 4.23 se muestran los resultados de los modelos con la arquitectura SudoRMRF.

En las tablas 4.24, 4.25, 4.26 y 4.27 se muestran los resultados de los modelos con la arquitectura DPTNet.

Recordemos que los valores presentados están en decibeles (dB) y que mientras el valor sea mayor, mejor será el desempeño. En esta sección solo se presentan los resultados numéricos, en la sección 4.4 se discute si todos estos valores son buenos o malos.

Modelo	Métrica	TeDareLoMejor	TheHeartOfLife	TuEstasAqui
P3	SIR	-0.634415	-2.947	-3.450807
	SDR	0.621256	-0.190042	-0.57893
	SAR	0.342149	-2.19197	-0.675151
P4	SIR	2.08905	-6.180352	-3.177189
	SDR	0.7702	-0.2286	-0.024332
	SAR	-2.542807	-2.509145	-1.197761
M5	SIR	0.644816	-3.663889	-4.053224
	SDR	0.865658	-0.19294	-0.867104
	SAR	-0.442464	-1.645664	6.19E-03
M6	SIR	-1.235416	-1.896728	-4.973493
	SDR	0.082095	-0.711393	-1.806678
	SAR	-1.52148	-0.049619	0.501215
M7	SIR	-1.590219	-5.199157	-4.175478
	SDR	0.075114	-0.637858	-0.942623
	SAR	1.334021	0.069145	0.919733
M8	SIR	1.005565	-4.31483	-3.334416
	SDR	0.872763	-0.758237	-0.704073
	SAR	-0.808348	-0.643059	-0.658665
M9	SIR	-0.446515	-5.40611	-3.045793
	SDR	0.287165	-0.326009	-0.598055
	SAR	0.857736	-3.507746	-0.733301
M10	SIR	0.546511	-3.441509	-3.24585
	SDR	0.762432	-0.768569	-0.635673
	SAR	-0.99724	-0.398358	-0.928691
M11	SIR	0.464391	-2.906335	-3.218171
	SDR	0.622223	-0.568909	-0.54417
	SAR	-0.567838	-0.146959	-0.746153
M12	SIR	-2.886905	-6.350127	-6.02395
	SDR	0.189685	-1.112103	-1.236314
	SAR	1.667885	1.352383	2.385956
M13	SIR	-2.120078	-2.915407	-4.641133
	SDR	-0.282089	-0.930812	-1.841591
	SAR	-0.879043	-0.232449	-0.015554
M14	SIR	-2.378488	-3.547917	-4.167955
	SDR	0.006377	-0.347875	-0.650674
	SAR	0.720631	-0.963855	1.040537
M15	SIR	0.968526	-3.296038	-3.713928
	SDR	0.686399	-0.69218	-1.019985
	SAR	-2.115554	-0.114715	-0.831474

Tabla 4.10: Evaluaciones de canciones hechas a base MIDI, correspondientes a los modelos con arquitectura Open-Unmix con 3 capas de BLSTM's.

Modelo	Métrica	Bearlin Roads	LizNelson Rainfall	QuePena TantoFaz 6-19	QuePena TantoFaz 46-57	ThIsAmazin Grace
P3	SIR	-4.640208	10.962505	10.246872	12.015539	-1.741526
	SDR	0.039902	0.115803	1.270224	1.247318	0.687323
	SAR	-1.474013	-8.286233	0.749585	1.15619	-0.384743
P4	SIR	-4.644532	11.485537	7.479928	13.473107	-0.549066
	SDR	0.071277	0.828624	1.707797	1.07829	0.64663
	SAR	-2.433663	-2.05505	1.940984	0.952758	-1.673876
M5	SIR	-3.674264	12.27728	9.817324	13.350953	-0.353317
	SDR	-0.46024	0.300428	2.195005	0.790797	0.597587
	SAR	-2.114901	-5.524855	3.111578	-1.548882	-3.1332
M6	SIR	-2.671876	11.256759	9.989387	9.615477	-0.583678
	SDR	-0.113469	1.065729	1.916366	1.174966	0.672836
	SAR	-2.892142	-1.635923	1.409459	0.118854	-2.193355
M7	SIR	-3.818472	9.358165	10.139433	10.466698	-2.049489
	SDR	0.116223	0.179578	1.256332	1.170725	0.621912
	SAR	-1.562574	-7.390688	0.753941	1.449659	-0.612383
M8	SIR	-2.113111	11.152387	7.482146	12.91534	-0.42469
	SDR	0.093676	0.857263	1.116242	1.103456	0.741089
	SAR	-3.728498	-2.908514	2.022338	-0.315198	-2.346941
M9	SIR	-3.028856	11.186695	12.214596	9.397328	-1.782084
	SDR	-0.50892	0.171565	0.876595	1.481996	0.452245
	SAR	-2.163073	-8.448964	-0.793377	0.272157	-1.289465
M10	SIR	-1.972445	11.276098	12.633506	10.710586	-0.71252
	SDR	-0.256217	0.636208	1.449291	0.985902	0.741829
	SAR	-3.447739	-3.498697	1.690624	-0.621144	-1.805145
M11	SIR	-4.999281	13.458853	6.709515	10.952307	-0.448081
	SDR	-0.685755	0.68381	1.107334	1.218058	0.715584
	SAR	-2.045306	-3.61124	2.005647	0.156803	-1.854869
M12	SIR	-3.896736	10.926977	11.253174	12.783924	-3.777439
	SDR	0.052176	1.55125	2.45116	1.312636	0.47523
	SAR	-0.900967	1.298709	3.749371	2.073451	1.60475
M13	SIR	-2.875406	9.621666	10.239594	10.563489	-0.846124
	SDR	0.025979	0.820872	1.71051	1.073449	0.741036
	SAR	-2.469123	-2.771083	1.687148	-0.064579	-1.589935
M14	SIR	-4.375177	11.710034	12.962938	13.216494	-2.585002
	SDR	0.112882	0.827489	2.148105	1.20307	0.650236
	SAR	-2.753918	-3.260764	3.192231	1.011454	-0.26972
M15	SIR	-1.904741	11.629041	8.671842	8.771093	0.402269
	SDR	-0.53846	0.704737	1.585616	1.328122	0.733096
	SAR	-2.680974	-3.838814	1.642166	1.104059	-2.816834

Tabla 4.11: Evaluaciones de canciones reales, correspondientes a los modelos con arquitectura Open-Unmix con 3 capas de BLSTM's.

Modelo	Métrica	TeDareLoMejor	TheHeartOfLife	TuEstasAqui
M1	SIR	-5.169739	-3.358166	-7.170179
	SDR	-4.304138	-0.968783	-5.934054
	SAR	4.065745	-1.00291	5.717989
M2	SIR	-3.228559	-4.483263	-6.479001
	SDR	0.173048	-0.261265	-1.259845
	SAR	1.763202	-0.497376	2.450578
M3	SIR	-5.08909	-4.449115	-7.197752
	SDR	-4.364827	-0.844482	-6.395596
	SAR	4.203198	-2.109264	5.826443
M4	SIR	-5.237307	-6.666511	-7.625753
	SDR	-4.312771	-2.06765	-6.981397
	SAR	4.407571	-0.258069	6.925655
M5	SIR	-3.346604	-4.489846	-6.489116
	SDR	-0.643332	-0.479387	-3.10542
	SAR	2.224064	-0.59827	3.367952

Tabla 4.12: Evaluaciones de canciones hechas a base MIDI, correspondientes a los modelos con arquitectura Open-Unmix con 5 capas de BLSTM's.

Modelo	Métrica	Bearlin Roads	LizNelson Rainfall	QuePena TantoFaz 6-19	QuePena TantoFaz 46-57	ThIsAmazin Grace
M1	SIR	-5.842305	11.470457	17.231954	11.119725	-5.680567
	SDR	-5.493609	3.004252	5.118452	2.967127	-3.467401
	SAR	2.778102	3.760966	9.029803	5.707954	3.633263
M2	SIR	-4.619902	8.902351	14.98995	11.213024	-3.975717
	SDR	-0.312667	1.273684	1.895348	1.168596	0.390322
	SAR	-0.646645	1.309175	2.347337	0.967677	0.872759
M3	SIR	-5.54215	8.665806	14.634435	11.013169	-5.018838
	SDR	-5.225385	2.108347	2.699888	1.878176	-1.632169
	SAR	1.666456	1.041587	3.294509	1.721901	1.28426
M4	SIR	-4.604003	7.361741	14.805227	13.052524	-4.623418
	SDR	-3.249873	2.745494	3.109056	1.901929	-1.553738
	SAR	1.672295	2.477697	4.009036	2.071687	3.14598
M5	SIR	-4.3043	10.444987	12.087282	8.369365	-2.592353
	SDR	-0.817162	2.353315	2.968504	1.685055	0.367768
	SAR	-0.178049	3.95795	3.863184	1.700543	1.110831

Tabla 4.13: Evaluaciones de canciones reales, correspondientes a los modelos con arquitectura Open-Unmix con 5 capas de BLSTM's.

Modelo	Métrica	TeDareLoMejor	TheHeartOfLife	TuEstasAqui
M1	SIR	-1.577626	-4.66818	-5.877003
	SDR	0.548572	-0.069712	-0.842546
	SAR	-0.724371	-3.61232	-0.37305
M2	SIR	-2.234502	-4.800376	-4.886609
	SDR	0.556803	-0.127928	-0.904011
	SAR	0.48325	-2.655105	0.344315
M3	SIR	-1.728207	0.868103	-5.256257
	SDR	0.556693	0.34754	-0.3948566
	SAR	-0.069626	-3.130344	-0.453957
M4	SIR	-5.562193	-3.374971	-7.31366
	SDR	-4.516489	-1.25035	-6.214226
	SAR	3.530514	-2.313265	5.456884
M5	SIR	-4.459998	-5.023088	-6.752566
	SDR	-3.320505	-2.154334	-6.200731
	SAR	3.420522	0.442252	5.650254

Tabla 4.14: Evaluaciones de canciones hechas a base MIDI, correspondientes a los modelos con arquitectura Open-Unmix con 7 capas de BLSTM's.

