UNTREF UNIVERSIDAD NACIONAL DE TRES DE FEBRERO

INGENIERÍA DE SONIDO

Separación de fuentes musicales mediante redes neuronales convolucionales †

Autor: Leonardo Pepino
Tutor: Dr. Laurence Bender

(†) Tesis para optar al título de Ingeniero de Sonido

Marzo 2019

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

LISTA DE ACRÓNIMOS

ADAM	Estimación adaptativa de momento
APS	Puntaje perceptual asociado a los artefactos
ASA	Análisis de escenas auditivas
BN	Normalización por lotes
BSS	Separación ciega de fuentes
CNN	Red neuronal convolucional
COLA	Solapamiento y suma constante
DA	Aumento artificial de datos
DFT	Transformada discreta de Fourier
DNN	Red neuronal profunda
DSD100	Demixing secrets dataset
ELU	Unidad exponencial lineal.
GAN	Red adversaria generativa
HPSS	Separación de fuentes armónicas y percusivas
IPS	Puntaje perceptual asociado a las interferencias
ISR	Relación imagen a fuente
LSTM	Memoria de corto y largo plazo
MIR	Recuperación de información musical
MLP	Perceptrón multicapa.
NMF	Factorización no negativa de matrices
OPS	Puntaje perceptual general
PEASS	Métodos de evaluación perceptual de separación de fuentes de audio
PSM	Medida de similaridad perceptual
ReLU	Unidad lineal rectificada.
SAR	Relación señal a artefactos
SDR	Relación señal a distorsión
SIR	Relación señal a interferencia
STFT	Transformada de corto término de Fourier
TPS	Puntaje perceptual asociado a la fuente objetivo

RESUMEN

En esta investigación se aborda el estudio de las técnicas de separación ciega de fuentes acústicas, en particular, fuentes musicales en mezclas estereofónicas. Se presenta el diseño, la implementación y la evaluación de un sistema que permite separar los distintos instrumentos musicales en mezclas de audio profesionalmente producidas. Se utilizan técnicas de procesamiento de señales combinadas con modelos de redes neuronales convolucionales, conforme al estado del arte actual. Se analiza el impacto de distintas técnicas de procesamiento aplicadas a los datos de entrada y salida, la influencia de la arquitectura de la red neuronal y el uso de aumento artificial de datos. El sistema final se evalúa utilizando métricas objetivas que muestran resultados comparables con trabajos recientes de separación de fuentes musicales. Por último, se proponen posibles mejoras al sistema desarrollado y futuras líneas de investigación.

Palabras clave: “Separación de Fuentes Musicales”; “Redes Neuronales Convolucionales”; “DSD100”;

ABSTRACT

In this research, techniques for blind source separation of acoustic sources are studied. A framework which allows the separation of multiple musical instruments in a stereophonic audio mixture is designed, developed and evaluated. Techniques from signal processing are combined with convolutional neural network models, according to the current state of the art. The impact of different methods used to process the inputs and outputs, the influence of the neural network architecture, and the use of data augmentation, is analyzed. The final system is evaluated using objective measures which give results comparable with recent research on music source separation. Finally, possible improvements to the framework and future lines of research are proposed.

Keywords: “Music Source Separation”; “Convolutional Neural Networks”; “DSD100”;

AGRADECIMIENTOS

Esta tesis es la culminación de un viaje de 5 años en donde conocí a excelentes personas y viví agradables experiencias para formarme como profesional. Por eso agradezco a todos los que me han acompañado.

En primer lugar a las autoridades de la Universidad Nacional de Tres de Febrero (UNTREF), a su Rector Lic. Anibal Jozami, a todo su personal docente y no docente, en especial a las áreas de Biblioteca y Pañol Multimedia de la UNTREF.

A los docentes de la carrera de Ingeniería de Sonido de la UNTREF y a su coordinador Alejandro Bidondo.

A Laurence Bender quien brindó una tutoría excepcional aportando mucho de su tiempo y ayudándome a organizar las ideas y fijar un rumbo concreto.

A Joaquín Mansilla y Daniel Ottobre por acompañar el proceso de elaboración de esta tesis.

A mis compañeros y amigos que siempre muestran interés y curiosidad por lo que hago y me alientan a seguir adelante.

A mi familia, en especial a mis padres, Alberto y Alicia, quienes han dispuesto todo para que llegue a esta instancia y me han apoyado siempre.

A la comunidad online que genera herramientas y conocimiento libre y gratuito.

Leonardo Pepino

CAPÍTULO 1: INTRODUCCIÓN

1.1 FUNDAMENTACIÓN

La separación de fuentes consiste en la extracción de una o varias señales de interés de una mezcla que contiene múltiples señales superpuestas. Esta operación es de suma importancia en muchas aplicaciones del procesamiento de señales, ya que remover las fuentes indeseadas (como el ruido), o segregar las de interés, permite realizar operaciones con señales sin interferencias. Esta investigación se centra en la separación de instrumentos musicales a partir de mezclas de audio estereofónicas.

Es conocida la habilidad del ser humano para separar distintas fuentes acústicas, fenómeno que Cherry denominó efecto de fiesta de cóctel [1]. Si bien se han logrado grandes avances en los últimos años en cuanto a simular esta habilidad por computadora, aún se continúa en un constante proceso de desarrollo y mejoras dentro del campo.

La separación de fuentes musicales posee numerosas aplicaciones en el ámbito de la música y de las ciencias de la grabación, entre ellas:

- Remover instrumentos musicales de una mezcla para por ejemplo, realizar “karaoke” [2].
- Manipular instrumentos musicales separados para remezclar, generar sonido envolvente (upmixing) [3], procesar un solo instrumento de la mezcla sin afectar a los demás y restaurar grabaciones antiguas [4], entre otras aplicaciones.
- Analizar instrumentos musicales por separado para por ejemplo, realizar transcripción automática de piezas musicales [5] y analizar acústica o musicalmente un instrumento aislado [6].
- Extraer muestras de audio para su posterior uso en composiciones musicales (remixing).

1.2 OBJETIVOS

1.2.1 Objetivo general

Diseñar, implementar y evaluar un sistema computacional de separación de instrumentos musicales en mezclas estereofónicas (específicamente bajo, batería y voz).

1.2.2 Objetivos específicos

- Realizar un relevamiento bibliográfico de las técnicas existentes para separación de fuentes sonoras.
- Analizar distintas formas de preprocesamiento de las entradas y postprocesamiento de las salidas y evaluar su efecto en el resultado de separación.
- Implementar distintas arquitecturas de redes neuronales profundas y evaluar objetivamente su desempeño en separación de fuentes musicales.
- Generar código en Python que permita realizar separaciones de fuentes musicales[1]
- Seleccionar la combinación de técnicas que mejores resultados objetivos de separación produzca para desarrollar el sistema final.
- Evaluar objetivamente, con un conjunto de datos de prueba, el sistema final.
- Reportar, analizar y comparar los resultados obtenidos con los de otros sistemas existentes.

1.3 ESTRUCTURA DE LA INVESTIGACIÓN

En el capítulo 2 se presenta el marco teórico necesario para la comprensión del estudio, el cual trata principalmente tres temáticas: la transformada de tiempo corto de Fourier (STFT), redes neuronales artificiales y separación de fuentes sonoras. En el capítulo 3 se expone el estado del arte de la separación de fuentes musicales, que es la temática principal de este estudio. En el capítulo 4 se describe el sistema desarrollado, su implementación y la forma de evaluación del mismo. En el capítulo 5 se reportan los principales resultados obtenidos con el sistema propuesto, y en el capítulo 6 se analizan y discuten los mismos. En el capítulo 7 se exponen las conclusiones a las que se arribó en el estudio. Por último, en el capítulo 8 se proponen futuras líneas de investigación en base al análisis de resultados realizado.

[1] El código desarrollado junto a ejemplos de separación se encuentran accesibles públicamente en el siguiente repositorio de GitHub: https://github.com/mrpep/mss-tesis

CAPÍTULO 2: MARCO TEÓRICO

2.1 REPRESENTACIÓN DE SEÑALES SONORAS

El sonido se produce por una perturbación que se propaga en un medio elástico y causa una alteración de la presión o un movimiento de las partículas del material que puede ser percibido por una persona o medido con un instrumento [7]. Las perturbaciones sonoras pueden ser capturadas por un micrófono y representadas como una forma de onda, en la cual el eje de abscisas corresponde al tiempo y el eje de ordenadas a la amplitud. Si bien en la forma de onda se encuentra toda la información de la señal, su interpretación directa no es sencilla. Es por esto que a menudo se realiza una transformación sobre la señal que revele información útil, pasándose al dominio de frecuencia o tiempo-frecuencia.

En el caso del audio digital, la transformada discreta de Fourier (DFT) permite pasar una señal sonora discreta $x[n]$ con N muestras del dominio del tiempo a una representación discreta (espectro) en el dominio de frecuencia. Esta se define como:

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-j2\pi \frac{k}{N} n} \qquad k = 0, 1, 2, ..., N-1 \quad (2.1)$$

La DFT es invertible, por lo cual, es posible recuperar la señal en tiempo mediante la siguiente expresión:

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] e^{j2\pi \frac{k}{N} n} \qquad n = 0, 1, 2, ..., N-1 \quad (2.2)$$

2.1.1 Transformada de corto término de Fourier (STFT)

Las señales musicales muestran variaciones significativas de su espectro a lo largo del tiempo, aunque durante intervalos cortos suele mantenerse aproximadamente estacionario. La transformada de corto término de Fourier (STFT) consiste en calcular el espectro en ventanas cortas $w[n]$ de N muestras que se desplazan H muestras en tiempo [8]. Esta operación aplicada sobre una señal discreta $x[n]$ se expresa matemáticamente como:

$$X[t, k] = \sum_{n=0}^{N-1} w[n] x[tH + n] e^{-j \frac{2\pi k n}{N}} \quad (2.3)$$

La variable H se denomina tamaño de salto (hop size) y determina el factor de solapamiento de las ventanas en el tiempo. En la Figura 2.1 se ilustra el proceso descrito para obtener la STFT de una señal.

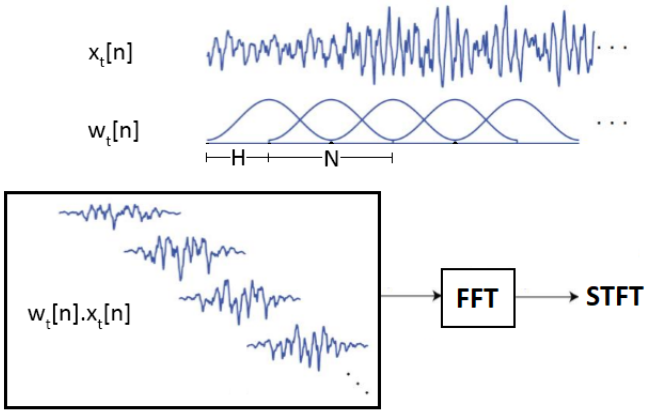

Figura 2.1: Proceso de obtención de la STFT. Extraído y adaptado de [8].

Los coeficientes de la STFT resultante forman una matriz con valores complejos, por lo que puede calcularse una magnitud y una fase. Se suele construir a partir de la magnitud una representación gráfica denominada espectrograma, la cual normalmente presenta el tiempo en el eje de abscisas y la frecuencia expresada en Hz en el eje de ordenadas. El valor absoluto de la STFT en cada punto tiempo-frecuencia es representado en un tercer eje, utilizando frecuentemente una escala de colores o grises.

2.1.2 Importancia del enventanado

La elección de la ventana $w[n]$ es importante para el cálculo de la STFT. Multiplicar dos señales en tiempo, por el teorema de modulación, es equivalente a una convolución en frecuencia. Por este motivo, la ventana utilizada distorsiona el espectro observado. Como se ilustra en la Figura 2.2, si se tiene una señal sinusoidal con su espectro (a) multiplicada por una ventana rectangular (b), el espectro de la señal resultante se distorsiona (c). Existen dos tipos de distorsión espectral:

- **Fuga (Leakage):** es ocasionada por los lóbulos secundarios del espectro de la ventana. Produce picos en frecuencias incorrectas o cambia la amplitud de los mismos.
- **Manchado (Smearing):** es ocasionado por el ancho del lóbulo principal del espectro de la ventana. Cuanto más ancho es, mayor la pérdida de resolución en frecuencia.

Se puede demostrar que es imposible reducir simultáneamente la amplitud de los lóbulos secundarios y el ancho del lóbulo principal. Por este motivo, se han diseñado un gran número de ventanas para disminuir la distorsión espectral. La ventana de Hann exhibe un buen compromiso entre ancho de lóbulo principal (decae 6 dB en dos muestras de frecuencia) y amplitud de lóbulos secundarios (se encuentra el primero 32 dB por debajo del principal) [9].

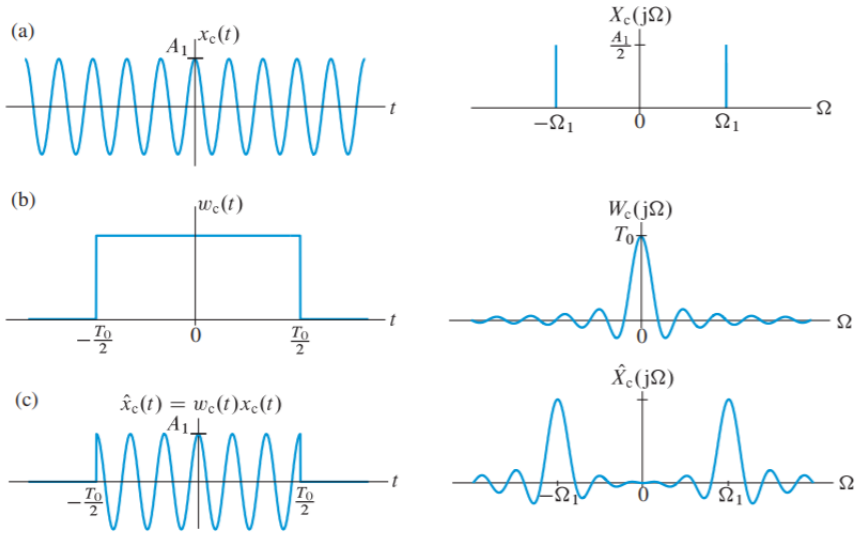

Figura 2.2: Efectos de distorsión espectral. Extraído de [9].

Otro aspecto muy importante en el cálculo de la STFT es el tamaño N de la ventana. Su elección determina la resolución en tiempo y frecuencia del análisis. Se puede probar que no es posible obtener infinita resolución en ambos dominios a la vez [10]. Una alta resolución en tiempo, que se obtiene mediante ventanas cortas con valores pequeños de N, supondrá una baja resolución en frecuencia. Contrariamente, una baja resolución en tiempo usando ventanas largas supondrá una alta resolución en frecuencia.