Modelo	Métrica	Bearlin Roads	LizNelson Rainfall	QuePena TantoFaz 6-19	QuePena TantoFaz 46-57	ThIsAmazin Grace
M1	SIR	-3.477983	9.616392	13.601139	12.439315	-5.002208
	SDR	0.153167	0.912045	1.887076	0.908464	0.31464
	SAR	-2.213432	-0.967613	1.482099	-0.899202	-0.824913
M2	SIR	-3.492308	10.201431	15.640626	12.776512	-3.101913
	SDR	0.120973	1.480061	2.232835	1.182625	0.55943
	SAR	-1.670215	1.035019	2.390484	0.357732	-0.280157
M3	SIR	-3.819999	7.590794	13.77882	10.749363	-3.112735
	SDR	-0.142932	0.641591	1.792276	1.051746	0.555732
	SAR	-0.314728	-1.530089	3.097496	0.90571	0.299724
M4	SIR	-4.908717	8.173602	12.536521	11.243721	-5.22489
	SDR	-4.991845	1.905653	3.234576	2.806708	-3.846937
	SAR	2.268267	1.273442	3.914321	3.524269	3.783036
M5	SIR	-5.169272	5.568091	10.620998	9.508149	-3.884008
	SDR	-4.010018	1.729927	3.879087	2.846882	-2.021124
	SAR	1.822691	1.687474	4.658286	3.946517	3.717331

Tabla 4.15: Evaluaciones de canciones reales, correspondientes a los modelos con arquitectura Open-Unmix con 7 capas de BLSTM's.

Modelo	Métrica	TeDareLoMejor	TheHeartOfLife	TuEstasAqui
P1	SIR	-9.925336	-11.405813	-9.79743
	SDR	-17.931459	-18.417316	-19.559192
	SAR	-12.602318	-12.52426	-12.311006
P2	SIR	-10.035887	-10.963786	-9.750737
	SDR	-17.732993	-18.038391	-19.194502
	SAR	-16.401765	-15.821684	-15.109983
P3	SIR	-4.023256	-6.814122	-5.969901
	SDR	-11.07449	-12.659739	-10.56082
	SAR	-9.312036	-8.340221	-8.225458
P4	SIR	-7.418208	-10.025112	-8.278611
	SDR	-10.650696	-12.159833	-9.918483
	SAR	5.837645	6.053726	6.392046
P5	SIR	-7.180932	-8.471139	-8.058537
	SDR	-10.629408	-12.340375	-9.925018
	SAR	5.795261	1.979167	6.131603
P6	SIR	-6.420268	-5.599114	-7.869622
	SDR	-10.462611	-12.715794	-10.036907
	SAR	3.097133	3.928	4.133405
M1	SIR	-5.654331	-4.653175	-7.277335
	SDR	-10.455755	-13.740777	-10.866475
	SAR	2.825318	0.932182	1.735088
M2	SIR	-4.784475	-7.907911	-6.951342
	SDR	-10.272537	-12.539653	-9.796059
	SAR	4.93121	4.24077	5.584858
M3	SIR	-6.340186	-5.924676	-6.66068
	SDR	-11.455352	-13.413126	-10.875728
	SAR	0.696317	0.08456	1.021655
M4	SIR	-7.106484	-6.096678	-7.51231
	SDR	-10.541863	-13.166554	-10.439193
	SAR	4.072988	1.889415	3.040509
M5	SIR	-3.433632	-5.857594	-5.177591
	SDR	-12.145362	-15.432483	-11.003223
	SAR	-1.202955	-4.603023	0.300179
M6	SIR	-3.79337	-3.934154	-5.637984
	SDR	-11.232939	-13.10206	-10.509362
	SAR	-10.743046	-9.075993	-6.64232
M7	SIR	-3.336596	-4.663076	-5.046564
	SDR	-10.905078	-13.069767	-10.413024
	SAR	-4.782624	-6.210111	-4.108072

Tabla 4.16: Evaluaciones de canciones hechas a base MIDI, correspondientes a los modelos con arquitectura DCUNet parte 1.

Modelo	Métrica	TeDareLoMejor	TheHeartOfLife	TuEstasAqui
M8	SIR	-3.944584	-4.507029	-5.327853
	SDR	-11.230708	-14.100759	-10.831251
	SAR	-0.725942	-1.179515	-0.775537
M9	SIR	-4.878591	-6.117088	-5.553776
	SDR	-11.067608	-14.830317	-10.8808
	SAR	-2.930337	-1.804534	-0.531657
M10	SIR	-5.538671	-5.498751	-5.731617
	SDR	-11.148759	-14.431783	-11.008363
	SAR	0.574889	-1.339528	0.070045
M11	SIR	-3.598391	-4.798989	-4.542171
	SDR	-11.242714	-14.883426	-11.049905
	SAR	-3.576363	-2.579469	-1.982644

Tabla 4.17: Evaluaciones de canciones hechas a base MIDI, correspondientes a los modelos con arquitectura DCUNet parte 2.

Modelo	Métrica	Bearlin Roads	LizNelson Rainfall	QuePena TantoFaz 6-19	QuePena TantoFaz 46-57	ThIsAmazin Grace
P1	SIR	-11.067176	5.188511	-2.388611	-0.568529	-10.058973
	SDR	-16.320769	-8.842232	-8.257646	-8.42947	-16.022898
	SAR	-12.359961	-12.493646	-12.374306	-12.440825	-12.754807
P2	SIR	-10.393206	5.686962	-1.665089	-0.484874	-9.678192
	SDR	-16.030201	-7.437195	-7.298345	-7.473405	-15.644205
	SAR	-15.443631	-15.046505	-15.365359	-15.853549	-16.008594
P3	SIR	-4.726384	8.232253	2.846828	4.181332	-4.995811
	SDR	-13.856807	-2.098843	-4.587243	-4.058745	-11.154917
	SAR	-8.972423	-5.137049	-4.133638	-5.907688	-8.992009
P4	SIR	-7.756066	2.307279	-1.145531	-0.362672	-6.122455
	SDR	-13.372877	1.025791	-2.317354	-1.231929	-10.429263
	SAR	3.54959	8.816794	10.617893	10.72744	4.010763
P5	SIR	-7.996498	5.567693	3.140387	3.08265	-6.477063
	SDR	-13.359363	2.003345	-1.580902	-1.070548	-10.532913
	SAR	4.251819	6.534006	5.761862	4.406225	3.68647
P6	SIR	-6.075845	5.716227	5.121219	4.799993	-6.384785
	SDR	-13.193406	1.06445	-2.183972	-1.892964	-10.505194
	SAR	4.341473	5.63358	5.242026	4.78893	3.795389
M1	SIR	-5.674847	6.923947	4.418018	4.788726	-5.776522
	SDR	-13.445428	-0.575497	-3.723636	-2.705346	-10.525595
	SAR	2.889915	2.438138	2.769589	3.097554	3.22408

Tabla 4.18: Evaluaciones de canciones reales, correspondientes a los modelos con arquitectura DCUNet parte 1.

Modelo	Métrica	Bearlin Roads	LizNelson Rainfall	QuePena TantoFaz 6-19	QuePena TantoFaz 46-57	ThIsAmazin Grace
M2	SIR	-6.081969	5.633766	3.563709	3.479197	-5.349082
	SDR	-13.294135	1.482858	-1.186345	-1.205417	-10.466385
	SAR	4.075641	7.462083	8.274313	6.145296	4.38957
M3	SIR	-5.561192	7.708779	4.976957	5.425309	-4.291647
	SDR	-14.051443	-0.568142	-3.05514	-3.022464	-11.22127
	SAR	-1.575557	2.436694	3.867823	1.83289	-0.934493
M4	SIR	-5.979728	9.436075	8.469343	4.558117	-5.339074
	SDR	-13.449412	0.187022	-2.450625	-2.35466	-10.491276
	SAR	3.277643	3.808909	5.615977	4.691117	3.498143
M5	SIR	-4.911886	11.790681	10.177098	7.988987	-4.245444
	SDR	-14.741087	-2.69553	-5.707365	-4.619157	-11.736511
	SAR	-1.19125	-2.560688	-2.491665	-0.689519	-1.116316
M6	SIR	-3.351065	9.817867	8.681702	6.741764	-5.841337
	SDR	-13.929547	-2.761119	-4.364779	-4.373808	-11.448631
	SAR	-12.00218	-7.913498	-2.809208	-5.977563	-10.789198
M7	SIR	-3.945229	9.748016	9.71441	6.83406	-4.918769
	SDR	-13.769698	-2.598748	-3.905533	-3.683871	-11.308868
	SAR	-7.49486	-7.741341	-2.11571	-4.897398	-7.622377
M8	SIR	-4.467632	12.188197	8.680438	8.999584	-3.985713
	SDR	-13.919451	-1.120658	-2.956078	-3.438181	-11.316136
	SAR	0.022977	1.116443	4.011135	0.43105	-0.706468
M9	SIR	-5.134651	7.581396	11.760391	8.227068	-4.613318
	SDR	-14.73607	-1.441387	-3.768724	-4.312078	-11.64594
	SAR	-1.221666	0.986078	3.049233	1.388756	-0.543961
M10	SIR	-4.81183	9.885031	5.853522	6.446206	-5.3495
	SDR	-13.786166	-1.192025	-4.349083	-4.391027	-10.769227
	SAR	0.752892	1.716332	2.105905	0.038066	1.316278
M11	SIR	-4.416711	12.096969	9.23386	8.545862	-5.264553
	SDR	-14.266132	-0.831837	-4.036973	-4.696186	-11.402172
	SAR	-0.74781	2.813661	2.775936	0.482117	-0.656205

Tabla 4.19: Evaluaciones de canciones reales, correspondientes a los modelos con arquitectura DCUNet parte 2.

Modelo	Métrica	TeDareLoMejor	TheHeartOfLife	TuEstasAqui
P1	SIR	-8.087234	-8.56002	-9.232925
	SDR	-12.468013	-13.382094	-11.886539
	SAR	1.517114	4.223597	2.299755
P2	SIR	-6.116078	-5.537651	-7.558648
	SDR	-11.427769	-13.948739	-11.174851
	SAR	1.77091	2.059999	1.557564
P3	SIR	-6.105366	-5.179572	-7.48397
	SDR	-12.066048	-14.861962	-11.925167
	SAR	0.689234	1.036096	0.116988
M1	SIR	-5.973127	-4.721725	-7.592045
	SDR	-11.623693	-14.113636	-11.595028
	SAR	1.404626	1.515559	0.908543
M2	SIR	-5.815569	-8.033926	-7.552881
	SDR	-10.753865	-13.559873	-11.237365
	SAR	3.140672	3.049435	1.672743
M3	SIR	-6.024399	-7.242135	-7.62832
	SDR	-10.702875	-14.008023	-10.972098
	SAR	4.331601	2.25634	2.792727
M4	SIR	-6.804524	-7.411715	-8.007196
	SDR	-11.995218	-14.394277	-11.394218
	SAR	1.293741	1.512189	2.164934
M5	SIR	-6.14383	-5.262265	-7.620574
	SDR	-10.668662	-14.616969	-11.431844
	SAR	3.971549	1.687873	1.603969
M6	SIR	-5.704578	-7.148578	-7.512145
	SDR	-10.736618	-13.802665	-11.503581
	SAR	3.185022	3.211333	1.156066
M7	SIR	-6.228856	-7.621752	-7.825824
	SDR	-12.500435	-14.161546	-12.217062
	SAR	-0.036742	2.035781	0.026281
M8	SIR	-6.371624	-7.897575	-7.672387
	SDR	-11.866019	-14.70277	-12.165978
	SAR	0.971393	1.160755	0.25846
M9	SIR	-5.78112	-6.539302	-7.45562
	SDR	-11.014407	-14.298558	-11.530838
	SAR	2.551144	1.812401	1.261434
M10	SIR	-5.670785	-7.069439	-8.119087
	SDR	-11.303068	-14.237531	-11.359981
	SAR	1.713375	1.82113	1.690886

Tabla 4.20: Evaluaciones de canciones hechas a base MIDI, correspondientes a los modelos con arquitectura SudoRMRF parte 1.

Modelo	Métrica	TeDareLoMejor	TheHeartOfLife	TuEstasAqui
M11	SIR	-6.452744	-8.362914	-7.943318
	SDR	-11.058918	-13.596437	-10.811801
	SAR	3.306649	3.284757	4.032529
M12	SIR	-7.141949	-6.847762	-8.514219
	SDR	-11.326748	-13.472503	-10.930141
	SAR	2.731219	3.835592	3.739293
M13	SIR	-6.084827	-7.956558	-7.669855
	SDR	-10.666636	-14.132841	-10.68946
	SAR	4.112069	2.072266	3.418687
M14	SIR	-9.238071	-12.171863	-9.430586
	SDR	-11.093118	-12.430286	-10.253973
	SAR	12.422203	13.775175	12.12678

Tabla 4.21: Evaluaciones de canciones hechas a base MIDI, correspondientes a los modelos con arquitectura SudoRMRF parte 2.

Modelo	Métrica	Bearlin Roads	LizNelson Rainfall	QuePena TantoFaz 6-19	QuePena TantoFaz 46-57	ThIsAmazin Grace
P1	SIR	-8.597494	5.374761	1.285227	1.186915	-7.102994
	SDR	-14.892808	-0.222406	-3.769853	-2.886319	-12.234309
	SAR	2.887553	4.735239	5.500412	5.12575	2.530205
P2	SIR	-6.146939	5.402163	2.067416	2.260588	-6.24845
	SDR	-14.07213	-0.385232	-2.854283	-2.047788	-11.888587
	SAR	1.80058	4.04633	6.222361	6.171706	1.008563
P3	SIR	-6.399445	8.140343	3.110027	2.879751	-6.157284
	SDR	-14.361968	-0.838754	-4.639865	-3.591463	-12.108113
	SAR	1.550115	3.288693	3.247883	3.317946	0.87123
M1	SIR	-6.589117	6.600682	2.197804	3.008272	-6.314369
	SDR	-14.203001	-0.954642	-3.992456	-2.993597	-11.987507
	SAR	1.735524	2.897254	4.180284	4.272529	0.784006
M2	SIR	-6.506935	5.256676	2.416591	2.655448	-5.271112
	SDR	-13.953826	-1.741144	-2.523088	-2.581037	-11.463363
	SAR	2.491224	2.384617	7.981531	5.663785	1.595519
M3	SIR	-6.737463	3.556867	1.537566	2.643234	-5.748808
	SDR	-13.967239	-1.017427	-3.366255	-2.953334	-11.255169
	SAR	3.017922	4.123899	6.351181	4.971837	2.648935
M4	SIR	-7.465792	5.195641	1.393655	2.178996	-5.465935
	SDR	-14.844419	-0.60643	-3.999448	-3.549258	-12.014364
	SAR	1.428888	4.072654	4.821958	3.621305	1.192858

Tabla 4.22: Evaluaciones de canciones reales, correspondientes a los modelos con arquitectura SudoRMRF parte 1.

Modelo	Métrica	Bearlin Roads	LizNelson Rainfall	QuePena TantoFaz 6-19	QuePena TantoFaz 46-57	ThIsAmazin Grace
M5	SIR	-6.62814	6.979254	2.482915	4.270401	-6.242131
	SDR	-14.007645	-0.250561	-3.503086	-3.206107	-11.891965
	SAR	3.124717	4.445541	5.828248	4.195471	1.369848
M6	SIR	-6.387662	5.452582	0.910176	3.429212	-6.338379
	SDR	-13.782317	-0.785977	-3.769813	-3.283878	-11.914314
	SAR	2.710783	3.85617	5.370306	4.168921	1.128659
M7	SIR	-7.557151	5.102861	1.427362	2.062891	-5.039386
	SDR	-15.434603	-1.922738	-3.850439	-2.930074	-13.031859
	SAR	0.392025	2.037713	4.90638	4.319952	-0.720598
M8	SIR	-7.394414	4.081482	0.844805	1.778111	-5.842655
	SDR	-15.093656	-1.111256	-4.231722	-4.159379	-12.73529
	SAR	0.793799	3.390721	4.357832	2.957738	-0.1123
M9	SIR	-6.563756	5.907157	2.511732	3.287691	-5.241348
	SDR	-14.503941	-1.219539	-2.669043	-2.893671	-11.485015
	SAR	1.412636	3.277698	7.905446	5.048023	1.645969
M10	SIR	-6.21207	7.37453	1.917232	2.707246	-6.56409
	SDR	-14.18744	-1.153734	-3.053254	-2.634302	-12.343247
	SAR	2.002629	2.984544	6.672272	5.16279	0.394544
M11	SIR	-6.963096	3.638776	1.462398	2.387426	-5.2298163
	SDR	-13.827462	-0.683881	-3.143217	-2.601292	-11.051935
	SAR	3.361517	5.744876	8.277058	6.214484	2.716287
M12	SIR	-7.387412	5.826038	2.09195	2.480187	-5.77719
	SDR	-13.92275	0.653545	-2.718597	-2.030345	-11.215809
	SAR	4.108515	6.791504	8.697877	7.292376	3.188107
M13	SIR	-6.885188	3.276584	1.589023	2.636578	-5.118119
	SDR	-13.766349	-1.050629	-2.958305	-2.738332	-11.157515
	SAR	3.661143	4.275076	7.612346	5.516852	2.170281
M14	SIR	-12.05725	0.310484	-4.144551	-1.75935	-9.607199
	SDR	-13.657255	0.559614	-3.697609	-2.229905	-10.833047
	SAR	13.131397	13.268408	14.472438	12.493132	11.775693

Tabla 4.23: Evaluaciones de canciones reales, correspondientes a los modelos con arquitectura SudoRMRF parte 2.