En el marco del análisis de señales musicales, una ventana corta permite una buena localización en tiempo de los eventos percusivos, pero no permite localizar en frecuencia eventos más persistentes como notas producidas por un instrumento. Este problema de la localización en frecuencia se agrava en el caso de instrumentos musicales con un importante contenido espectral en baja frecuencia, como por ejemplo, un bajo. Por el contrario, una ventana larga dificulta la localización en tiempo de eventos percusivos y los comienzos (onsets) y finales (offsets) de notas. El uso de una ventana fija en la STFT es una de sus principales desventajas. Existen métodos de análisis multiresolución que permiten superar estos inconvenientes, como el uso de wavelets [11], pero no serán tratados en esta investigación.

2.1.3 Información de fase

En tareas de procesamiento de audio es común descartar la información de fase y operar solamente sobre la magnitud de la STFT. Esto se debe en gran parte a que la fase no muestra estructuras tan marcadas como la magnitud.

Sin embargo, la derivada de la fase respecto al tiempo (frecuencia instantánea), así como la derivada negativa respecto a la frecuencia (retardo de grupo), pueden proporcionar estructuras relevantes de la señal bajo análisis. Como ejemplo, en la Figura 2.3 se muestra la

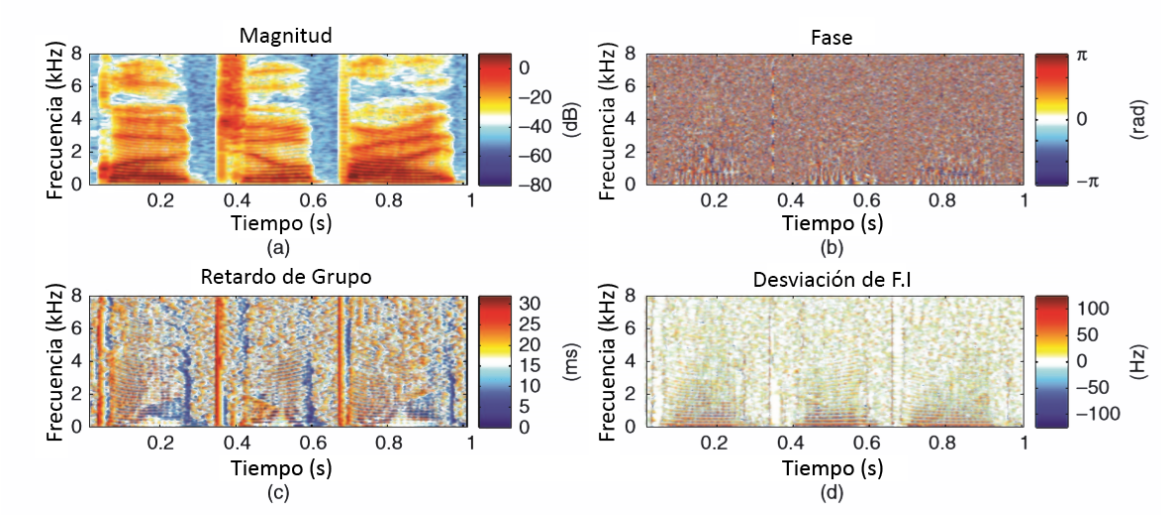

Figura 2.3: Espectrogramas de magnitud, fase, retardo de grupo y frecuencia instantánea de una señal. Extraído de [12].

magnitud (a), la fase (b), el retardo de grupo (c) y la desviación de la frecuencia instantánea respecto a la central (d) de la STFT de una frase hablada. La información de fase se puede aprovechar para lograr una localización en tiempo y frecuencia más precisa [13].

Por otro lado, una gran cantidad de estudios psicoacústicos han concluido que las distorsiones de fase son audibles [14, 15, 16], por lo que descartar la información de fase produce un deterioro perceptible de la señal de audio.

2.1.4 Recuperación de la señal en tiempo

Por lo general, en las tareas de procesamiento de audio, después de calcular la STFT de la señal (etapa de análisis), se modifica el espectrograma resultante y por último se recupera la señal en tiempo (etapa de síntesis).

En la etapa de síntesis, para recuperar la señal en tiempo se suele aplicar la técnica de solapamiento y suma (overlap-add). Esta consiste en calcular la DFT inversa para cada cuadro t de la STFT y sumar las señales resultantes desplazándolas tH muestras. Matemáticamente la operación se expresa como:

$$x[n] = \sum_{t=1}^{L-1} Shift_{tH,n}[\frac{1}{N} \sum_{k=0}^{N-1} x[t,k] e^{j\frac{2\pi kn}{N}}] \tag{2.4}$$

Para una reconstrucción correcta hay que tener en cuenta el efecto de la ventana utilizada, la cual debe cumplir el requerimiento de solapamiento y suma constante (COLA):

$$\sum_{t=0}^{L-1} w[n - tH] = \alpha \tag{2.5}$$

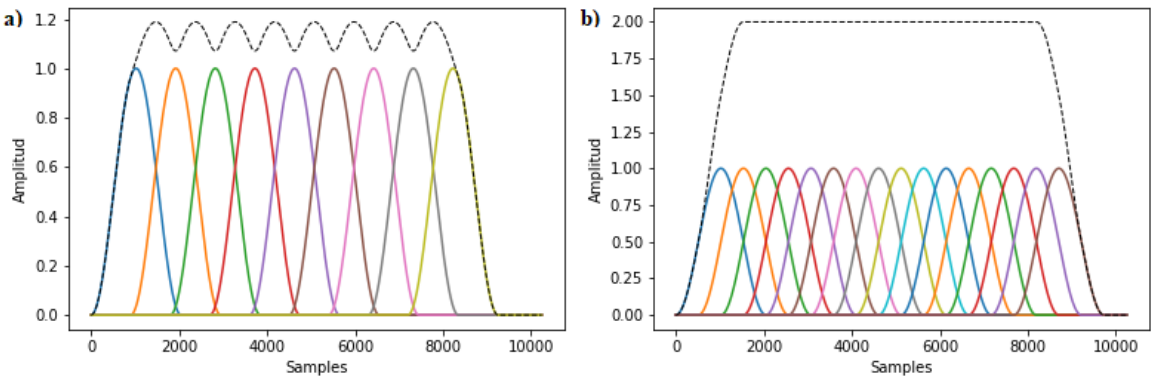

Figura 2.4: Efecto del solapamiento de las ventanas en la STFT.

donde α es un valor constante. En el caso de la ventana de Hann, para lograr una reconstrucción correcta, la relación $\frac{N}{H}$ debe ser un número entero mayor a 1. Esto se ilustra en la Figura 2.4, la cual muestra el efecto de un enventanado incorrecto (a) realizado con un valor de $\frac{N}{H} = 2,27$, y un enventanado correcto (b) realizado con un valor de $\frac{N}{H} = 4$. Se puede observar que no cumplir con el requerimiento COLA provoca una modulación de baja frecuencia en la amplitud de la señal recuperada en tiempo.

A su vez, al ser modificada la STFT de una señal, no siempre se puede garantizar que exista una señal en el dominio del tiempo cuya STFT corresponda al espectrograma modificado. Cuando esto ocurre se dice que el espectrograma es inconsistente. En este estudio se utiliza la técnica no iterativa propuesta por Griffin y Lim en [17] para estimar la señal en tiempo. Este método es similar al de solapamiento y suma pero su formulación está basada en minimizar el error cuadrático entre el espectrograma modificado y el espectrograma de la señal estimada. De esta forma, si el espectrograma a invertir es inconsistente, la STFT de la señal estimada diferirá de manera mínima. Otra técnica propuesta en [17] permite una reconstrucción aproximada de la señal en el dominio del tiempo a partir de la magnitud de la STFT, estimando iterativamente la información de fase.

2.2 REDES NEURONALES ARTIFICIALES

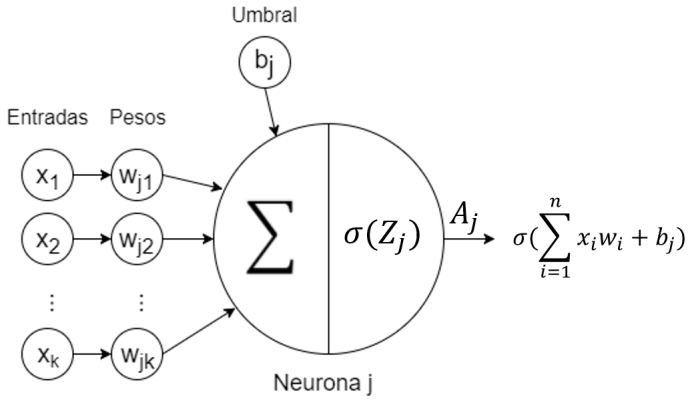

Figura 2.5: Esquema de una neurona artificial básica.

Las redes neuronales artificiales son sistemas de mapeos no lineales cuya estructura se basa en principios observados en los sistemas nerviosos de los animales [18]. Su unidad básica es la neurona artificial, propuesta por primera vez por McCulloch y Pitts [19] en 1943.

2.2.1 Neurona artificial

Una neurona artificial genérica se ilustra en la Figura 2.5 y consta de:

- **Entradas (x_k):** reciben las salidas de otras neuronas o los datos de entrada a la red neuronal artificial.
- **Pesos sinápticos (w_{jk}):** son parámetros que determinan el efecto de una entrada en la neurona artificial. Sus valores son modificados durante la fase de entrenamiento.

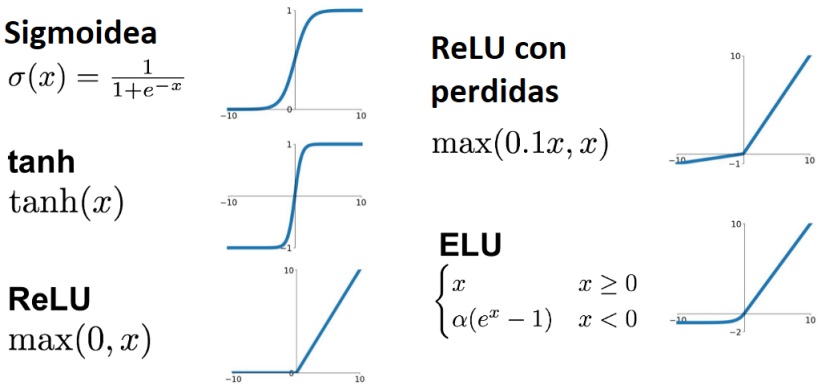

Figura 2.6: Funciones de activación comúnmente utilizadas en redes neuronales artificiales[1].

[1]Extraído y adaptado de https://medium.com/@shrutijadon10104776/survey-on-activation-functions-for-deep-learning-9689331ba092

- **Umbral o bias (b_j):** es una entrada externa a la neurona artificial. Su valor es modificado durante la fase de entrenamiento junto a los pesos sinápticos.
- **Regla de propagación:** Determina la entrada efectiva (Z_j) de una unidad a partir de las entradas. Habitualmente se usa una suma ponderada:

$$Z_j = \sum_k w_{jk} x_k + b_j \tag{2.6}$$

- **Función de activación ($\sigma(Z_j)$):** determina la salida de la neurona artificial a partir de aplicar una función matemática a la entrada efectiva Z_j. Permite introducir alinealidades en el modelo. Algunas de las funciones de activación más utilizadas son la función logística, la tangente hiperbólica, unidades lineales rectificadas (ReLU), ReLU con perdidas (Leaky ReLU) y unidades lineales exponenciales (ELU). Estas funciones se ilustran en la Figura 2.6.
- **Salida(A_j):** es el resultado de transformar la entrada efectiva a la neurona mediante la función de activación y se puede expresar, utilizando la suma ponderada como regla de propagación, de la siguiente manera:

$$A_j = \sigma(\sum_k w_{jk} x_k + b_j) \tag{2.7}$$

2.2.2 Capas completamente conectadas

Las neuronas artificiales descritas suelen agruparse en capas. Una de las primeras redes neuronales exitosas fue el perceptrón, desarrollado en 1958 por Rosenblatt [20]. El perceptrón consistía de una sola capa de neuronas artificiales que producían una salida deseada a partir de una entrada. Este sistema era capaz de aprender tareas básicas de clasificación ajustando sus pesos sinápticos para reducir una función de costo. Sin embargo, poseía limitaciones como la imposibilidad de aprender una compuerta XOR [21]. No fue hasta el redescubrimiento del algoritmo de propagación de error hacia atrás (backpropagation) en 1986 [22] que fue posible entrenar redes neuronales con múltiples capas.

La incorporación de una capa de neuronas entre la entrada y la salida (capa oculta), permitió resolver el problema XOR y aproximar una gran variedad de funciones [23]. En la Figura 2.7 se puede observar un ejemplo de estas redes, denominadas perceptrones multicapa (MLP), la cual en el caso ilustrado posee 5 variables de entrada, 3 neuronas en la capa oculta y 2 en la salida. A partir de un conjunto de variables de entrada, el modelo predice un conjunto de variables de salida. Internamente, cada neurona se encuentra conectada con todas las de la capa previa, por eso sus capas se denominan completamente conectadas.

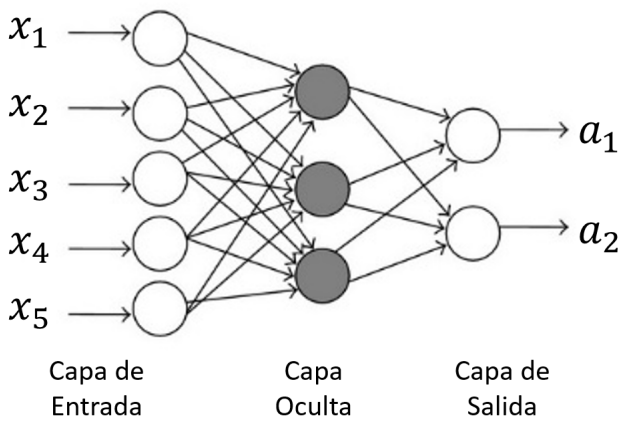

Figura 2.7: Diagrama de un perceptrón multicapa (MLP)[2].

2.2.3 Aprendizaje de la red

En el estudio del aprendizaje por máquina (Machine Learning), existen tres formas de aprendizaje [24]:

- **Aprendizaje supervisado:** se aprende mediante la observación de ejemplos. Estos ejemplos forman un conjunto de entrenamiento y proveen tanto los datos de las entradas al modelo como los valores de salida deseados. De esta manera, mediante ejemplos, el modelo aprende una función que mapea las entradas a las salidas deseadas.
- **Aprendizaje no supervisado:** se aprenden patrones relevantes en los datos de entrada sin disponer de los datos de salida deseados.
- **Aprendizaje por refuerzo:** el modelo aprende mediante recompensas y castigos.