Modelo	Métrica	TeDareLoMejor	TheHeartOfLife	TuEstasAqui
P1	SIR	-7.098673	-7.903718	-8.85577
	SDR	-12.188929	-12.958767	-11.524796
	SAR	1.858169	4.925547	2.53752
P2	SIR	-8.107196	-8.408468	-9.153506
	SDR	-11.807705	-12.566494	-11.10464
	SAR	3.252784	7.298639	4.447323
P3	SIR	-10.275005	-11.262599	-10.281296
	SDR	-10.787223	-12.034516	-10.101973
	SAR	16.954428	20.485384	17.551675
M1	SIR	-10.413702	-11.195004	-10.391341
	SDR	-10.782823	-12.020122	-10.101768
	SAR	18.009156	20.743713	18.731403
M2	SIR	-10.025254	-11.381889	-10.111916
	SDR	-10.800078	-12.043429	-10.066364
	SAR	17.278924	19.469997	17.781225
M3	SIR	-10.219946	-11.316855	-10.239647
	SDR	-10.787413	-12.042126	-10.099252
	SAR	16.525548	19.965746	17.006383
M4	SIR	-10.185512	-11.412993	-10.19897
	SDR	-10.787696	-12.049261	-10.097127
	SAR	16.409845	19.756688	16.911085
M5	SIR	-9.724858	-11.854045	-9.936559
	SDR	-10.80315	-12.091651	-10.019359
	SAR	13.507934	16.38145	13.704364
M6	SIR	-9.737766	-11.954212	-9.946277
	SDR	-10.816728	-12.106268	-10.016622
	SAR	15.020975	17.438708	14.942373
M7	SIR	-9.842115	-11.838735	-10.00007
	SDR	-10.803642	-12.099221	-10.055471
	SAR	15.850186	18.383821	15.998908
M8	SIR	-10.173476	-11.371634	-10.190083
	SDR	-10.79083	-12.04943	-10.109331
	SAR	16.554237	19.884619	17.081329
M9	SIR	-10.107138	-11.423982	-10.145901
	SDR	-10.790148	-12.051118	-10.086271
	SAR	15.788982	19.163116	16.320184
M10	SIR	-10.188988	-11.433281	-10.202627
	SDR	-10.789206	-12.052641	-10.099749
	SAR	16.822923	19.830745	17.199553

Tabla 4.24: Evaluaciones de canciones hechas a base MIDI, correspondientes a los modelos con arquitectura DPTNet parte 1.

Modelo	Métrica	TeDareLoMejor	TheHeartOfLife	TuEstasAqui
M11	SIR	-9.742256	-11.937073	-9.888035
	SDR	-10.808775	-12.119958	-10.063904
	SAR	15.424368	18.187241	15.535984
M12	SIR	-9.733284	-11.916139	-9.912531
	SDR	-10.806193	-12.110328	-10.046942
	SAR	15.726215	18.25024	15.819364
M13	SIR	-9.83376	-11.891137	-9.990545
	SDR	-10.812043	-12.108265	-10.071252
	SAR	15.061965	18.180888	15.315634
M14	SIR	-9.842494	-11.902464	-9.995478
	SDR	-10.807046	-12.107506	-10.061302
	SAR	15.196163	18.210685	15.443517
M15	SIR	-6.952148	-6.662083	-8.484857
	SDR	-11.541677	-14.123911	-11.377232
	SAR	2.932568	2.25604	2.824906
M16	SIR	-10.129746	-11.405466	-10.180529
	SDR	-10.790322	-12.051255	-10.095215
	SAR	15.817196	19.459633	16.414436
M17	SIR	-7.48018	-8.506981	-8.96183
	SDR	-11.871046	-13.19497	-11.266606
	SAR	2.659611	5.802099	4.354613
M18	SIR	-10.820914	-11.032634	-10.854053
	SDR	-11.223742	-12.310352	-10.50804
	SAR	10.921337	10.12675	10.398242
M19	SIR	-10.570153	-11.048071	-10.648201
	SDR	-11.11228	-12.246975	-10.398986
	SAR	13.905151	12.801371	13.098036
M20	SIR	-9.792302	-11.103232	-10.354556
	SDR	-11.49316	-12.611699	-10.691499
	SAR	9.989517	12.997648	10.641756
M21	SIR	-9.122764	-9.756643	-9.47248
	SDR	-10.582491	-12.302607	-10.253671
	SAR	7.793382	10.352024	7.408976
M22	SIR	-7.04771	-9.439139	-8.603191
	SDR	-11.886313	-12.562418	-11.099278
	SAR	2.863142	9.403422	4.625092

Tabla 4.25: Evaluaciones de canciones hechas a base MIDI, correspondientes a los modelos con arquitectura DPTNet parte 2.

Modelo	Métrica	Bearlin Roads	LizNelson Rainfall	QuePena TantoFaz 6-19	QuePena TantoFaz 46-57	ThIsAmazin Grace
P1	SIR	-8.521691	5.465684	1.632289	2.203527	-6.518926
	SDR	-14.365624	0.466996	-3.380092	-1.550748	-11.902638
	SAR	3.47924	6.471249	7.222073	8.109541	2.26963
P2	SIR	-8.311787	5.05167	0.180269	1.635034	-7.298367
	SDR	-14.549778	1.94877	-2.908057	-1.259605	-11.755851
	SAR	3.58542	9.702024	8.400845	8.604897	3.744621
P3	SIR	-12.309238	3.078052	-2.59726	-0.990687	-10.729377
	SDR	-13.420552	3.021552	-2.589083	-1.156678	-10.730209
	SAR	21.262634	22.790159	23.488041	20.490668	18.118856
M1	SIR	-12.360174	3.314533	-2.406279	-0.871134	-10.77724
	SDR	-13.413185	3.207551	-2.498657	-1.079519	-10.727481
	SAR	22.486851	23.839204	24.037039	21.584007	19.551746
M2	SIR	-12.3425	2.845494	-2.796477	-1.079836	-10.621394
	SDR	-13.433593	2.82586	-2.683893	-1.236724	-10.72657
	SAR	20.091155	21.300221	22.132448	19.229764	17.576414
M3	SIR	-12.274475	2.911354	-2.715399	-1.069101	-10.673097
	SDR	-13.428132	2.884707	-2.64147	-1.203844	-10.72978
	SAR	20.419706	21.70875	22.652244	19.752304	17.425742
M4	SIR	-12.189648	2.608983	-2.938329	-1.204592	-10.533454
	SDR	-13.426646	2.661473	-2.734003	-1.276078	-10.716398
	SAR	20.094499	21.428871	22.636326	19.49739	17.061385
M5	SIR	-12.21109	1.672253	-3.687385	-1.38052	-10.046938
	SDR	-13.446845	1.90449	-3.083223	-1.493306	-10.663037
	SAR	16.543062	17.463067	18.285552	15.003878	13.390156
M6	SIR	-12.11689	1.413536	-3.735404	-1.405371	-9.975711
	SDR	-13.450777	1.753642	-3.100482	-1.490024	-10.669205
	SAR	17.270065	18.583392	19.988668	16.676334	14.502575
M7	SIR	-12.029579	1.543085	-3.659869	-1.472062	-10.103689
	SDR	-13.446008	1.866093	-3.050058	-1.470974	-10.684745
	SAR	17.950801	19.320882	20.781848	17.878372	15.363844
M8	SIR	-12.213291	2.752224	-2.832307	-1.139345	-10.575991
	SDR	-13.433923	2.762292	-2.688203	-1.241411	-10.724155
	SAR	20.101932	21.219995	22.633898	19.623678	17.126267
M9	SIR	-12.215303	2.60821	-2.93054	-1.19803	-10.556516
	SDR	-13.427568	2.650168	-2.726612	-1.280387	-10.722208
	SAR	19.470745	20.773995	21.713718	18.863091	16.473887
M10	SIR	-12.176151	2.561335	-2.971809	-1.226182	-10.48217
	SDR	-13.430061	2.625852	-2.749311	-1.285937	-10.712444
	SAR	20.296837	21.382298	22.8395	19.696959	17.141072

Tabla 4.26: Evaluaciones de canciones reales, correspondientes a los modelos con arquitectura DPTNet parte 1.

Modelo	Métrica	Bearlin Roads	LizNelson Rainfall	QuePena TantoFaz 6-19	QuePena TantoFaz 46-57	ThIsAmazin Grace
M11	SIR	-11.924193	1.313557	-3.775313	-1.527952	-10.003328
	SDR	-13.455654	1.702255	-3.098631	-1.497746	-10.681321
	SAR	17.379218	19.016947	20.893554	18.049508	15.039365
M12	SIR	-11.952623	1.316316	-3.766974	-1.511587	-10.025003
	SDR	-13.45035	1.709059	-3.097896	-1.498005	-10.679048
	SAR	17.689746	19.348502	21.248951	18.205307	15.270448
M13	SIR	-11.943112	1.371205	-3.733013	-1.491293	-10.040323
	SDR	-13.453654	1.740504	-3.082522	-1.483754	-10.685392
	SAR	17.331126	18.982139	21.128063	17.927932	14.883559
M14	SIR	-11.932362	1.319245	-3.767556	-1.516817	-10.023682
	SDR	-13.443876	1.710765	-3.099184	-1.499802	-10.678531
	SAR	17.622248	19.370704	21.378125	18.040039	15.079346
M15	SIR	-8.029735	5.941341	2.953608	3.33823	-6.101946
	SDR	-14.554694	0.633453	-3.440906	-2.523994	-11.88428
	SAR	2.888288	6.668661	6.209428	5.775011	2.050187
M16	SIR	-12.215339	2.66097	-2.906401	-1.169032	-10.570543
	SDR	-13.430217	2.690214	-2.723728	-1.266082	-10.722239
	SAR	19.630117	20.992876	22.037269	19.090045	16.750553
M17	SIR	-8.849998	5.333613	1.700792	1.426069	-6.754886
	SDR	-15.045961	1.438663	-2.824075	-1.967939	-11.938186
	SAR	3.104554	8.916741	9.858241	8.804925	3.617635
M18	SIR	-12.606915	4.450833	-1.698027	0.020807	-10.894139
	SDR	-13.742511	3.014959	-2.753672	-1.394337	-11.094259
	SAR	11.879383	11.537664	10.644586	9.777538	10.434143
M19	SIR	-12.617879	4.419715	-1.710425	0.010954	-10.889651
	SDR	-13.665512	3.46912	-2.573322	-1.173317	-11.019386
	SAR	15.272339	15.214649	14.063412	12.602409	13.124865
M20	SIR	-11.931597	3.429375	-1.988901	0.021197	-10.024822
	SDR	-13.958491	2.308427	-3.144322	-1.643444	-11.276816
	SAR	13.762827	13.998105	14.152505	12.769839	12.217094
M21	SIR	-9.954688	4.047172	-0.402977	0.581305	-7.701751
	SDR	-13.368854	2.175117	-2.426559	-1.186268	-10.756663
	SAR	8.485543	11.270017	13.441629	12.213408	7.324764
M22	SIR	-7.968039	4.36035	0.43065	1.029734	-5.882606
	SDR	-14.569174	1.484304	-2.503394	-1.400945	-11.594448
	SAR	4.330603	10.400647	12.713288	11.529222	4.138452

Tabla 4.27: Evaluaciones de canciones reales, correspondientes a los modelos con arquitectura DPTNet parte 2.

A continuación se muestran los valores de SIR de manera gráfica. Cada grupo de barras corresponden a una canción, y cada barra en el grupo corresponde a un modelo. Las barras que corresponden a un mismo modelo estan del mismo color. Se muestran solo los diez mejores modelos por cada arquitectura, con excepción de Open-Unmix de 5 y 7 capas, los cuales se entrenaron solo 5 modelos de cada una.

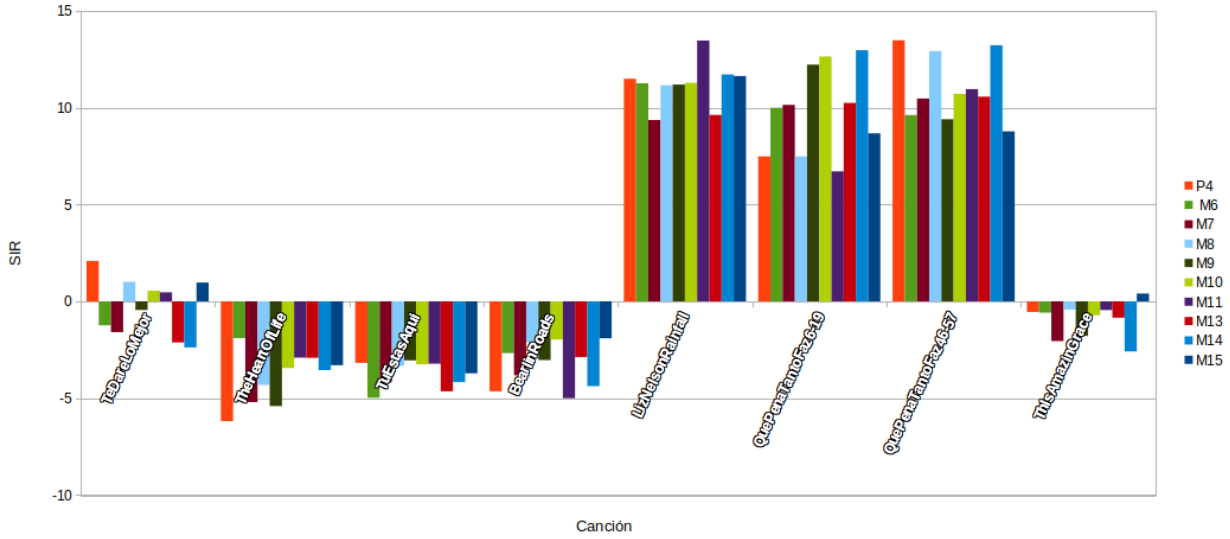

Figura 4.1: SIR de los 10 mejores modelos de la arquitectura Open-Unmix de 3 capas.

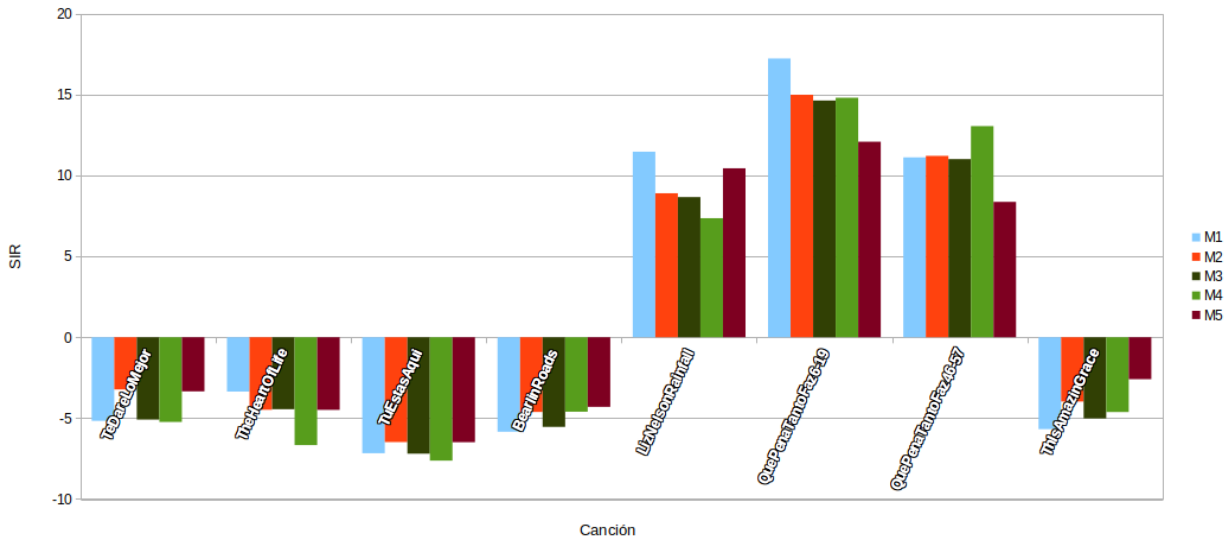

Figura 4.2: SIR de los modelos de la arquitectura Open-Unmix de 5 capas.

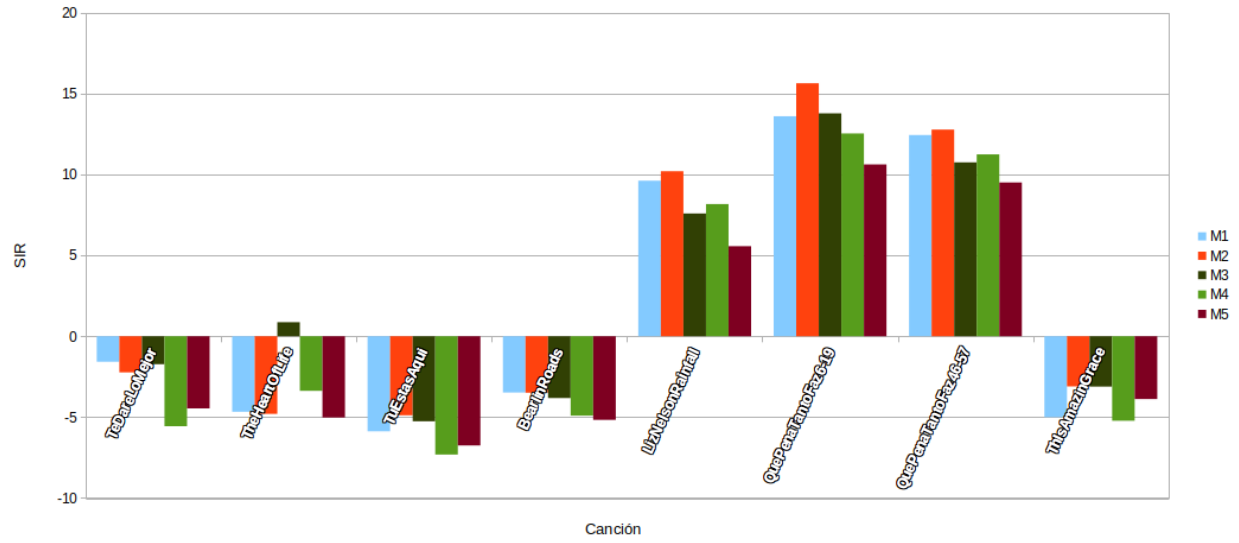

Figura 4.3: SIR de los modelos de la arquitectura Open-Unmix de 7 capas.

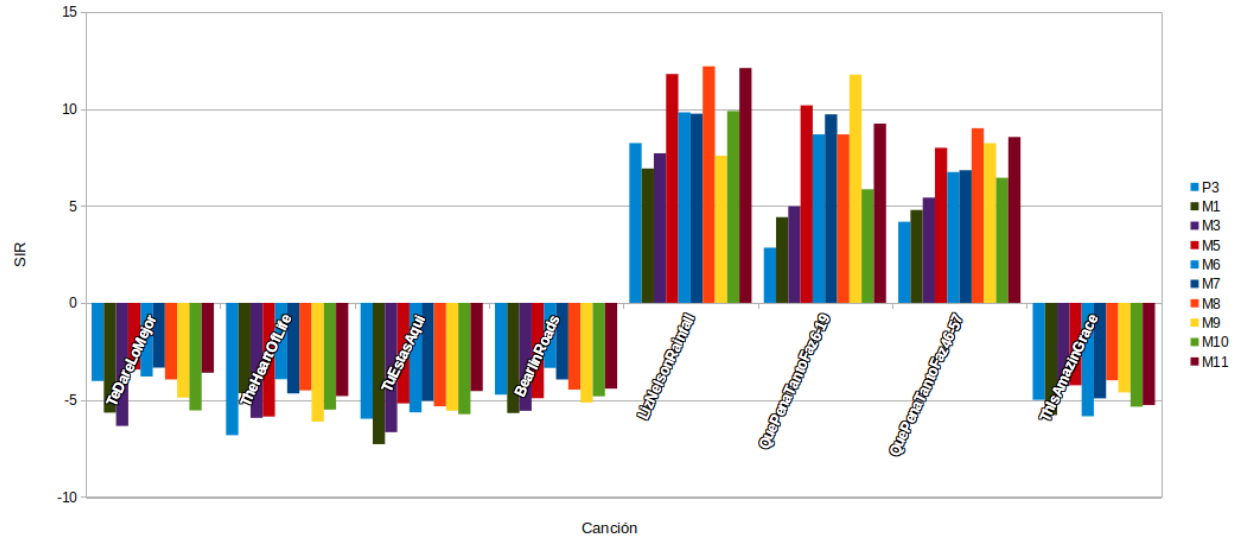

Figura 4.4: SIR de los 10 mejores modelos de la arquitectura DCUNet.