En esta investigación se utilizan técnicas de aprendizaje supervisado. En una red neuronal, el proceso de aprendizaje puede sintetizarse en cuatro pasos:

1. **Prealimentación (Feedforward):** se alimenta a la red neuronal con un conjunto de datos de entrada y se obtienen las respectivas salidas.
2. **Cálculo del error:** Las salidas de la red neuronal en general diferirán de las salidas deseadas. Se define y calcula una función de costo, la cual proporciona una medida del error. Una de las funciones de costo más utilizadas es el error cuadrático medio.
3. **Propagación hacia atrás del error (backpropagation):** El error en la salida es propagado hacia las demás capas de la red. De esta manera, es posible computar el gradiente de la función de costo respecto a los pesos y umbrales de la red neuronal.

[2] Adaptado de https://medium.com/pankajmathur/a-simple-multilayer-perceptron-with-tensorflow-3effe7bf3466.

4. **Actualización de los parámetros:** en base al gradiente computado, se actualizan los pesos y umbrales de la red neuronal con el fin de disminuir el error en la salida. Este ajuste de los parámetros, por lo general se realiza mediante descenso por gradiente o variantes del mismo [25]. Se define una tasa de aprendizaje (η) que controla cuánto son ajustados los parámetros de la red respecto al gradiente de la función de costo. Si el gradiente se computa en base a una sola muestra de los datos por vez, la técnica se denomina descenso por gradiente estocástico (SGD). Si se utiliza un número mayor de muestras para cada ajuste del gradiente se denomina descenso por gradiente con mini-lotes. En esta investigación se utilizan mini-lotes.

Estos cuatro pasos son repetidos múltiples veces. Se completa una época cuando todos los ejemplos de la base de datos de entrenamiento son ingresados a la red. Normalmente las redes neuronales se entrenan durante varias épocas hasta que el error converge a un valor deseado. En el Anexo I se realiza un análisis matemático del proceso de aprendizaje mediante propagación hacia atrás del error y descenso por gradiente.

Uno de los grandes desafíos del aprendizaje por maquina es que los modelos deben desempeñarse bien con datos nuevos que nunca hayan sido presentados como entrada. Es decir, los modelos deben generalizar. Se suele dividir las bases de datos en varios conjuntos:

- **Conjunto de entrenamiento:** son los datos sobre los cuales se realiza la optimización de los parámetros del modelo, en este caso, los pesos y umbrales de la red neuronal.
- **Conjunto de validación:** son los datos sobre los cuales el modelo realiza predicciones y se evalúa el error durante el proceso de entrenamiento. Sirve para verificar el comportamiento del modelo y modificar sus hiperparámetros[3].
- **Conjunto de prueba:** son los datos sobre los cuales se realiza la evaluación final del modelo. Sirve para medir el error de generalización del mismo.

Los dos objetivos principales de un algoritmo de aprendizaje por máquina son:

- Minimizar el error de entrenamiento.
- Minimizar la diferencia entre el error de entrenamiento y el error de prueba.

Cuando no se logra obtener un error de entrenamiento lo suficientemente bajo se dice que hay un subajuste. Esto ocurre porque el modelo no tiene una capacidad de representa-

[3]Un hiperparámetro es un parámetro cuyo valor es determinado antes de iniciar el entrenamiento. Algunos ejemplos de hiperparámetros en una red neuronal son la tasa de aprendizaje y el número de capas y neuronas.

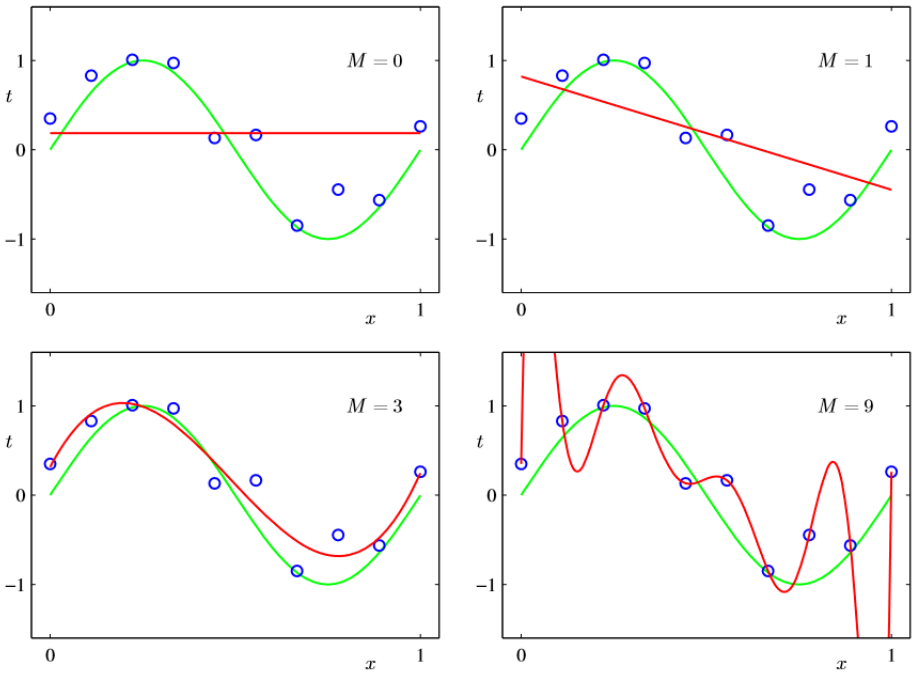

Figura 2.8: Ejemplos de subajuste y sobreajuste en regresiones polinómicas. Extraído de [27].

ción[4] suficiente para modelar los datos observados. Por el contrario, cuando se obtiene un error de entrenamiento bajo pero el error de prueba es alto se dice que hay un sobreajuste. Esto sucede porque el modelo tiene demasiada capacidad y modela el ruido en los datos de entrenamiento en vez de capturar el modelo subyacente. En la Figura 2.8 se observan estos fenómenos en un ejemplo simple de regresión polinómica. Al utilizar un polinomio de orden M bajo (casos M = 1 y 2) ocurre un subajuste de los datos, mientras que al utilizar un valor de M alto (caso M = 9) se puede observar un sobreajuste. El caso con M = 3 corresponde a una regresión polinómica aproximadamente óptima para los datos utilizados.

Existen diversas estrategias que permiten mitigar los problemas de sobreajuste:

- **Aumento de los datos de entrenamiento:** una forma de mejorar la generalización de un modelo es mediante el uso de más datos de entrenamiento. Sin embargo, estos usualmente son limitados por lo cual una estrategia es generar artificialmente nuevos datos. Por ejemplo, en el caso de audio, pueden generarse nuevas muestras mediante la aplicación de efectos como reverberación, distorsión, ecualización, etc. a los datos disponibles.
- **Detención temprana:** se guardan los pesos sinápticos de la red neuronal en cada épo-

[4]Informalmente, la capacidad de un modelo se vincula con la posibilidad de ajustar una amplia variedad de funciones. El modelo especifica la familia de funciones cuyos parámetros pueden ser variados en el proceso de aprendizaje. Esto se denomina capacidad de representación del modelo. Sin embargo, la capacidad efectiva del modelo queda determinada también por el algoritmo de optimización, ya que en la práctica no siempre es posible encontrar la función que mejor ajuste los datos dentro de la familia de funciones [26].

ca y se seleccionan aquellos que muestran un error de validación menor. El entrenamiento se detiene cuando el error de validación deja de disminuir.

- **Uso de parámetros compartidos:** permite reducir el número de parámetros, y por ende, la capacidad del modelo.
- **Dropout [28]:** durante el entrenamiento de la red neuronal se anula aleatoriamente la salida de un determinado porcentaje de las neuronas de una capa. Esto equivale a entrenar un ensamble de redes neuronales más pequeñas.

Otras estrategias importantes para mejorar el proceso de entrenamiento de una red neuronal artificial son:

- **Normalización por lotes [29]:** se normalizan las entradas de cada capa de la red substrayendo la media y dividiendo por la desviación estándar. Se introducen 2 parámetros a ser aprendidos: α y β, los cuales permiten revertir la normalización. Los beneficios de esta técnica han permitido el entrenamiento de redes cada vez más profundas y de manera más veloz. Además, su uso reduce el riesgo de sobreajuste.
- **Inicialización de los pesos**: existen diversas técnicas para una adecuada inicialización de los parámetros de la red [30, 31]. Estas son necesarias para asegurar el correcto aprendizaje de la misma.
- **Uso de ReLU**: el uso de funciones de activación que no presenten saturación (puntos donde su primera derivada tienda a cero) es importante en redes profundas para evitar efectos de desvanecimiento o explosión del gradiente (ver Anexo 1).

2.2.4 Redes neuronales convolucionales

Los patrones de conexión en las redes neuronales convolucionales (CNN) están inspirados en el funcionamiento de la corteza visual en mamíferos [32]. El primer antecedente de este tipo de arquitecturas fue el neocognitron [33] propuesto por Fukushima en 1980. En 1989, LeCun aplicó las primeras redes convolucionales al reconocimiento óptico de caracteres [34]. A partir de 2012 [35], las redes convolucionales comenzaron a incorporar un número cada vez mayor de capas y a revolucionar el campo del procesamiento de imágenes. El impacto de estas arquitecturas también ha alcanzado al mundo del audio [36, 37].

Las características que definen a una capa convolucional son:

- Sus neuronas, a diferencia de las presentes en una capa completamente conectada, sólo responden a una región de las entradas denominada campo receptivo. De esta manera, el número de pesos sinápticos se reduce.
- Los pesos sinápticos de las neuronas se comparten. Esto permite lograr invariancia a la traslación y reducir el número de parámetros a optimizar.

Este tipo de conexionado entre neuronas artificiales es equivalente a aplicar filtros de convolución[5] a un volumen de entrada, el cual comúnmente es una imagen. Las versiones filtradas del volumen de entrada, que son las salidas de la capa convolucional, se denominan mapas de características. Por ende, una red neuronal convolucional ajusta sus pesos sinápticos aprendiendo filtros de convolución. En la Figura 2.9 se muestran ejemplos de mapas de características obtenidos al aplicar distintos filtros de convolución 2D a los pixeles de una imagen de entrada. En una capa convolucional, los valores de las matrices corresponden a los pesos sinápticos.

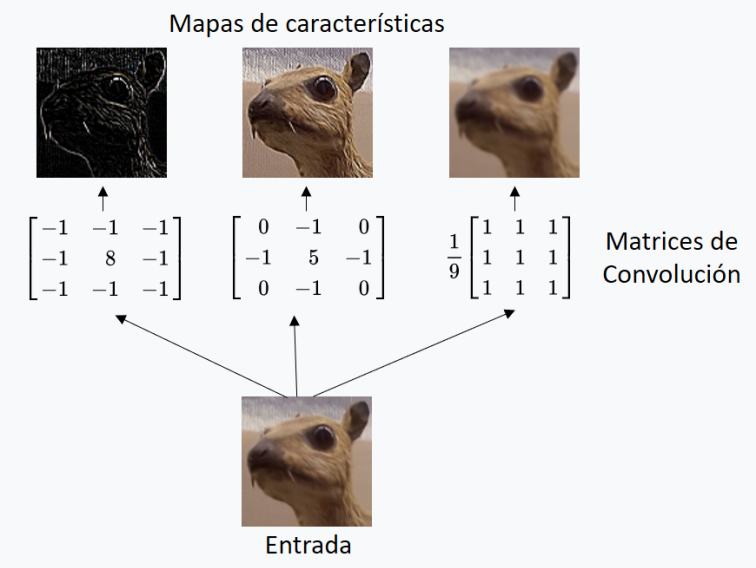

Figura 2.9: Mapas de características obtenidos mediante filtros de convolución[6].

Los elementos que definen a una capa convolucional y que se encuentran ilustrados en la Figura 2.10 son:

- **Tamaño de filtros (f_h,f_w):** define el campo receptivo de las neuronas. En la Figura 2.10 se utiliza un tamaño (3,3).
- **Número de filtros (f_n):** determina la cantidad de mapas de características a la salida de la capa convolucional.
- **Paso (s_h,s_w):** determina la distancia (en entradas) del campo receptivo de dos neuronas consecutivas. Permite reducir la dimensionalidad de las salidas. En la Figura 2.10 se utiliza un paso (1,1) en el diagrama de la izquierda y un paso (2,2) en el de la derecha.
- **Relleno de ceros:** se suelen agregar ceros en los bordes de la entrada para conservar la dimensionalidad en las salidas.

[5] Si bien en la literatura se emplea el término convolución, la operación matemática corresponde a una correlación cruzada.

[6] Adaptado de https://en.wikipedia.org/wiki/Kernel_(image_processing)

Los mapas de características resultantes poseen una dimensionalidad de:

$$\frac{W - F + 2P}{S} + 1 \tag{2.8}$$

donde W es el tamaño de la entrada, F el tamaño del filtro, P la cantidad de ceros de relleno, y S el paso.

Otro tipo de capas utilizadas en redes neuronales convolucionales son las de reducción de muestreo. Estas permiten reducir la dimensionalidad de las entradas transformando los elementos del campo receptivo en un solo valor global que suele ser el máximo (max pooling).

Para aumentar la dimensionalidad de la entrada, es decir el proceso inverso a las capas de reducción de muestreo, se utilizan convoluciones transpuestas. Una explicación detallada del funcionamiento de esta y otras capas se da en [39].

Existen numerosas arquitecturas de redes neuronales convolucionales que combinan de distintas maneras estas diferentes capas. Usualmente se intercalan capas convolucionales y capas de reducción de muestreo. A su vez, se suelen emplear capas completamente conectadas en la salida.

En este trabajo de tesis se utilizan arquitecturas de redes convolucionales consistentes de las siguientes etapas:

- **Codificador**: extrae las características representativas de la entrada reduciendo su dimensionalidad. Estas características forman un espacio latente, el cual es una representación comprimida (con baja dimensionalidad) de la entrada. El codificador se compone de una combinación de capas convolucionales colocadas en serie, generalmente con capas de reducción de muestreo intercaladas.
- **Decodificador**: su función es recuperar la dimensionalidad de la entrada a partir del espacio latente mediante el uso de convoluciones transpuestas u otras técnicas similares.

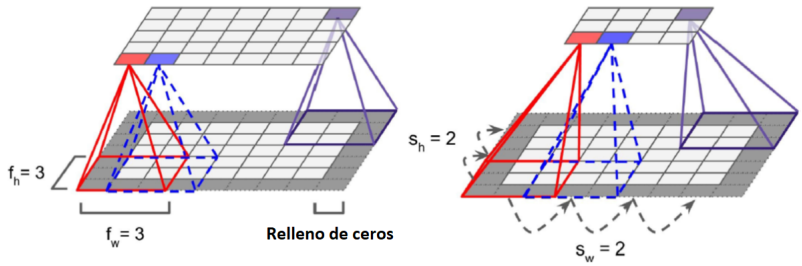

Figura 2.10: Diagrama de los distintos elementos de una capa convolucional. Extraído de [38].