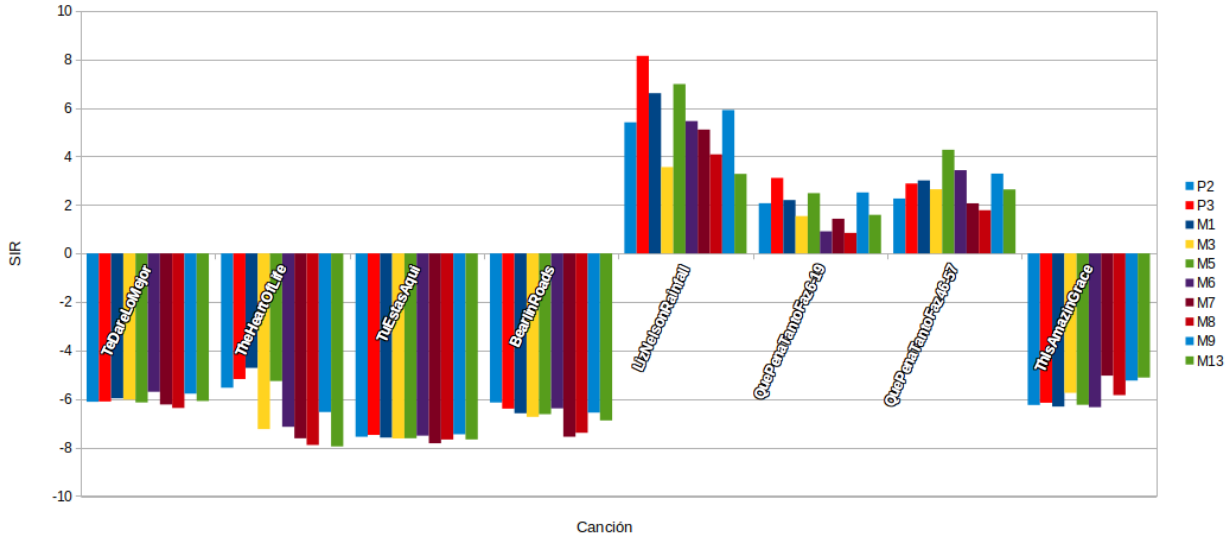

Figura 4.5: SIR de los 10 mejores modelos de la arquitectura SudoRMRF.

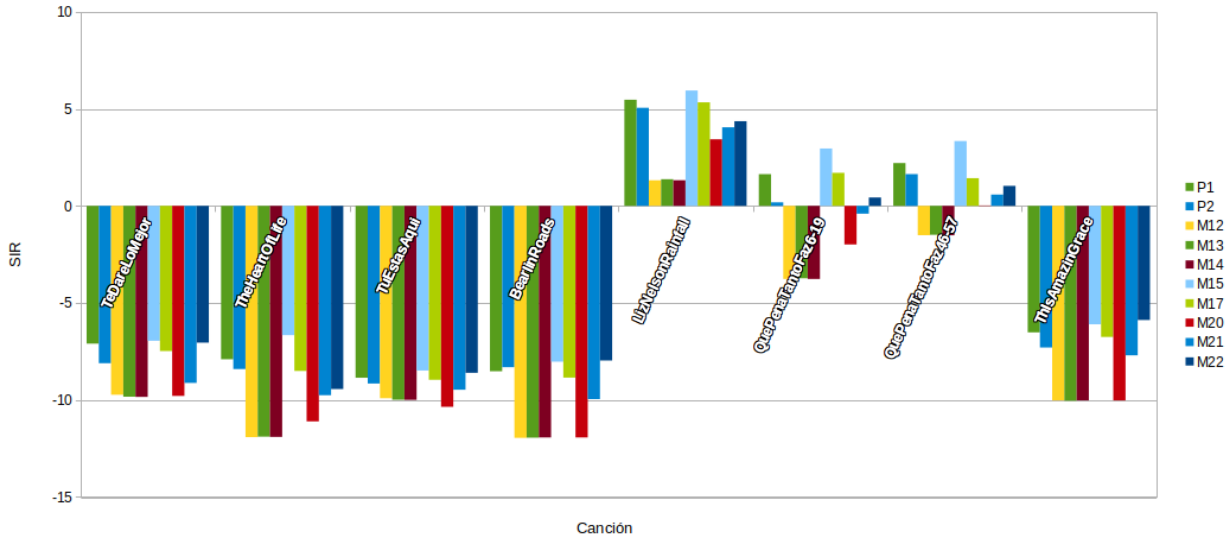

Figura 4.6: SIR de los 10 mejores modelos de la arquitectura DPTNet.

4.4. Discusión

En el capítulo 2 se menciona que a lo largo de los años se han observado varios errores de implementación para calcular la métricas SDR y SAR; su implementación no está bien definida. Estas mediciones, por lo tanto, no son muy aceptadas en la comunidad. Esto no es el caso de la métrica SIR, por lo que se tomó como métrica principal.

En promedio, la arquitectura que obtuvo mejores resultados de SIR fue la de Open-Unmix y entre las variaciones de sus capas BLSTM's, el que obtuvo mejores resultados fueron los modelos de 3 capas. La arquitectura que en promedio obtuvo peor desempeño (de acuerdo a los resultados del SIR) fué la de DPTNet.

Una intuición de por qué Open-Unmix obtuvo mejores resultados para separar la guitarra acústica es que de las 4 arquitecturas escogidas, Open-Unmix es la única que se diseñó específicamente para separación de fuentes musicales. Pero solo separa las pistas de batería, bajo, voces y otros. En cambio las otras 3 arquitecturas trabajan con la voz y uno de ellos separa sonidos ambientales también. Algo interesante de notar es que a pesar de que DCUNet es una arquitectura de mejora de la voz, no fue la que peor desempeño obtuvo.

En las tablas se observa que para cada canción los resultados varían, esto se debe a algunos factores como que si la canción es hecha con MIDI o con instrumentos reales y la cantidad de instrumentos que hay en una canción. Este último factor influye mucho en la

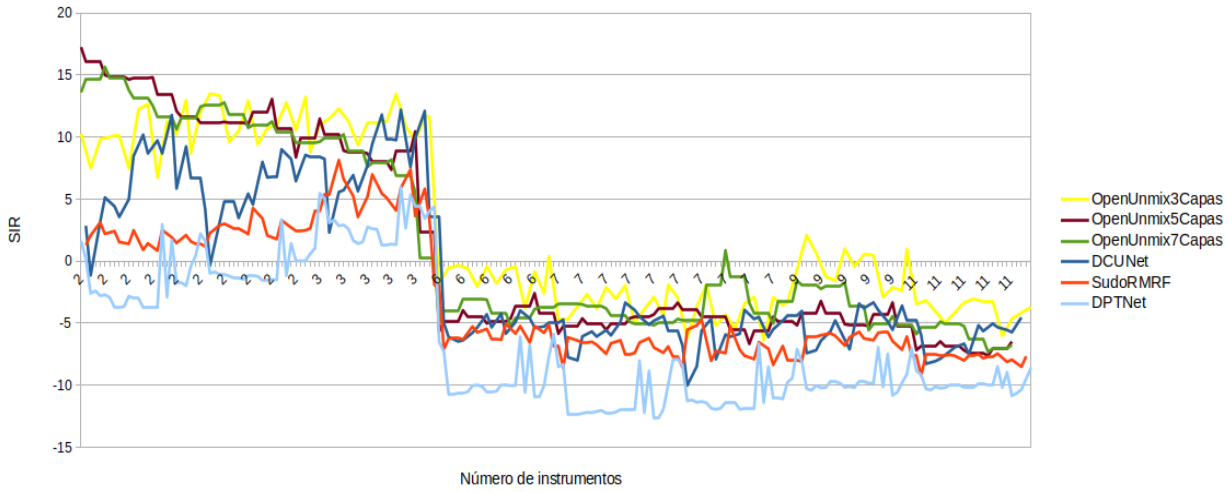

Figura 4.7: Gráficas de SIR vs número de instrumentos, para todas las arquitecturas entrenadas.

calidad de la separación. Esto se puede observar en la figura 4.7 la cual muestra las gráficas de la tendencia del SIR, en función de la cantidad de instrumentos en una canción, esto para cada una de las arquitecturas entrenadas. Las gráficas muestran que hay una gran caída del SIR a partir de 6 instrumentos.