2.3 SEPARACIÓN DE FUENTES SONORAS

2.3.1 Máscaras tiempo-frecuencia

Separar fuentes sonoras no estacionarias implica la realización de un filtro variante en el tiempo. Una manera de implementar este tipo de filtrado es mediante el uso de máscaras tiempo-frecuencia, las cuales se multiplican elemento a elemento con el espectrograma a filtrar. Posteriormente se invierte el espectrograma y recupera la señal en tiempo. Existen varios tipos de máscaras, siendo las más utilizadas:

- **Máscara binaria:** toma un valor de 1 si la señal a separar $S_i(t, f)$ es dominante, y 0 en caso contrario. Matemáticamente esto equivale a:

$$M_{b_i}(t, f) = \begin{cases} 1, & \text{si } S_i(t, f) \geq S_n(t, f) \; \forall \; n \neq i \\ 0, & \text{en otro caso} \end{cases} \tag{2.9}$$

- **Máscara suave:** toma valores entre 0 y 1 según cuánto domina una fuente por sobre las demás. Su expresión matemática es:

$$M_{s_i}(t, f) = \frac{S_i(t, f)}{\sum_k S_k(t, f)} \tag{2.10}$$

Las máscaras binarias permiten una mayor reducción de las interferencias en la señal separada, mientras que el uso de máscaras suaves provoca menor distorsión en la fuente separada [40, 41, 42].

Para calcular las máscaras se deben estimar las fuentes individuales en el dominio tiempo-frecuencia. Esto puede realizarse mediante una red neuronal que las prediga a partir de la señal de mezcla, o por otros métodos como los descritos a continuación.

2.3.2 Separación de imagen estereofónica

El análisis de escenas auditivas (ASA) es el proceso por el cual el ser humano organiza el sonido en elementos perceptuales significativos. Este proceso fue descrito por Albert Bregman [43], quien determinó que el ser humano analiza las mezclas sonoras mediante una etapa de segmentación y otra de agrupamiento. En la etapa de segmentación se descompone el estímulo sonoro en regiones tiempo-frecuencia, mientras que en la de agrupamiento se combinan los segmentos que es probable que hayan sido originados por una misma fuente en una estructura perceptual. Mediante experimentos psicoacústicos se determinaron algunas de las reglas que hacen posible el agrupamiento. Estas consisten en la proximidad en

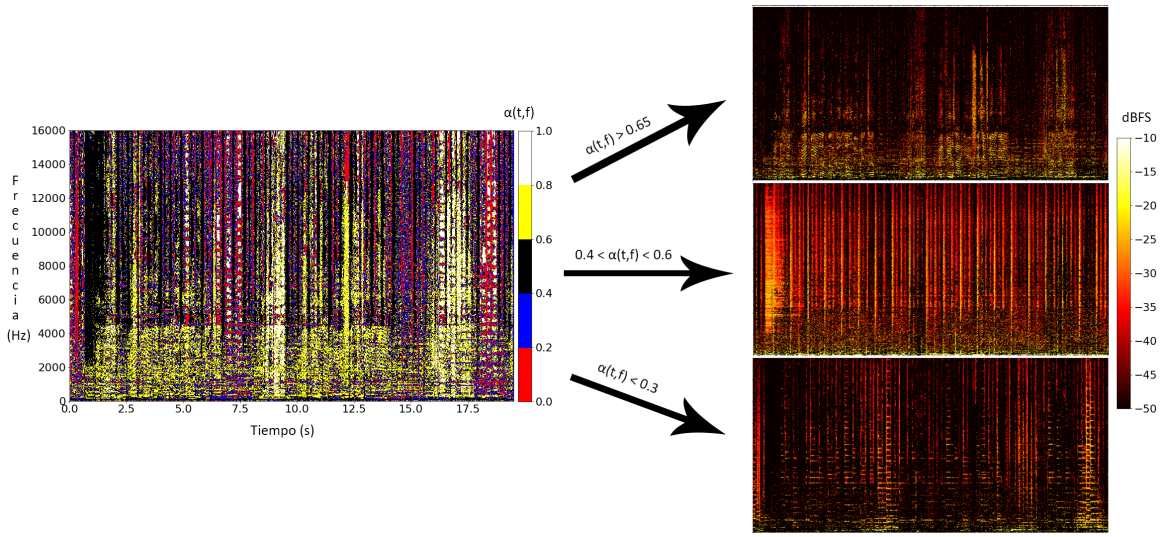

Figura 2.11: Ejemplo de separación de fuentes musicales mediante el uso de información panorámica.

tiempo y frecuencia, la armonía, las trayectorias continuas en tiempo, la información de comienzos (onsets) y finales (offsets), modulaciones comunes en amplitud y frecuencia, ritmo y localización espacial [43].

Las mezclas estereofónicas contienen información espacial sobre las fuentes sonoras facilitando la separación de fuentes. La ubicación de una fuente en una mezcla estereofónica queda principalmente determinada por el panorama. Diversas técnicas han sido desarrolladas para aprovechar la información panorámica en tareas de separación de fuentes [44, 45, 46].

Suponiendo una regla de panorama lineal, una fuente s_i es replicada en los canales izquierdo s_L y derecho s_R de acuerdo a las siguientes ecuaciones:

$$s_R = \alpha s_i \tag{2.11}$$

$$s_L = (1 - \alpha) s_i \tag{2.12}$$

Un valor de $\alpha = 0$ resulta en una ubicación de la fuente hacia la izquierda, un valor $\alpha = 0{,}5$ resulta en una ubicación central y un valor de $\alpha = 1$ produce una localización a la derecha. Si se toma la magnitud de la STFT de ambos canales, $|STFT_R|$ y $|STFT_L|$, y se dividen elemento a elemento, debido a la propiedad de linealidad de la DFT se tiene:

$$\frac{|STFT_R(t,f)|}{|STFT_L(t,f)|} = \frac{\alpha(t,f)}{1 - \alpha(t,f)} \tag{2.13}$$

Operando se puede obtener una representación del panorama en función del tiempo y la frecuencia:

$$\alpha(t,f) = \frac{\frac{|STFT_R(t,f)|}{|STFT_L(t,f)|}}{1 + \frac{|STFT_R(t,f)|}{|STFT_L(t,f)|}} \tag{2.14}$$

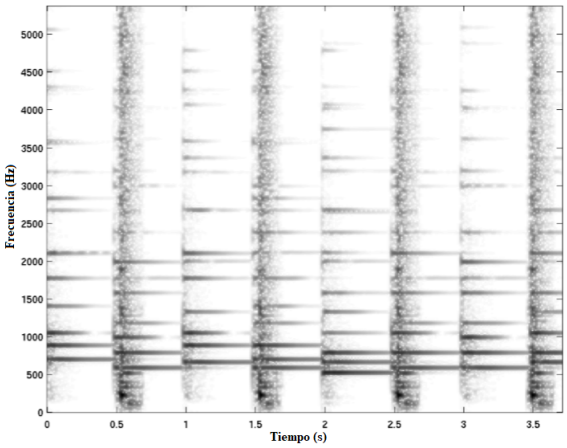

Figura 2.12: Espectrograma de una mezcla de componentes percusivas y armónicas. Extraído de [47].

El valor de $\alpha(t, f)$ se corresponderá con el panorama siempre y cuando no exista solapamiento en tiempo y frecuencia de las fuentes. Si una fuente sonora se encuentra ubicada en una región de la imagen estéreo en donde no hay otras fuentes sonoras, es posible separarla mediante el uso de máscaras tiempo-frecuencia que filtren el espectrograma en los valores donde $\alpha(t, f)$ corresponda al panorama de la fuente. En la Figura 2.11 se observa un ejemplo de separación de fuentes musicales mediante esta técnica, aplicada sobre la canción "Here Comes the Sun" de The Beatles. A partir de $\alpha(t, f)$, el cual se observa a la izquierda en el diagrama, se separan las fuentes ubicadas a la derecha ($\alpha(t, f) > 0{,}65$), las ubicadas al centro ($0{,}4 < \alpha(t, f) < 0{,}6$), y las ubicadas a la izquierda ($\alpha(t, f) < 0{,}3$) en la imagen estereofónica. De esta manera se obtienen los espectrogramas correspondientes a cada región de la imagen estereofónica, los cuales se observan a la derecha en el diagrama.

2.3.3 Separación de componentes percusivas y armónicas

La separación de una señal de audio en sus componentes armónicas y percusivas (HPSS) posee numerosas aplicaciones, principalmente como etapa de preprocesamiento en tareas de recuperación de información musical (MIR). Existen varios métodos para realizar la separación, entre los cuales se destaca el uso de filtros de mediana móvil [47]. Este algoritmo se basa en el hecho de que las componentes percusivas aparecen como líneas verticales en un espectrograma, mientras que las armónicas aparecen como líneas horizontales como puede apreciarse en el espectrograma de la Figura 2.12. El método es simple y de bajo costo computacional pudiendo resumirse en los siguientes pasos:

1. Se obtiene la magnitud de la STFT de la señal.
2. Se utiliza un filtro de mediana móvil a lo largo del eje de frecuencias. Esto produce un espectrograma con las componentes percusivas enfatizadas (S_P). Por otro lado, se

utiliza un filtro de mediana móvil en el eje del tiempo produciendo un espectrograma con las componentes armónicas enfatizadas (S_H).

3. Se calculan máscaras binarias o suaves a partir de comparar para cada punto tiempo-frecuencia si predominan las componentes percusivas (S_P) o las armónicas (S_H).
4. Se aplican las máscaras al espectrograma original y se invierten los espectrogramas resultantes.

2.3.4 Evaluación objetiva

La evaluación objetiva de los sistemas de separación de fuentes sonoras suele realizarse en base a la descomposición de la fuente estimada $\hat{S}_j$ en varias señales [48]:

- s_{target}: fuente verdadera S_j afectada por una distorsión permitida.
- e_{interf}: interferencias provenientes de otras fuentes.
- e_{noise}: ruido proveniente de los sensores usados para capturar la mezcla.
- e_{artif}: artefactos que no provienen de la señal original y son propios del algoritmo utilizado en la separación.

BSS_Eval [49] es un conjunto de herramientas de amplia difusión que permiten la evaluación objetiva mediante la descomposición de la fuente estimada. Las métricas obtenidas con BSS_Eval son la relación señal a distorsión (SDR), relación señal a interferencia (SIR) y relación señal a artefactos (SAR):

$$SDR = 10log_{10}\frac{||s_{target}||^2}{||e_{interf} + e_{noise} + e_{artif}||^2} \tag{2.15}$$

$$SIR = 10log_{10}\frac{||s_{target}||^2}{||e_{interf}||^2} \tag{2.16}$$

$$SAR = 10log_{10}\frac{||s_{target} + e_{interf} + e_{noise}||^2}{||e_{artif}||^2} \tag{2.17}$$

También, en el caso de señales con múltiples canales se puede calcular la relación imagen a distorsión espacial (ISR) propuesta en [50]:

$$ISR = 10log_{10}\frac{\sum_{i=1}^{I}(s_{ij}^{img})^2}{\sum_{i=1}^{I}(e_{ij}^{spatial})^2} \tag{2.18}$$

, donde s_{ij}^{img} es la fuente original i en el canal j, mientras que $e_{ij}^{spatial}$ es el error en preservar la imagen de la fuente i en el canal j. Las métricas obtenidas mediante el uso de BSS_Eval asignan un mismo peso a los distintos términos de error y no tienen en cuenta los aspectos perceptivos de la audición humana.

Como alternativa, se desarrolló el conjunto de herramientas Perceptual Evaluation Methods for Audio Source Separation (PEASS) [51], las cuales utilizan filtros gammatone [52] para descomponer la señal estimada en los términos de error. Además, en lugar de utilizar relaciones de energía, usan la medida de similaridad perceptiva (PSM) obtenida usando el modelo auditivo PEMO-Q [53]. Los descriptores obtenidos mediante PSM aplicado en los términos en los que se descompuso la señal son relacionados con resultados de pruebas subjetivas MUSHRA [54]. Mediante una red neuronal MLP se obtiene un mapeo de los descriptores ($q^{overall}$, q^{target}, q^{interf} y q^{artif}) a los puntajes de las pruebas subjetivas. De esta manera, el sistema devuelve los puntajes esperados si se realiza una evaluación MUSHRA sobre la señal. Estos puntajes son el puntaje perceptual total (OPS), el puntaje perceptual relacionado a la fuente objetivo (TPS), el puntaje perceptual relacionado a la interferencia (IPS) y el puntaje perceptual relacionado a los artefactos (APS).

Existe controversia en la literatura sobre cuál métrica es más adecuada para el análisis objetivo de la calidad de separación de fuentes musicales. En [55] se concluye que ni BSS_Eval ni PEASS poseen correlaciones significativas con resultados de pruebas subjetivas, mientras que en [56] se observa una mayor correlación de PEASS con las pruebas subjetivas. Estas discrepancias pueden deberse al tipo de algoritmo de separación utilizado en cada estudio, ya que en [55] se evalúan las métricas en tareas de HPSS, mientras que PEASS [57] fue diseñado mediante la correlación con pruebas subjetivas evaluando separaciones de habla e instrumentos musicales.

CAPÍTULO 3: ESTADO DEL ARTE

Los primeros trabajos enfocados en separar fuentes sonoras se basaron en emular los procesos realizados por los seres humanos para identificar y segregar eventos sonoros mediante ASA [58, 59]. Sin embargo, estos métodos sólo son aplicables a mezclas sencillas. Hacia el año 2000, las técnicas de factorización no negativa de matrices (NMF) [60] comenzaron a ser aplicadas exitosamente a la separación de fuentes musicales. NMF es una técnica que consiste en aproximar una matriz V como el producto de dos matrices: W y H:

$$\hat{V} = WH. \tag{3.1}$$

En la literatura, W suele recibir el nombre de diccionario y H de matriz de activaciones. La dimensión común a ambas matrices se denomina rango. Las matrices V, W y H deben poseer solamente elementos positivos y se minimiza el error entre la matriz original V y su aproximación $\hat{V}$.