Al estar entrenando los modelos de SudoRMRF se encontró con un *bug* en el repositorio de Asteroid. El error no permitía entrenar modelos grandes; movía algunos tensores al CPU y no todo al GPU. Este *bug* lo reporté en la sección de discusión del repositorio. Debido a esto, los modelos de SudoRMRF entrenados fueron aquellos cuyas características se ajustaban para no llegar a rebasar la memoria, de tal forma que los tensores no llegasen a moverse al CPU. Esto fue una limitante en el entrenamiento para este tipo de arquitectura.

A decir verdad, los servidores utilizados en el estado del arte rebasan las características de memoria y potencia, comparado con el servidor utilizado en este proyecto. Para poder entrenar una red muy grande es necesaria mucha memoria de GPU, y eso es algo que en el estado del arte no se tiene como limitante. Es por ello que, para la mayoría de los modelos entrenados en este proyecto fueron ajustados las características y la cantidad de tamaño de lote y segundos por cada canción. De esta manera, se pudieron entrenar con la memoria disponible en las GPU's del servidor.

En contexto con el estado del arte; a continuación se muestran los resultados de SIR que se presentan en Demucs [3], Spleeter [5] y *Multi Channel U-Net* [32]. En [3] y [5] los resultados corresponden a las separaciones de voces, bajo, batería y otros. Mientras que en [32] los resultados de SIR mostrados corresponden a 3 de sus modelos los cuáles tienen diferentes variantes de ponderación basada en energía (EBW).

- Demucs: voz: 13.31, bajo: 10.55, batería: 11.81, otros: 5.9.
- Spleeter: voz: 15.19, bajo: 10.01, batería: 12.24, otros: 7.86.
- *Multi Channel U-Net*: 7.97, 7.98, 8.33.

En [9] se realiza separación de la guitarra eléctrica y presentan un SIR global de: 10.35 dB.

Si observamos las tablas con los resultados, notamos que (en promedio) para las canciones con 2 o 3 instrumentos los modelos de Open-Unmix alcanzan o llegan a rebasar los valores

del estado del arte, mientras que los demás modelos no logran alcanzar los valores del estado del arte. Para canciones con 6 o más instrumentos ningún modelo entrenado logra alcanzar los valores del estado del arte.

Capítulo 5

Conclusiones

La separación de instrumentos en grabaciones musicales se ha enfocado en la voz, el bajo, la batería, y el resto de los instrumentos se agrupan en la cuarta categoría "otros". Dado la popularidad de la guitarra en una gran variedad de géneros musicales, se tuvo como objetivo separar la guitarra de grabaciones musicales. Dado los pocos recursos de entrenamiento para este propósito, se creó un corpus a partir de grabaciones que originalmente están en formato MIDI pero que fueron transformadas a formatos que convencionalmente se utilizarían para separación de instrumentos en grabaciones musicales. Con esto, se lograron obtener algunos modelos de aprendizaje profundo que separan la guitarra acústica de la mezcla de una canción.

La guitarra acústica es un instrumento muy popular en varios géneros musicales, sin embargo al ver que en el estado del arte solo dos aplicaciones realizan la separación de este instrumento surge la pregunta: ¿Porque la guitarra acústica no es separada tanto como ocurre con el bajo, la batería y la voz? Podría ser porque existen diferentes tipos de guitarras acústicas con sus respectivos timbres y tipos de cuerda o simplemente porque el bajo, la batería y la voz son instrumentos elementales en cualquier canción, más elementales que cualquier otro instrumento. En la base de datos creada se procuró que se tengan canciones de ambos tipos de cuerdas de guitarra: de nylon y de metal; así para que no se generalice a un tipo de cuerda nada más.

Se probaron 4 arquitecturas con variantes en sus características, lo que al final generó 84 modelos entrenados en total. Se utilizaron los repositorios Open-Unmix [23] y Asteroid

[61] como base, y se encontró que la arquitectura que obtuvo mejor desempeño fue la de Open-Unmix.

Es importante considerar que para entrenar modelos de separación de fuentes musicales requiere de mucha memoria computacional. También se encontró que la cantidad de instrumentos que hay en una canción afecta notablemente el desempeño de las separaciones de fuentes musicales.

Es interesante notar que en el caso de los modelos de Open-Unmix de diferentes capas BLSTM se obtuvo mejor desempeño en los de 3 capas. Pero también hay que mencionar que para los modelos de 3 capas se entrenaron 19 modelos, mientras que para los modelos de 5 capas y 7 capas solo se entrenaron cinco modelos de cada uno.

Dado el gran tamaño de los datos con los cuales se debe entrenar una red para separación de fuentes musicales y el tamaño de las arquitecturas propuestas, se requiere de mucha memoria computacional para entrenar modelos grandes de separación de fuentes musicales. Mientras se entrene con más canciones y más segundos por canción dicha memoria deberá ser mayor para los entrenamientos.

5.1. Trabajo a futuro

Como objetivo colateral del proyecto, los desarrolladores de Asteroid actualizaron el repositorio, reparando un *bug* (el cual había reportado) que aparecía al entrenar modelos grandes de SudoRMRF. Otra mejora que los desarrolladores realizaron es que ya se puede entrenar con una versión más actualizada de *Pytorch Lightning*. Un trabajo a futuro sería utilizar la versión más actualizada de Asteroid para los entrenamientos.

Otro trabajo a futuro sería utilizar más de 8 canciones para la evaluación, e incluir canciones con 4, 5 y 8 instrumentos.

En este proyecto solo se probaron 4 arquitecturas, pero existen decenas de arquitecturas para separación de fuentes: algunas mas recientes y complejas que otras. Un trabajo a futuro sería probar más arquitecturas para crear modelos de separación de la guitarra acústica.

Bibliografía

[1] T. Li, J. Chen, H. Hou, and M. Li, "Sams-net: A sliced attention-based neural network for music source separation.," *2021 12th International Symposium on Chinese Spoken Language Processing (ISCSLP), Chinese Spoken Language Processing (ISCSLP), 2021 12th International Symposium on*, pp. 1 – 5, 2021.

[2] E. C. Cherry, "Some experiments on the recognition of speech, with one and with two ears.," *Journal of the Acoustical Society of America*, vol. 25, pp. 975 – 979, 1953.

[3] A. Défossez, N. Usunier, L. Bottou, and F. Bach, "Music source separation in the waveform domain," *arXiv preprint arXiv:1911.13254*, 2019.

[4] R. Das, D. Deshwal, P. Sangwan, and N. Nehra, "Music source separation: A guide," in *2021 International Conference on Industrial Electronics Research and Applications (ICIERA)*, pp. 1–4, 2021.

[5] R. Hennequin, A. Khlif, F. Voituret, and M. Moussallam, "Spleeter: a fast and efficient music source separation tool with pre-trained models," *Journal of Open Source Software*, vol. 5, no. 50, p. 2154, 2020. Deezer Research.

[6] "Lalal.ai presents phoenix, a new step in evolution of audio source separation." `https://www.lalal.ai/blog/phoenix-neural-network-vocal-separation/`. Accessed: 2022-05-02.

[7] "Remix software & stem extraction for djs & more - ripx deepremix." `https://hitnmix.com/remix-software/`. Accessed: 2022-05-02.

[8] E. Manilow, G. Wichern, and J. L. Roux, "Hierarchical musical instrument separation," in *International Society for Music Information Retrieval Conference*, 2020.

[9] W.-H. Lai and S.-L. Wang, "Separation of electric guitar sound based on stacked recurrent neural network," in *2020 International Conference on Technologies and Applications of Artificial Intelligence (TAAI)*, pp. 52–55, 2020.

[10] E. Manilow, P. Seetharman, and J. Salamon, *Open Source Tools & Data for Music Source Separation*. https://source-separation.github.io/tutorial, October 2020.

[11] J. Schneider, *The contemporary guitar*. Rowman & Littlefield, 2015.

[12] C. Rascon, "A corpus-based evaluation of beamforming techniques and phase-based frequency masking," *Sensors*, vol. 21, no. 15, p. 5005, 2021.

[13] S. N. Jain and C. Rai, "Blind source separation and ica techniques: a review," *International Journal of Engineering Science and Technology*, vol. 4, no. 4, pp. 1490–1503, 2012.

[14] E. Cano, D. FitzGerald, A. Liutkus, M. D. Plumbley, and F.-R. Stöter, "Musical source separation: An introduction," *IEEE Signal Processing Magazine*, vol. 36, no. 1, pp. 31–40, 2019.

[15] "Audacity | free, open source, cross-platform audio software for multi-track recording and editing.." `https://audacityteam.org/`. Accessed: 2023-02-05.

[16] C. Shannon, "Communication in the presence of noise," *Proceedings of the IRE*, vol. 37, no. 1, pp. 10–21, 1949.

[17] J. O. Smith, *Spectral Audio Signal Processing*. http://ccrma.stanford.edu/ jos/sasp/, 2011.

[18] "Spek – free acoustic spectrum analyzer / spectrogram viewer." `http://spek.cc/`. Accessed: 2023-02-07.

[19] D. Griffin and J. Lim, "Signal estimation from modified short-time fourier transform," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 32, no. 2, pp. 236–243, 1984.

[20] D. Gunawan and D. Sen, "Iterative phase estimation for the synthesis of separated sources from single-channel mixtures," *IEEE Signal Processing Letters*, vol. 17, no. 5, pp. 421–424, 2010.

[21] E. Vincent, R. Gribonval, and C. Fevotte, "Performance measurement in blind audio source separation," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 14, no. 4, pp. 1462–1469, 2006.

[22] Y. Luo and N. Mesgarani, "Conv-tasnet: Surpassing ideal time–frequency magnitude masking for speech separation," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 27, no. 8, pp. 1256–1266, 2019.