En tareas de separación de fuentes sonoras suele aplicarse NMF sobre la magnitud del espectrograma de la mezcla, la cual es inherentemente no negativa. Debido a la restricción de no negatividad, las matrices W y H conservan un sentido físico. Esto puede observarse en la Figura 3.1, en la cual se muestra el resultado de aplicar NMF con rango igual a 2 sobre el espectrograma de una secuencia de dos notas, primero tocadas por separado y luego al unísono. La matriz W captura patrones espectrales en sus columnas, mientras que la matriz H captura la evolución de la amplitud en el tiempo de los patrones espectrales. Las columnas de la matriz W suelen denominarse átomos, mientras que las filas de la matriz H se denominan activaciones. En el ejemplo de la Figura 3.1, W contiene el espectro de cada nota y H la amplitud de las mismas a lo largo del tiempo. Por lo tanto, las técnicas de NMF aplicadas sobre espectrogramas realizan una descomposición espectral.

La combinación de NMF con el uso de funciones de costo que promuevan la generación de matrices de activación ralas [61, 62], la periodicidad de las activaciones [63] o una continuidad temporal de las mismas [64], favorece la separación de fuentes musicales. Asimismo, se han desarrollado extensiones de NMF a números complejos [65], lo cual permite modelar la fase y magnitud de la STFT en forma simultánea.

En los últimos años, las técnicas de aprendizaje profundo han permitido mejorar los sistemas previos de separación de fuentes basados comúnmente en NMF. Los modelos de aprendizaje profundo consisten de redes neuronales artificiales con un elevado número de capas. Cada capa utiliza la salida de la capa anterior como entrada. De esta manera, se forma una representación jerárquica en la cual las representaciones de los niveles superiores se

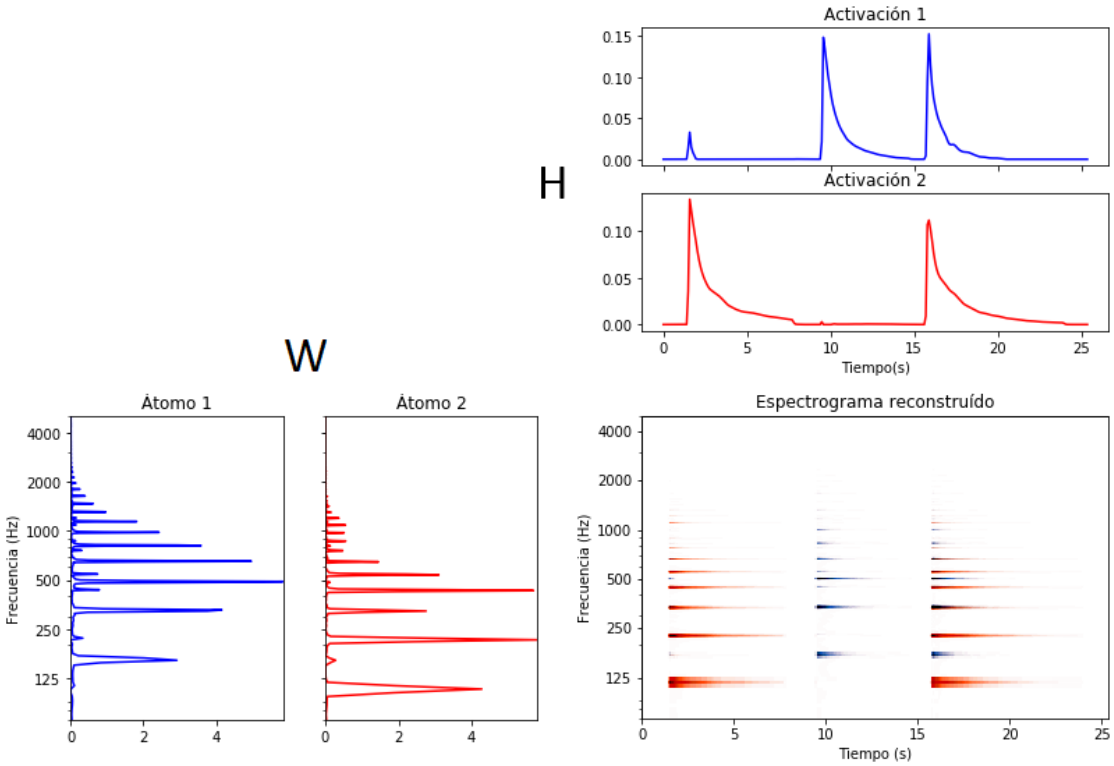

Figura 3.1: Descomposición mediante NMF de la magnitud de un espectrograma.

derivan de las de nivel inferior.

En el marco de la separación de fuentes, permiten modelar un proceso que prediga a partir de una señal de entrada la señal de salida deseada. En este caso, la señal de entrada será la mezcla de audio y la salida deseada las fuentes separadas. Tanto la entrada como la salida pueden ser muestras de audio, o una representación del mismo (espectrogramas, cocleagramas, escalogramas, etc...). Se han aplicado numerosas arquitecturas de red neuronal para separar fuentes, como por ejemplo, redes neuronales profundas (DNN); redes neuronales convolucionales (CNN); redes de memoria de corto y largo plazo (LSTM) y redes generativas adversarias (GAN).

En la Tabla 3.1 se muestran algunos de los trabajos de separación de fuentes musicales que han sido realizados en los últimos años basados en técnicas de aprendizaje profundo. Se indican las formas de representación de la señal y las arquitecturas de red neuronal profunda empleadas.

Tabla 3.1: Sinopsis de los trabajos de separación de fuentes musicales mediante técnicas de aprendizaje profundo.

Año	Referencia	Representación	Arquitectura	Comentarios
2013	Grais et al. [66]	$\|STFT\|$	DNN	Inicializada con NMF.
2015	Simpson et al.[2]	$\|STFT\|$	DNN	Aplicada a Karaoke
2015	Roux et al. [67]	$\|STFT\|$	NMF + DNN	
2016	Nugraha et al.[68]	$\|STFT\|$	DNN	
2017	Chandna et al.[69]	$\|STFT\|$	CNN	
2017	Jansson et al.[70]	$\|STFT\|$	CNN (U-NET)	
2017	Takahashi y Mitsufuji[71]	$\|STFT\|$	CNN (Densenet)	
2017	Uhlich et al.[72]	$\|STFT\|$	DNN + LSTM	Mejor desempeño en SISEC MUS 2016.
2017	Luo et al.[73]	$\|STFT\|$	Deep Clustering	
2017	Stoller et al.[74]	$\|STFT\|$	GAN	Semisupervisado
2018	Grais et al.[75]	Muestras	CNN	
2018	Park et al.[76]	$\|STFT\|$	CNN (Relojes de arena apilados)	
2018	Takahashi et al.[77]	$\|STFT\|$	CNN (Densenet) + LSTM	
2018	Stoller et al.[78]	Muestras	CNN (Wave-U-NET)	
2018	Uhlich et al.[79]	$\|STFT\|$ + Descriptores de Fase	DNN	Uso de retardo de grupo y frecuencia instantánea

CAPÍTULO 4: METODOLOGÍA

4.1 MANEJO DE DATOS

Para el entrenamiento de una red neuronal se necesita una gran cantidad de datos. En este estudio fueron obtenidos de la base de datos Demixing Secrets Dataset (DSD100) [80], que brinda mezclas estereofónicas de 100 canciones con sus respectivas pistas. Las categorías a las cuales pertenecen las pistas son: bajo, batería, voz y otros. La categoría otros puede contener una gran variedad de instrumentos musicales como guitarras, sintetizadores, instrumentos de viento, etc. Además, la base de datos abarca un gran número de estilos musicales, entre ellos: pop, rock, country, rap, reggae y metal. Sin embargo, como se observa en la Tabla 4.1, los géneros dominantes son el rock y pop.

La base de datos DSD100 se encuentra dividida en dos subconjuntos de 50 canciones, uno destinado al entrenamiento del modelo y el otro a la evaluación del mismo. En los modelos desarrollados se utilizó un 20 % de las canciones del conjunto de prueba como conjunto de validación y se entrenó sobre el conjunto de entrenamiento completo. Una vez finalizado el desarrollo del sistema, se evaluó el mismo sobre la totalidad del conjunto de prueba.

Tabla 4.1: Distribución de géneros musicales en DSD100.

Género	Prueba	Entrenamiento
Country	1	0
Electrónica	2	4
Metal	7	5
Jazz	1	1
Pop/Rock	36	35
Rap	3	3
Reggae	0	2

4.2 SISTEMA PROPUESTO

4.2.1 Funcionamiento general

Se desarrolló un sistema con una red neuronal convolucional que realiza una separación conjunta de las cuatro fuentes presentes en la base de datos [81]. La Figura 4.1 muestra el

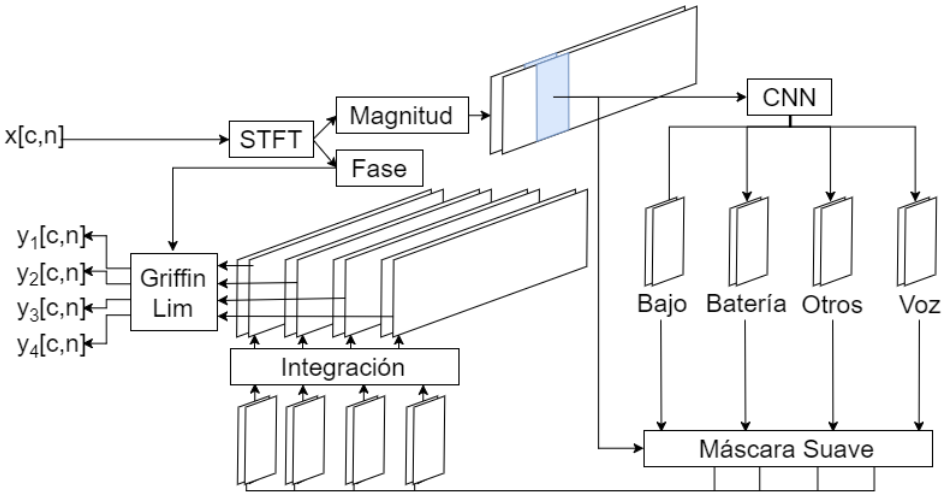

Figura 4.1: Diagrama en bloques del proceso de separación de fuentes musicales.

diagrama en bloques del sistema de separación de fuentes musicales desarrollado. Se parte del audio correspondiente a la mezcla $x[c, n]$, el cual tiene dimensiones $C \times N$, siendo C el número de canales y N el número de muestras. En este estudio, $C = 2$ ya que se utilizan audios de mezclas estereofónicas.

Se calcula la STFT de $x[c, n]$ obteniéndose una matriz de elementos complejos. Se empleó una ventana Hann de 2048 muestras y un tamaño de salto de 512 muestras, lo cual resulta en un factor de solapamiento del 75 %. Se obtienen la magnitud y fase de la STFT calculada. En el sistema propuesto, se realiza la separación sobre la magnitud de la STFT descartando la información de fase. La magnitud de la STFT $S[c, f, t]$ posee dimensiones $C \times F \times L$, donde F es el número de muestras en frecuencia, el cual queda determinado por la longitud de la ventana utilizada y vale 1025 (se descartan las frecuencias negativas), y L es el número de cuadros en el espectrograma, el cual queda determinado por el número de muestras del audio (N) y el tamaño de salto (H).

Como la red neuronal convolucional implementada posee entradas de tamaño fijo se debe segmentar el espectrograma en ventanas contextuales con un número de cuadros T, donde $T < L$. En este estudio se utilizan ventanas contextuales de 21 cuadros ($T = 21$), las cuales corresponden a 244 ms de audio para una frecuencia de muestreo de 44100 Hz. De esta manera, se logran entradas de tamaño fijo que son tensores en $\mathbb{R}^{2\times21\times1025}$. La ventana contextual permite a la red neuronal modelar dependencias temporales presentes en la señal que son importantes en audio.

La red neuronal convolucional (CNN) procesa las ventanas contextuales de entrada y devuelve como salida estimaciones de la magnitud de la STFT de las distintas fuentes a separar (bajo, batería, voz y otros). Cada una de las salidas posee la misma dimensionalidad que las ventanas contextuales de entrada. Para reducir el proceso a un filtrado en tiempo-frecuencia, se calculan máscaras suaves a partir de las fuentes estimadas por la red neuronal.

Estas máscaras son aplicadas a la entrada mediante una multiplicación elemento a elemento de los tensores, y como resultado se obtienen versiones filtradas del espectrograma de mezcla, que corresponden a los espectrogramas de las fuentes separadas.

Posteriormente es preciso integrar las distintas salidas de la red neuronal para recuperar espectrogramas con los L cuadros de la mezcla. En este estudio se exploran dos maneras de realizar esta integración de las salidas.

Por último, se tienen que recuperar señales de audio a partir de los espectrogramas estimados. Para esto se emplea el algoritmo de Griffin Lim utilizando como fase la correspondiente a la mezcla. En este estudio se asumirá que el efecto de distorsión de fase es irrelevante en comparación con los posibles errores al estimar la magnitud de la STFT.

4.2.2 Arquitectura de red neuronal

La arquitectura de red neuronal utilizada está inspirada en el trabajo de Chandna et al. [69]. Consiste en un codificador que reduce la dimensionalidad de la entrada seguido de múltiples decodificadores que recuperan las fuentes separadas. La arquitectura de red se ilustra en la Figura 4.2 y se describe a continuación.

El codificador consiste de 2 capas convolucionales en serie. La primera capa ($C1$) tiene N_V filtros de dimensión 1025×1. Estos filtros poseen una forma vertical en el espectrograma que abarca todas las frecuencias de un cuadro. De esta manera, se pretende que la red neuronal capture patrones espectrales presentes en el espectrograma de la mezcla. La segunda capa convolucional ($C2$) tiene N_H filtros de dimensión 1×6. Estos filtros poseen una forma horizontal para capturar la evolución temporal de los patrones espectrales aprendidos por la capa de filtros verticales $C1$. A diferencia del modelo propuesto en [69], se utilizan funciones de activación ReLU a la salida de las capas convolucionales y normalización por lotes (BN) en las entradas. El uso de las capas convolucionales reduce las dimensiones del espectrograma de entrada y tiene como objetivo capturar sus características más relevantes.

La salida de la capa convolucional de filtros horizontales $C2$ se introduce en una capa completamente conectada (FCL) de N_L neuronas, la cual en principio representa estas características relevantes formando un espacio latente. De esta manera, los 43050 elementos del tensor de entrada son reducidos a N_L elementos correspondientes a las activaciones de las neuronas del espacio latente.

La salida de la capa completamente conectada sirve de entrada a un conjunto de decodificadores en paralelo, cuya función es transformar el espacio latente y recuperar los espectrogramas de cada una de las fuentes. Se utilizan cuatro decodificadores correspondientes a las fuentes musicales a separar: bajo, batería, voz y otros. Cada decodificador consiste de una capa densa completamente conectada FC_i de N_L neuronas seguida de capas de convolu-

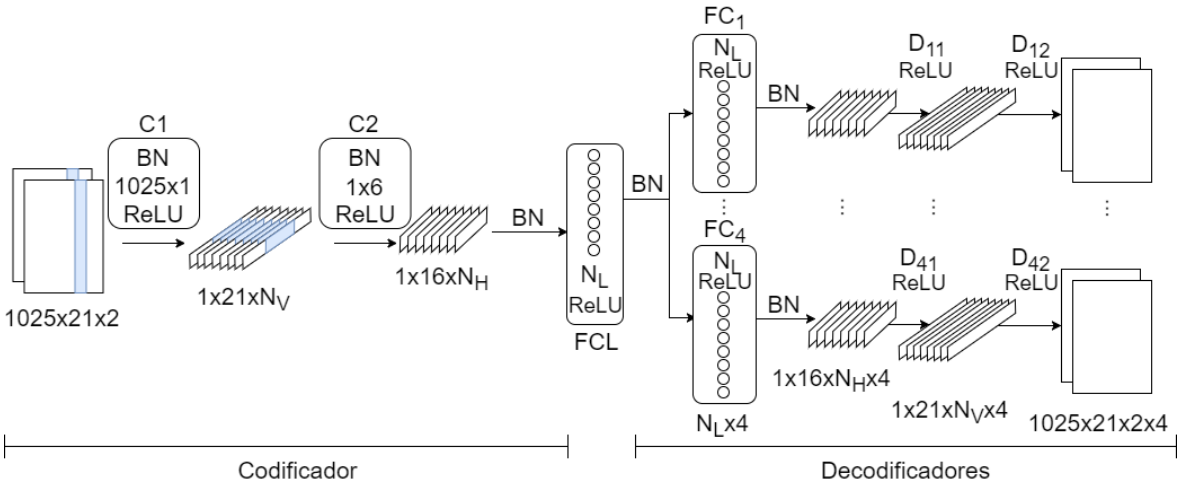

Figura 4.2: Modelo de red neuronal convolucional implementado.

ción transpuesta (D_{i1} y D_{i2}). La capa densa representa las características del espacio latente correspondientes a la fuente a separar, mientras que las capas de convolución transpuesta tienen la función de recuperar los espectrogramas de las fuentes estimadas a partir del espacio latente procesado. Se comparten los pesos de las capas de convolución transpuesta con el fin de reducir el número de parámetros, por lo que la separación ocurre esencialmente en las capas completamente conectadas y el proceso de recuperación de los espectrogramas se comparte entre las distintas fuentes. En las capas de convolución transpuesta de los decodificadores no se realiza normalización por lotes al comprobarse empíricamente que empeoraba la convergencia del error de entrenamiento.

4.2.3 Función de costo

La función de costo utilizada posee varios términos que representan distintas contribuciones al error total:

- **Error de estimación de las fuentes:** es el error cuadrático medio entre las salidas deseadas y las estimadas. Se excluye del cálculo la fuente "otros" debido a su dificultad de modelado.

$$L_{MSE} = \sum_{i=1}^{N-1} \overline{(\hat{S}_i - S_i)^2} \tag{4.1}$$

- **Interferencia entre fuentes:** es el error cuadrático medio entre las fuentes estimadas. Cuanto menos interferencias existan, mayor las diferencias entre distintas fuentes y, por ende, mayor este término.

$$L_{INT} = \sum_{i=1}^{N-1} \overline{(\hat{S}_i - \hat{S_{i\neq n}})^2} \tag{4.2}$$

- **Interferencia entre voces y otros:** es el error cuadrático medio entre las pistas estimadas de las voces y otros. Debido a que normalmente la voz comparte características espectrales con instrumentos como guitarras y sintetizadores, se utiliza un término que penaliza sus interferencias.

$$L_{OTH-VOC} = \overline{(\hat{S_N} - \hat{S_{VOZ}})^2} \tag{4.3}$$

- **Interferencia entre otros y demás fuentes:** es el error cuadrático medio entre las estimaciones de la fuente otros y las de las demás fuentes.

$$L_{OTH} = \sum_{i=1}^{N-1} \overline{(\hat{S_N} - \hat{S_i})^2} \tag{4.4}$$

- **Error de reconstrucción:** es el error cuadrático medio entre la suma de las fuentes estimadas y el espectrograma de la mezcla.

$$L_{REC} = \overline{(\sum_{i=1}^{N} \hat{S}_i - \sum_{i=1}^{N} S_i)^2} \tag{4.5}$$

Las expresiones se combinan en una función de costo final:

$$L = L_{MSE} - \alpha L_{INT} - \beta L_{OTH} - \gamma L_{OTH-VOC} + \delta L_{REC} \tag{4.6}$$

Los valores de α, β y γ son de 0.001, 0.01 y 0.03 respectivamente y fueron tomados de [69]. El valor de δ se estableció experimentalmente en 0.01.

4.2.4 Detalles de implementación

El sistema de separación de fuentes musicales se implementó en el lenguaje de programación Python mediante el uso de la biblioteca Keras[1], la cual permite un rápido prototipado de redes neuronales y funciona sobre la biblioteca TensorFlow[2].

Las distintas variantes del modelo se entrenaron durante 20 épocas, cada una consistente en el procesamiento completo en orden aleatorio de las 50 canciones provistas en el conjunto de entrenamiento de la base de datos DSD100. La optimización se realizó mediante el algoritmo de estimación adaptativa de momento (ADAM) [82], con una tasa de aprendizaje de 0.001 y recorte de gradiente [83] en 0.9. Los modelos fueron entrenados con un tamaño de lote de 32 muestras. Se utilizó la técnica de detención temprana consistente en detener el entrenamiento cuando el error de validación comienza a aumentar, y se seleccionaron los pesos sinápticos correspondientes a la época con menor error de validación. A su

[1] Sitio oficial de Keras: https://keras.io/

[2] Sitio oficial de TensorFlow: https://www.tensorflow.org/

Tabla 4.2: Tamaños de las redes neuronales evaluadas.

Modelo	N_V	N_H	N_L	Número de Parámetros
A	32	32	512	1456422
B	64	64	1024	5558854
C	128	128	2048	21701766
D	256	256	4096	85739782

vez, cuando el error de validación seguía disminuyendo tras las 20 épocas, el entrenamiento se extendió hasta que el error convergiera.

El entrenamiento se realizó en GPU utilizando una computadora con una tarjeta gráfica GTX1060 6GB y un procesador i7-4770k. Se entrenó utilizando la GPU debido a la aceleración que brinda respecto al uso de la CPU para tareas de aprendizaje profundo.

4.2.5 Ajuste de hiperparámetros

Se procedió a implementar la arquitectura de red neuronal propuesta utilizando distintos tamaños de red. Los cuatro modelos entrenados con sus respectivos valores de N_H, N_V y N_L se muestran en la Tabla 4.2. Una vez seleccionado el tamaño de red que produce el máximo SDR promedio entre las fuentes, se evaluó la influencia de modificar la escala de las entradas mediante la aplicación de las siguientes transformaciones:

$$Log_2(1 + |S_{IN}|) \tag{4.7}$$

$$Log_{10}(1 + |S_{IN}|) \tag{4.8}$$

Se experimentó con el uso de dropout en las capas FCL y FC_i utilizando dos estrategias: aplicación de 50 % de dropout en todas las capas completamente conectadas, y aplicación de 30 % de dropout en la capa FCL y 50 % de dropout en las capas FC_i.

4.2.6 Integración de las ventanas contextuales estimadas

Tanto las entradas como las salidas de la red neuronal convolucional consisten de T cuadros, sin embargo, la señal a procesar contiene L cuadros siendo $T < L$. En este estudio se evaluaron dos maneras de obtener las ventanas contextuales de entrada e integrar las salidas obtenidas con el fin de recuperar espectrogramas con L cuadros:

- Se desplaza la ventana contextual un cuadro por vez ($D = 1$) para obtener las entradas a la red neuronal. De las salidas se toma solo el cuadro central, correspondiente al onceavo, y los demás cuadros se descartan.

- Se desplaza la ventana contextual 3 cuadros por vez ($D = 3$) para obtener las entradas a la red neuronal. A partir de estas entradas, la red neuronal predice salidas que se solapan temporalmente entre si ya que D es menor a T. En esta estrategia no se descarta la información de ningún cuadro, sino que las salidas son solapadas y promediadas entre si. Finalmente, se recalculan las máscaras suaves para asegurar que los espectrogramas de salida sean versiones filtradas del espectrograma de entrada.

4.3 EXTENSIONES AL MODELO PROPUESTO

Una vez seleccionada experimentalmente la forma de transformar las entradas a la red y la regularización que producen el máximo SDR promedio en el conjunto de validación, se procedió a diseñar variantes de la arquitectura propuesta.

4.3.1 Incorporación de información panorámica

Se calculan los coeficientes de panorama mediante la aplicación de la ecuación (2.14) y se añaden como un tercer canal en la entrada a la red. De esta manera, la entrada pasa a ser un tensor en $\mathbb{R}^{3\times 21\times 1025}$ ($C = 3$). Se pretende que al incorporar información espacial la red neuronal pueda aprender patrones sobre la ubicación de las fuentes en la imagen estereofónica que ayuden a mejorar la separación.

4.3.2 Uso de HPSS en la entrada

Se utilizan filtros de mediana móvil y máscaras suaves para separar las componentes armónicas y percusivas de los espectrogramas de entrada. El tamaño de los filtros de mediana móvil empleados fue de 17 muestras. El filtrado se implementó mediante la función hpss del módulo decompose de la biblioteca LibROSA[3]. De este modo, para cada canal de la mezcla estereofónica se obtienen dos espectrogramas: el de componentes percusivas y el de componentes armónicas. Por lo tanto, se añaden 4 canales nuevos a los dos correspondientes al espectrograma de la mezcla, pasando a ser la entrada a la red un tensor en $\mathbb{R}^{6\times 21\times 1025}$ ($C = 6$).

4.3.3 Uso de múltiples ramas de filtrado: red convolucional dual

La arquitectura de red neuronal de la Figura 4.2 favorece el modelado de señales que presentan una forma de espectro fija cuya amplitud varía en el tiempo. Sin embargo, es co-

[3]Sitio oficial de LibROSA: https://librosa.github.io/librosa/

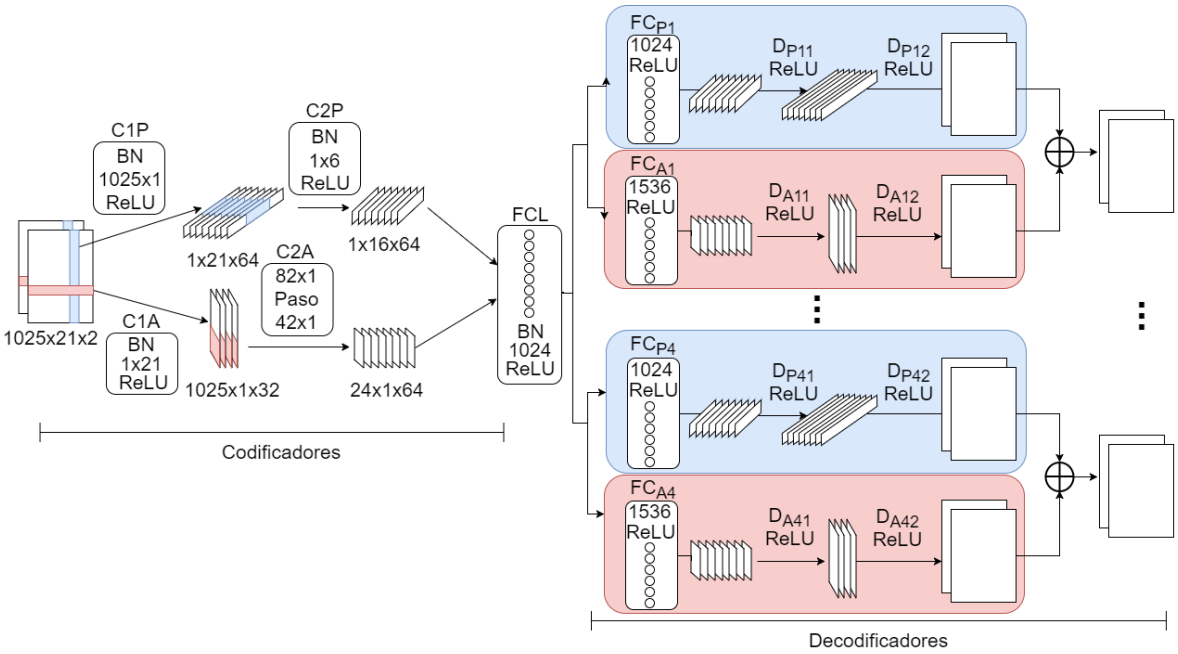

Figura 4.3: Modelo de red convolucional dual.

mún encontrar modulaciones en frecuencia, por ejemplo, en una voz realizando vibrato o un instrumento musical ejecutando un glissando[4]. Además, la forma vertical de los filtros convolucionales en la primera capa favorece la detección de patrones verticales en el espectrograma de entrada, los cuales se corresponden a señales percusivas.

Para modelar señales armónicas se añadió una rama en paralelo tanto en el codificador como en cada decodificador. Esta rama consiste de una capa convolucional de 32 filtros de tamaño (1,21) que capturan la evolución temporal de cada frecuencia. Se coloca otra capa convolucional en serie de 64 filtros de tamaño (82,1) y paso (41,1), los cuales integran la información en frecuencia y mediante el paso reducen la dimensionalidad. Posteriormente, las salidas de las dos ramas se concatenan y sirven de entrada a las neuronas de FCL.

La rama añadida en la etapa decodificadora es simétrica respecto a la de la etapa codificadora. Las salidas individuales de ambas ramas decodificadoras son sumadas formando una única salida sobre la que se realiza la optimización de la red neuronal. En la Figura 4.3 se observa esta variante, la cual se denominará como red convolucional dual en el resto del escrito. En [84] se presenta una arquitectura de red neuronal similar que usa satisfactoriamente ramas paralelas de bloques convolucionales para resolver tareas de visión por computadora. En [85] se utilizan ramas paralelas de filtros convolucionales con forma vertical y horizontal para clasificación de géneros musicales.

[4]Efecto sonoro consistente en pasar rápidamente de un sonido hasta otro más agudo o más grave haciendo que se escuchen todos los sonidos intermedios posibles dependiendo de las características del instrumento musical.