[23] F.-R. Stöter, S. Uhlich, A. Liutkus, and Y. Mitsufuji, "Open-unmix - a reference implementation for music source separation," *Journal of Open Source Software*, 2019.

[24] Z. Rafii and B. Pardo, "Repeating pattern extraction technique (repet): A simple method for music/voice separation," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 21, no. 1, pp. 73–84, 2013.

[25] Z. Rafii and B. Pardo, "Music/voice separation using the similarity matrix," in *Proceedings of the 13th International Society for Music Information Retrieval Conference, ISMIR 2012*, pp. 583–588, Dec. 2012. 13th International Society for Music Information Retrieval Conference, ISMIR 2012 ; Conference date: 08-10-2012 Through 12-10-2012.

[26] P. Seetharaman, F. Pishdadian, and B. Pardo, "Music/voice separation using the 2d fourier transform," in *2017 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, pp. 36–40, 2017.

[27] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. W. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio," *CoRR*, vol. abs/1609.03499, 2016.

[28] Y. Luo and N. Mesgarani, "Tasnet: Time-domain audio separation network for real-time, single-channel speech separation," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 696–700, 2018.

[29] A. Jansson, E. J. Humphrey, N. Montecchio, R. M. Bittner, A. Kumar, and T. Weyde, "Singing voice separation with deep u-net convolutional networks," in *18th International Society for Music Information Retrieval Conference, ISMIR*, October 2017.

[30] 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.

[31] E. Tzinis, Z. Wang, and P. Smaragdis, "Sudo rm-rf: Efficient networks for universal audio source separation," in *2020 IEEE 30th International Workshop on Machine Learning for Signal Processing (MLSP)*, pp. 1–6, IEEE, 2020.

[32] V. S. Kadandale, J. F. Montesinos, G. Haro, and E. Gómez, "Multi-channel u-net for music source separation," in *2020 IEEE 22nd International Workshop on Multimedia Signal Processing (MMSP)*, pp. 1–6, IEEE, 2020.

[33] J. Chen, Q. Mao, and D. Liu, "Dual-path transformer network: Direct context-aware modeling for end-to-end monaural speech separation," October 2020.

[34] N. Takahashi and Y. Mitsufuji, "D3net: Densely connected multidilated densenet for music source separation," October 2020.

[35] J. R. Hershey, Z. Chen, J. Le Roux, and S. Watanabe, "Deep clustering: Discriminative embeddings for segmentation and separation," in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 31–35, 2016.

[36] Y. Luo, Z. Chen, J. R. Hershey, J. Le Roux, and N. Mesgarani, "Deep clustering and conventional networks for music separation: Stronger together," in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 61–65, 2017.

[37] E. Vincent, R. Gribonval, C. Févotte, A. Nesbit, M. D. Plumbley, M. E. Davies, and L. Daudet, "Bass-db multitrack recordings," Jun 2005.

[38] E. Vincent, H. Sawada, P. Bofill, S. Makino, and J. P. Rosca, "First stereo audio source separation evaluation campaign: Data, algorithms and results," in *Independent Component Analysis and Signal Separation* (M. E. Davies, C. J. James, S. A. Abdallah, and M. D. Plumbley, eds.), (Berlin, Heidelberg), pp. 552–559, Springer Berlin Heidelberg, 2007.

[39] M. Vinyes, "MTG MASS dataset." `http://mtg.upf.edu/download/datasets/mass`, 2008.

[40] R. Bittner, J. Salamon, M. Tierney, M. Mauch, C. Cannam, and J. Bello, "Medleydb: A multitrack dataset for annotation-intensive mir research," *Proceedings - 15th International Society for Music Information Retrieval Conference (ISMIR 2014)*, 10 2014.

[41] R. M. Bittner, J. Wilkins, H. Yip, and J. P. Bello, "Medleydb 2.0: New data and a system for sustainable data collection," *Proceedings - 17th International Society for Music Information Retrieval Conference (ISMIR 2016)*, 2016.

[42] A. Liutkus, F.-R. Stöter, Z. Rafii, D. Kitamura, B. Rivet, N. Ito, N. Ono, and J. Fontecave, "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. Michel, and N. Thirion-Moreau, eds.), (Cham), pp. 323–332, Springer International Publishing, 2017.

[43] Z. Rafii, A. Liutkus, F.-R. Stöter, S. I. Mimilakis, and R. Bittner, "The MUSDB18 corpus for music separation," Dec. 2017.

[44] Z. Rafii, A. Liutkus, F.-R. Stöter, S. I. Mimilakis, and R. Bittner, "Musdb18-hq - an uncompressed version of musdb18," Aug. 2019.

[45] E. Manilow, G. Wichern, P. Seetharaman, and J. Le Roux, "Cutting music source separation some slakh: A dataset to study the impact of training data quality and quantity,"

in *2019 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, pp. 45–49, 2019.

[46] "Deezer internal datasets." `https://github.com/deezer/spleeter/issues/802`. Accessed: 2022-12-13.

[47] P. Chandna, M. Miron, J. Janer, and E. Gómez, "Monoaural audio source separation using deep convolutional neural networks," in *International conference on latent variable analysis and signal separation*, pp. 258–266, Springer, 2017.

[48] Y. Luo, Z. Chen, J. R. Hershey, J. Le Roux, and N. Mesgarani, "Deep clustering and conventional networks for music separation: Stronger together," in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 61–65, 2017.

[49] Y. Luo, Z. Chen, J. R. Hershey, J. Le Roux, and N. Mesgarani, "Deep clustering and conventional networks for music separation: Stronger together," in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 61–65, 2017.

[50] H.-S. Choi, J. Kim, J. Huh, A. Kim, J.-W. Ha, and K. Lee, "Phase-aware speech enhancement with deep complex u-net," *CoRR*, vol. abs/1903.03107, 2019. Withdrawn.

[51] C. Veaux, J. Yamagishi, and S. King, "The voice bank corpus: Design, collection and data analysis of a large regional accent speech database," in *2013 International Conference Oriental COCOSDA held jointly with 2013 Conference on Asian Spoken Language Research and Evaluation (O-COCOSDA/CASLRE)*, pp. 1–4, 11 2013.

[52] D. Samuel, A. Ganeshan, and J. Naradowsky, "Meta-learning extractors for music source separation," in *2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 816–820, 2020.

[53] "Partes de la guitarra acústica (guía completa)." `https://musicapod.com/partes-de-la-guitarra-acustica/`. Accessed: 2022-06-22.

[54] "Partes de la guitarra: Cuáles son y para qué sirven." https://promocionmusical.es/lecciones-de-guitarra/partes-de-la-guitarra/. Accessed: 2022-06-22.

[55] "Los elementos de la música." https://oscrove.wordpress.com/teoria-musical/los-elementos-de-la-musica/. Accessed: 2022-07-5.

[56] "La música: ritmo, melodía y armonía. fundamentos." https://colaboratorio.net/xphnx/multimedia/audio/2017/la-musica-ritmo-melodia-y-armonia/. Accessed: 2022-07-5.

[57] "Introducción a midi." https://www.prometec.net/midi-introduccion/. Accessed: 2022-07-10.

[58] "Midi files." https://mido.readthedocs.io/en/latest/midi_files.html. Accessed: 2022-07-17.

[59] "Midi a wav." https://audio.online-convert.com/es/convertir/midi-a-wav. Accessed: 2022-07-18.

[60] "Midi to wav converter." https://www.freeconvert.com/midi-to-wav. Accessed: 2022-07-18.

[61] M. Pariente, S. Cornell, J. Cosentino, S. Sivasankaran, E. Tzinis, J. Heitkaemper, M. Olvera, F.-R. Stöter, M. Hu, J. M. Martín-Doñas, D. Ditter, A. Frank, A. Deleforge, and E. Vincent, "Asteroid: the PyTorch-based audio source separation toolkit for researchers," in *Proc. Interspeech*, 2020.

[62] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), pp. 8024–8035, Curran Associates, Inc., 2019.

[63] W. Falcon and The PyTorch Lightning team, "PyTorch Lightning," March 2019.

[64] Y.-Y. Yang, M. Hira, Z. Ni, A. Chourdia, A. Astafurov, C. Chen, C.-F. Yeh, C. Puhrsch, D. Pollack, D. Genzel, D. Greenberg, E. Z. Yang, J. Lian, J. Mahadeokar, J. Hwang, J. Chen, P. Goldsborough, P. Roy, S. Narenthiran, S. Watanabe, S. Chintala, V. Quenneville-Bélair, and Y. Shi, "Torchaudio: Building blocks for audio and speech processing," *arXiv preprint arXiv:2110.15018*, 2021.

[65] B. McFee, A. Metsai, M. McVicar, S. Balke, C. Thomé, C. Raffel, F. Zalkow, A. Malek, Dana, K. Lee, O. Nieto, D. Ellis, J. Mason, E. Battenberg, S. Seyfarth, R. Yamamoto, viktorandreevichmorozov, K. Choi, J. Moore, R. Bittner, S. Hidaka, Z. Wei, nullmightybofo, A. Weiss, D. Hereñú, F.-R. Stöter, L. Nickel, P. Friesch, M. Vollrath, and T. Kim, "librosa/librosa: 0.9.2," June 2022.

[66] F.-R. Stöter, A. Liutkus, and N. Ito, "The 2018 signal separation evaluation campaign," in *Latent Variable Analysis and Signal Separation: 14th International Conference, LVA/ICA 2018, Surrey, UK*, pp. 293–305, 2018.

[67] The pandas development team, "pandas-dev/pandas: Pandas."