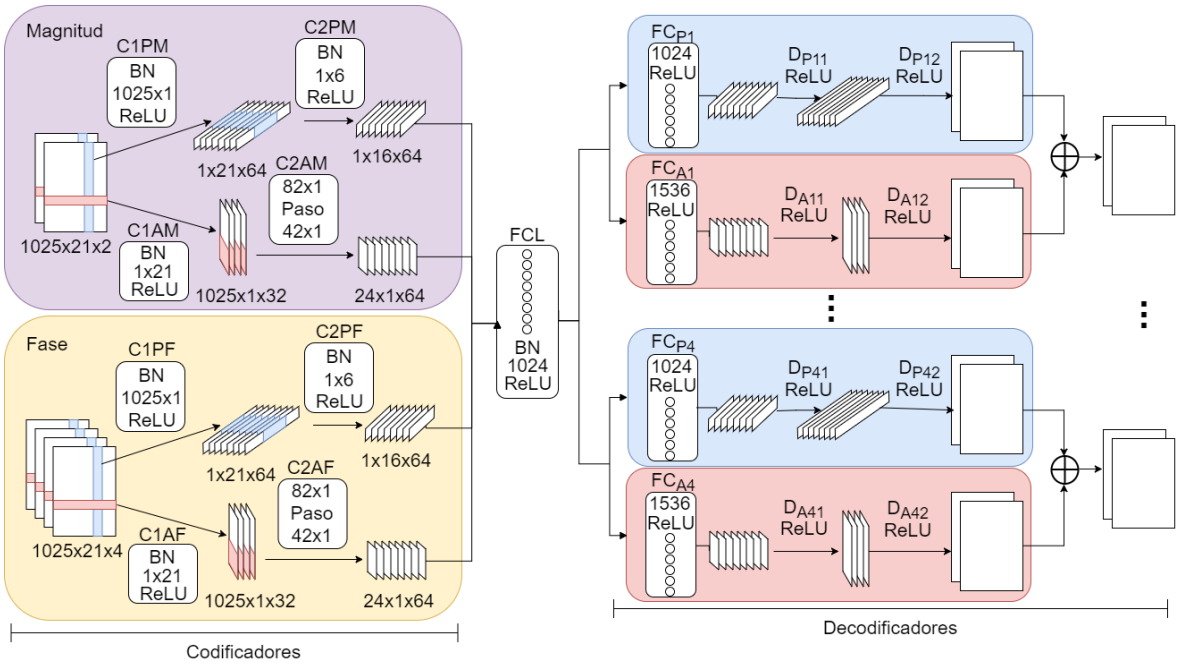

Figura 4.4: Modelo de red convolucional dual con descriptores de fase.

4.3.4 Incorporación de información de fase

Consiste en agregar información de fase a la red convolucional dual propuesta. Los descriptores de fase incorporados son la frecuencia instantánea y el retardo de grupo. Ambos se obtienen de la manera descrita en [79], la cual consiste en calcular diferencias entre cuadros sucesivos del espectrograma de fase y aplicar términos de corrección. Se determinó experimentalmente en [79] que resulta beneficioso procesar los descriptores de fase en una rama convolucional paralela y concatenar las salidas en el espacio latente, en vez de incorporar los descriptores de fase como canales adicionales en la entrada. La rama codificadora de la información de fase es idéntica a la que procesa la magnitud del espectrograma, y sus salidas se concatenan para alimentar a las neuronas de FCL. De esta manera, una rama de la etapa codificadora de la red neuronal se especializa en extraer propiedades relevantes de la magnitud del espectrograma, mientras que otra rama se dedica a procesar la información de fase. Esta arquitectura se ilustra en la Figura 4.4.

4.4 AUMENTO ARTIFICIAL DE DATOS

Debido a los pocos datos disponibles, se propone generar nuevas muestras de manera artificial para mejorar la generalización del modelo a mezclas no observadas. Esto se denomina aumento artificial de datos (DA) y se aplicó en el sistema final. Se utilizaron varias técnicas de DA:

- **Modificación de amplitud de las fuentes**: se multiplican las pistas de los instrumentos musicales por valores aleatorios entre 0.25 y 1.25 y se genera una nueva mezcla.
- **Mezclado de fuentes**: se combinan aleatoriamente instrumentos de distintas canciones.
- **Modificación de panorama de las fuentes**: se aplica panorama de manera aleatoria a cada instrumento generando una nueva mezcla con distinta ubicación espacial de las fuentes.
- **Intercambio de canales izquierdo y derecho**: se intercambian los canales con el fin de hacer al sistema más robusto ante el caso monofónico.
- **Conversión de estéreo a mono**: se convierte la mezcla estereofónica en una monofónica para lograr un sistema robusto ante la falta de información espacial.
- **Uso de reverberación artificial**: se aplica un reverberador artificial a cada uno de los instrumentos musicales. Se aleatorizan sus parámetros.
- **Cambio de tonalidad de la voz**: se realiza un pitch-shifting aleatorio entre -4 y 4 semitonos de la voz original.
- **Distorsión y realce de agudos en bajos**: se aplica distorsión a la pista de bajo y un filtro shelving de agudos. Se aleatoriza la frecuencia de corte entre 2000 y 5000 Hz, y la ganancia entre 10 y 20 dB. La finalidad es generar mezclas en las que el bajo posea mayor saliencia.

La implementación de estas técnicas de aumento artificial de datos se realizó mediante el uso de la biblioteca de Python pysndfx[5], la cual utiliza la aplicación SoX[6] que permite aplicar efectos y realizar conversiones de formatos de audio, entre otras funcionalidades.

El aumento de los datos se realiza al inicio de cada época de entrenamiento. Tras leer los datos originales se genera la misma cantidad de datos artificiales por lo que la cantidad de datos utilizados en cada época se duplica. La técnica que se utiliza para generar cada muestra aumentada es seleccionada de manera aleatoria. A su vez, al aplicar aumento de datos se incorpora el error cuadrático medio de la fuente otros en la función de costo con un peso de 0.5 respecto a las demás fuentes.

[5]Sitio de pysndfx: https://pypi.org/project/pysndfx/
[6]Sitio oficial de SoX: http://sox.sourceforge.net/

4.5 EVALUACIÓN OBJETIVA DE LOS SISTEMAS

Para llevar a cabo el ajuste de los hiperparámetros y la selección del sistema final, se realizaron evaluaciones objetivas mediante el uso del conjunto de herramientas BSS_EVAL 3.0[7]. Estas herramientas permiten calcular los parámetros o índices SDR, SIR, SAR e ISR de las 10 canciones del conjunto de validación para establecer posteriormente el sistema con mayor valor promedio de SDR.

Una vez seleccionado el sistema final, se evaluaron los mismos parámetros pero en el conjunto de prueba completo (50 canciones). BSS_EVAL 3.0 calcula las métricas en segmentos de 30 segundos, por lo que para obtener un valor final asociado a la canción se realizó un promedio de los resultados en los distintos segmentos. Esta forma de realizar la evaluación es la que se aplica en la mayor parte de los trabajos de separación de fuentes musicales. Por lo tanto, es posible una comparación directa de los resultados obtenidos con este sistema y los obtenidos en otros estudios.

A su vez, se evaluó el sistema final con el conjunto de herramientas PEASS[8], el cual permite calcular métricas correlacionadas con pruebas subjetivas de percepción sonora. Sin embargo, la mayor parte de los estudios sobre separación de fuentes musicales no reportan los resultados obtenidos mediante PEASS, por lo que no es posible comparar la calidad de la separación obtenida mediante estas métricas. Una serie de dificultades surgen al aplicar PEASS en las canciones de DSD100 debido a su duración, la cual provoca errores de falta de memoria computacional durante el cálculo. Para resolver este problema se dividió cada canción en 20 segmentos de igual duración y se calcularon las métricas. Finalmente, se promediaron los resultados obtenidos en cada segmento. Cabe destacar que el tiempo de procesamiento de PEASS es mucho mayor al de BSS_Eval.

[7] Estas herramientas se encuentran disponibles en: https://github.com/faroit/dsd100mat

[8] Estas herramientas se encuentran disponibles en: http://bass-db.gforge.inria.fr/peass/

CAPÍTULO 5: RESULTADOS

5.1 INFLUENCIA DEL TAMAÑO DE LA RED NEURONAL

En la Tabla 5.1 se muestran métricas obtenidas para cada uno de los modelos de la Tabla 4.2. Los valores presentados se calcularon sobre el total del conjunto de prueba realizando un promedio entre todas las canciones. En base al valor del índice SDR se escogió el tamaño correspondiente al modelo B (64,64,1024) de la Tabla 5.1, el cual posee el mayor SDR en todos los instrumentos salvo el bajo. Los resultados reportados a continuación corresponden a este tamaño de red neuronal.

Tabla 5.1: SDR medios para cada tamaño de red evaluado.

Modelo	Bajo	Batería	Otros	Voz	Promedio
A	2.16	3.66	2.36	2.79	2.74
B	2.28	**3.95**	**2.46**	**2.97**	**2.92**
C	2.26	3.87	2.30	2.84	2.82
D	**2.37**	3.91	2.35	2.87	2.88

5.2 APLICACIÓN DE TRANSFORMACIONES A LOS DATOS DE ENTRADA

Posteriormente, se evaluó el efecto de aplicar las transformaciones (4.7) y (4.8) a los datos de entrada. Los valores del SDR promedio se obtuvieron sobre el conjunto de validación conformado por las últimas 10 canciones del conjunto de prueba y se muestran en la Tabla 5.2. Se observa que el mayor SDR promedio se alcanza al aplicar la transformación logarítmica en base 2 a la magnitud del espectrograma (ecuación 4.7), aunque no transformar la magnitud del espectrograma da resultados similares.

Tabla 5.2: Resultados promedios de SDR en dB obtenidos al aplicar las transformaciones 4.7 - 4.8 a los datos de entrada.

	Bajo	Batería	Otros	Voz	Promedio
Abs	1.20	**5.37**	**2.55**	3.08	3.05
Log2	**1.31**	5.32	2.49	**3.13**	**3.06**
Log10	0.79	4.87	2.19	2.70	2.64

5.3 EFECTO DEL DROPOUT Y DE LA INTEGRACIÓN DE LAS VENTANAS CONTEXTUALES DE SALIDA.

Se evaluó el uso de la técnica de dropout, la cual ayuda a prevenir el sobreajuste del modelo. Los resultados se exponen en la Tabla 5.3. Se determinó que el uso de dropout no es beneficioso en la arquitectura propuesta (tamaño de red B con transformación logarítmica en base 2 de las entradas) ya que no mejora el SDR de ninguna de las fuentes. En consecuencia, no se aplicó dropout en el sistema final.

Tabla 5.3: Resultados promedios de SDR en dB obtenidos con distintas configuraciones de dropout.

	Bajo	Batería	Otros	Voz	Promedio
Sin Dropout	**1.31**	**5.32**	**2.49**	**3.13**	**3.06**
Dropout 50 % - 50 %	1.21	5.12	2.45	3.00	2.94
Dropout 30 % - 50 %	1.08	5.23	2.34	2.27	2.73

El efecto de la técnica utilizada para la integración de las ventanas contextuales de salida se muestra en la Tabla 5.4. Se observa que el promediado de las ventanas contextuales de salida permitió mejorar el SDR promedio. También se redujeron los tiempos de procesamiento, ya que al ser el desplazamiento igual a 3 cuadros, el número de ventanas contextuales procesadas por la red neuronal disminuye por un factor de 3.

Tabla 5.4: SDR medios en dB obtenidos al aplicar promediado temporal en las salidas.

	Bajo	Batería	Otros	Voz	Promedio
Sin Promediar	1.31	5.32	2.49	3.13	3.06
Promediado	**1.41**	**5.41**	**2.61**	**3.36**	**3.20**

5.4 EVALUACIÓN DE LAS EXTENSIONES AL MODELO PROPUESTO

En la Tabla 5.5 se muestran los resultados obtenidos para las distintas variantes del modelo B (Tabla 5.1). En todos los casos se aplicó la transformación (4.7) sobre los datos de entrada y se promediaron las ventanas contextuales de salida.

El uso de la red convolucional dual mejora considerablemente el desempeño del sistema para fuentes con contenido armónico, como la voz y la categoría otros. Esta arquitectura presenta el mayor SDR promedio entre los instrumentos musicales. La incorporación de descriptores de fase redujo el SDR promedio para todas las fuentes salvo la batería.

Tabla 5.5: SDR medios en dB obtenidos en distintas variantes del modelo.

Modelo	Bajo	Batería	Otros	Voz	Promedio
Modelo B con información panorámica	1.23	5.12	2.33	3.21	2.97
Modelo B con canales de HPSS	**1.54**	4.73	2.32	3.06	2.91
Red convolucional dual	1.39	5.28	**2.83**	**3.66**	**3.29**
Red convolucional dual + Fase	1.19	**5.29**	2.31	3.55	3.09

5.5 SISTEMA FINAL

El sistema final elegido consiste de la red convolucional dual de la Figura 4.3, la cual está conformada por dos ramas paralelas de filtros convolucionales, tanto en el codificador como el decodificador. Las entradas son transformadas mediante la ecuación (4.7) y las salidas se obtienen promediando ventanas contextuales desplazadas en tres cuadros (D = 3). No se aplicó dropout en ninguna capa.

La evaluación del sistema final se realizó sobre el conjunto de prueba completo de la base de datos DSD100. En la Tabla 5.6 se muestran los valores medios de SDR, calculados sin y con aumento artificial de datos. En la Tabla 5.6 también se muestran los valores correspondientes al sistema en el que se inspira este estudio [69] y a la primera versión del modelo (Figura 4.2), pudiendo observarse una mejora importante en la separación de todos los instrumentos.

La Tabla 5.7 presenta una comparación entre los resultados obtenidos con el sistema final aplicando DA y los de otros sistemas de separación recientes, utilizando la mediana del índice SDR en lugar de la media. La mediana es una medida robusta ante la presencia de valores atípicos [86], y por esta razón se utiliza frecuentemente para reportar resultados comparativos de separación de fuentes musicales utilizando la base de datos DSD100. Todos los sistemas comparados utilizan técnicas de aprendizaje profundo, salvo dNMF [87] que utiliza NMF. Los sistemas que muestran un mejor desempeño recurren a técnicas intensivas de aumento de datos (BLSTM [72], BLEND [72], MMDenseNet [71] y MMDenseLSTM [77]).

Tabla 5.6: SDR medios y sus desvíos estándar en dB evaluados sobre el conjunto de prueba.

Modelo	Bajo	Batería	Otros	Voz	Promedio
Chandna et al. [69]	0.90±2.70	2.40±2.00	0.80±1.50	1.30±2.40	1.35±0.73
Primer modelo	2.28±2.64	3.95±2.43	2.46±1.32	2.97±2.57	2.92±0.75
Sistema final sin DA	2.71±2.85	4.11±2.37	2.78±1.09	3.53±2.57	3.28±0.66
Sistema final con DA	**2.83±2.76**	**4.31±2.19**	**3.01±1.35**	**3.66±2.59**	**3.45±0.67**

Tabla 5.7: Mediana del SDR del sistema desarrollado comparada con otros sistemas de separación.

Modelo	Bajo	Batería	Otros	Voz	Promedio
Sistema Final con DA	2.91	3.95	3.26	4.21	3.58
DeepNMF [67]	1.88	2.11	2.64	2.75	2.35
NUG [68]	2.72	3.89	3.18	4.55	3.58
BLSTM [72]	2.89	4.00	3.24	4.86	3.75
BLEND [72]	2.98	4.13	3.52	5.23	3.97
MMDenseNet [71]	**3.91**	5.37	3.81	6.00	4.77
MMDenseLSTM [77]	3.73	**5.46**	**4.33**	**6.31**	**4.96**
SH-4stack [76]	1.77	4.11	2.36	5.16	3.35
dNMF [87]	0.91	1.87	2.43	2.56	1.94

En la Tabla 5.8 se muestran los resultados obtenidos empleando las métricas de PEASS. Estos valores no se pudieron comparar con los de otros trabajos ya que no suelen ser reportados. Los índices SDR, SIR, ISR y SAR difieren de los obtenidos mediante BSS_Eval debido a que los métodos de descomposición de la señal estimada son distintos. Las métricas OPS, TPS, IPS y APS pueden tomar valores en el rango de 0 a 100, siendo un valor más alto indicador de un mejor desempeño.

Tabla 5.8: Mediana de las métricas obtenidas mediante PEASS.

	Bajo	Batería	Otros	Voz
SDR	2.78	3.81	3.32	3.42
SIR	2.55	6.57	3.85	3.34
ISR	7.01	5.72	5.82	7.78
SAR	15.64	13.91	17.71	19.02
OPS	17.44	21.92	14.66	15.04
TPS	10.60	13.55	29.95	32.59
IPS	26.52	34.41	14.83	13.10
APS	19.58	19.96	46.20	42.70

5.6 TIEMPO DE PROCESAMIENTO

Se evaluó el tiempo de procesamiento del sistema final separando las pistas de un archivo de audio de un minuto de duración con frecuencia de muestreo de 44100 Hz. Se cronometró el tiempo completo que demora el sistema en obtener las fuentes separadas a partir del audio original. A su vez, se midió el tiempo de cálculo de la STFT, el tiempo de proce-

samiento de cada ventana contextual (predicción de la red neuronal y promediado de las salidas), y el tiempo de inversión de los espectrogramas de salida, incluyendo el guardado en disco duro de los audios separados. La evaluación se realizó en la computadora utilizada para el entrenamiento de la red neuronal (con GPU) y en una computadora portátil sin GPU que posee un procesador i5-7200U, repitiendo el procedimiento descrito 20 veces y promediando los tiempos obtenidos. Los resultados se muestran en la Tabla 5.9. Se observa que el tiempo de procesamiento en GPU (29 s) es menor a la duración de la mezcla.

Tabla 5.9: Tiempos de procesamiento del sistema implementado.

	GPU	CPU
Procesamiento completo	29 s	63.5 s
STFT	0.8 s	0.99 s
Procesamiento de cada ventana contextual	9.5 ms	26.3 ms
Inversión de la STFT	11.6 s	16.9 s

5.7 SEPARACIÓN DE MEZCLAS MONOFÓNICAS

Por último, se analizó la robustez del sistema final ante la ausencia de información estereofónica. Se transformaron las mezclas estereofónicas en monofónicas promediando los canales derecho e izquierdo. Posteriormente, se duplicó la mezcla monofónica resultante para generar los dos canales necesarios en la entrada a la red neuronal. Finalmente, los canales derecho e izquierdo en las salidas de la red neuronal fueron promediados para obtener señales monofónicas de las fuentes separadas. En la Tabla 5.10 se muestran las medianas del SDR, SIR y SAR obtenidas para cada fuente monofónica con BSS_Eval.

Tabla 5.10: Mediana del SDR, SIR y SAR del sistema desarrollado ante señales monofónicas.

	Bajo	Batería	Otros	Voz	Promedio
SDR	2.98	4.00	2.93	4.01	3.48
SIR	3.66	8.60	4.14	6.14	5.64
SAR	6.76	4.61	5.06	7.60	6.01

CAPÍTULO 6: DISCUSIÓN DE LOS RESULTADOS

6.1 EVALUACIÓN DEL SISTEMA PROPUESTO Y SELECCIÓN DEL SISTEMA FINAL

El modelo B de la Tabla 5.1, mediante el uso de activaciones ReLU y normalización por lotes, en combinación con un aumento en el número de párametros (debido a la mayor cantidad de filtros y neuronas), permitió obtener una mejora importante en la calidad de separación respecto al trabajo de Chandna [69] (ver Tabla 5.6).

Tanto en este estudio como en el de Chandna se da un mejor desempeño en la separación de la batería respecto a las demás fuentes musicales. Esto puede deberse a la forma de los filtros utilizados en la primera capa convolucional, los cuales abarcan todas las frecuencias de un cuadro propiciando el modelado de señales percusivas. En el caso de instrumentos musicales que producen notas, en principio sería preciso aprender un patrón espectral para cada una de las notas implicando el uso de muchos filtros. En [88] se reportan limitaciones parecidas de NMF para modelar modulaciones en frecuencia. Como se menciona en [69], la primera capa convolucional de la arquitectura de red neuronal de la Figura 4.2 es similar a NMF en cuanto a que solamente modela formas de espectro.

La aplicación de dropout para prevenir el sobreajuste no fue beneficiosa. Este resultado podría deberse a las siguientes razones:

- Ajuste inadecuado de los hiperparámetros: decidir en qué capa de la red neuronal aplicar dropout y la respectiva tasa no es una tarea trivial. Se requiere explorar un número suficiente de combinaciones lo cual es computacionalmente costoso.

- El modelo en el que se aplicó dropout (Figura 4.2) podría tener una capacidad de representación aproximadamente óptima o subóptima para los datos utilizados, por lo cual aplicar dropout disminuye su desempeño. Durante el entrenamiento de las distintas variantes de red neuronal no se observó en el monitoreo del error de validación que éste aumente, lo que sugiere que aplicar dropout no era necesario.

El promediado de las ventanas contextuales de salida en el modelo B (con transformación logarítmica en base 2 de las entradas) permitió mejorar el valor promedio de SDR en un 4.6 % y reducir el tiempo de procesamiento. Al solaparse las ventanas contextuales en el tiempo, un mismo cuadro de entrada es procesado múltiples veces en distintos contextos. De esta forma, se introduce redundancia en la información de entrada y para un mismo

cuadro la red neuronal genera varias salidas que difieren entre si. El promediado de varias predicciones parece contribuir a mejorar los resultados de separación, como se reporta en [72], donde las salidas que se promedian son obtenidas a partir del uso de múltiples redes neuronales con arquitecturas diferentes.

El modelo de red convolucional dual (Figura 4.3) con la transformación logarítmica en base 2 de las entradas, el promediado de las ventanas contextuales de salida y aumento artificial de datos, obtuvo el mayor SDR promedio, mejorando principalmente la calidad de separación de las fuentes voz y otros. Esto es razonable ya que la rama añadida de filtros convolucionales permite mejorar el modelado de componentes armónicos, los cuales están presentes frecuentemente en la voz y otros instrumentos. En cuanto al uso de descriptores de fase, su inclusión en el modelo de la Figura 4.4 no produjo una mejora en el SDR, a pesar de que en [79] se mostró que su incorporación era ventajosa.

6.2 SISTEMA FINAL

En la Tabla 5.6 se puede observar la mejora sucesiva del SDR en cada etapa de desarrollo del sistema de separación de fuentes musicales. El uso de aumento artificial de datos y la incorporación de la fuente otros en la función de costo permitió una mejora relativa promedio del 5.2 %. La categoría otros fue la más beneficiada por el aumento de datos, lo cual es razonable ya que es la fuente con mayor variabilidad. El aumento de datos es una práctica útil en sistemas de separación de fuentes, como quedó demostrado en [72] y [77], donde se aplicó aumento de datos de manera intensiva logrando mejorar el desempeño final del sistema. La importancia de disponer de una gran cantidad de datos se hace evidente particularmente en [77], donde se logran mejoras considerables al utilizar una base de datos propia de 800 canciones.

Como regla general se observa que los sistemas que utilizan aprendizaje profundo poseen un mayor SDR que aquellos basados en NMF [67, 87]. Los sistemas de separación de fuentes musicales suelen emplear una red neuronal por cada fuente a separar. En cambio, en este estudio, como en [69] y [76], se realiza la separación de las cuatro fuentes musicales empleando una sola red neuronal. Por un lado, esto permite que se utilice la información de todas las fuentes musicales en forma conjunta para la separación de cada fuente individual. Sin embargo, como se explica en [76], al asignar iguales pesos a cada fuente en la función de costo, la red neuronal tiende a separar mejor aquellas fuentes más distintivas y con mayor energía como la batería y la voz. Realizar una búsqueda de los pesos adecuados para cada fuente en la función de costo podría producir una mejora en los resultados finales, aunque la búsqueda de hiperparámetros sería computacionalmente costosa.

Los resultados obtenidos en este estudio son comparables a los de BLSTM y BLEND [72],

los cuales obtuvieron los mayores valores de SDR en la campaña de evaluación de separación de señales (SISEC) del año 2016 [80]. Los sistemas desarrollados más recientemente [77, 71] utilizan redes neuronales convolucionales con más capas y poseen conexiones saltadas[1], consiguiendo separaciones de mayor calidad. Sin embargo, cabe señalar que el tiempo de procesamiento del sistema propuesto es similar (e incluso menor al utilizar la GPU) a la duración de la mezcla a procesar, lo cual indica su potencial para realizar separaciones con una calidad comparable al estado del arte de manera veloz.

6.3 ANÁLISIS DE LAS MÉTRICAS OBJETIVAS OBTENIDAS

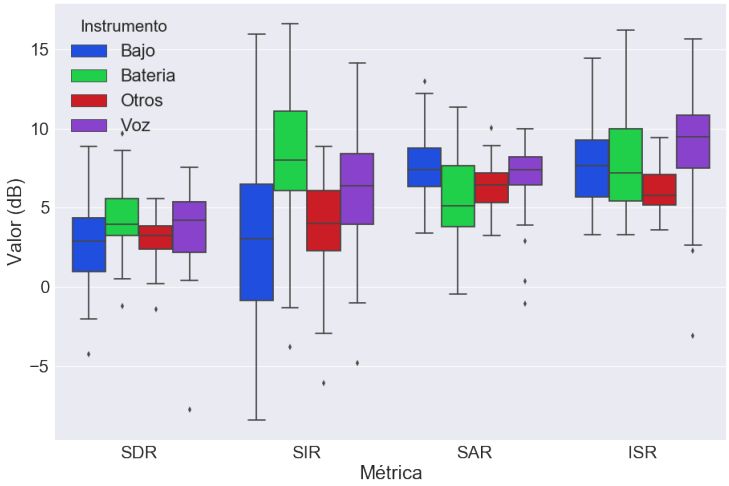

Figura 6.1: Diagrama de caja y bigotes de los resultados obtenidos mediante BSS_Eval en el sistema final.

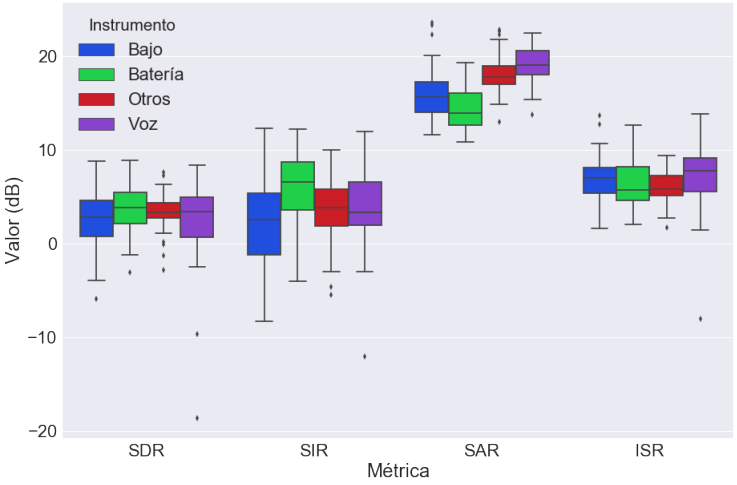

Figura 6.2: Diagrama de caja y bigotes del SDR, SIR, SAR e ISR obtenidos mediante PEASS en el sistema final.

[1]Las conexiones saltadas (skip connections) son conexiones entre una capa de la red y otra que no es la siguiente. Algunas arquitecturas de red neuronal que utilizan este tipo de conexiones son ResNet [89] y DenseNet [90].

Las figuras 6.1 y 6.2 muestran el diagrama de cajas y bigotes de los índices SDR, SIR, SAR e ISR obtenidos mediante BSS_Eval y PEASS. En la Figura 6.1 se observa la presencia de valores atípicos, especialmente en la voz, los cuales disminuyen el valor medio de SDR. Para el reporte de resultados los valores atípicos no fueron eliminados. En cuanto a los valores de SDR, se observa que el bajo es la fuente con mayor dispersión, mientras que la categoría otros es la que presenta menor variabilidad. Comparando las métricas, la que posee un mayor rango (diferencia entre el máximo y el mínimo) es el SIR. Esto podría deberse a que los niveles relativos, y por ende las interferencias entre instrumentos, varían de manera apreciable en las distintas canciones. Por ejemplo, cuando ocurre un golpe de batería, la energía del mismo es alta respecto a los demás instrumentos haciendo que el cociente $\frac{s_{target}}{e_{interf}}$ en la ecuación (2.16) tienda a infinito y el SIR sea elevado aún si el sistema de separación no logra reducir las interferencias.

Debido a que PEASS realiza una descomposición distinta de la señal, los índices SDR, SIR, SAR e ISR difieren respecto a BSS_Eval, como se observa en la Figura 6.2. Se puede ver que el SAR es mayor al utilizar PEASS como método de evaluación, indicando que la energía atribuida a los artefactos es menor que al emplear BSS_Eval. En [56], al comparar los métodos de evaluación objetiva PEASS y BSS_Eval, también se observa la tendencia a obtener valores mayores de SAR cuando se utiliza PEASS. Los valores de SDR son similares a los obtenidos con BSS_Eval salvo para el caso de la voz, en el que disminuye un 23 %.

En la Figura 6.3 se muestra el diagrama de cajas y bigotes de los índices perceptuales OPS, TPS, IPS y APS calculados mediante PEASS para las distintas fuentes musicales. Los índices OPS, que se relacionan con la calidad general del sistema de separación, indican que la batería es la fuente con mejores resultados de separación y la categoría otros la de menor calidad general de separación. El IPS es elevado en la batería indicando que al evaluar su separación con una prueba subjetiva, las interferencias de otras fuentes no serían percibidas

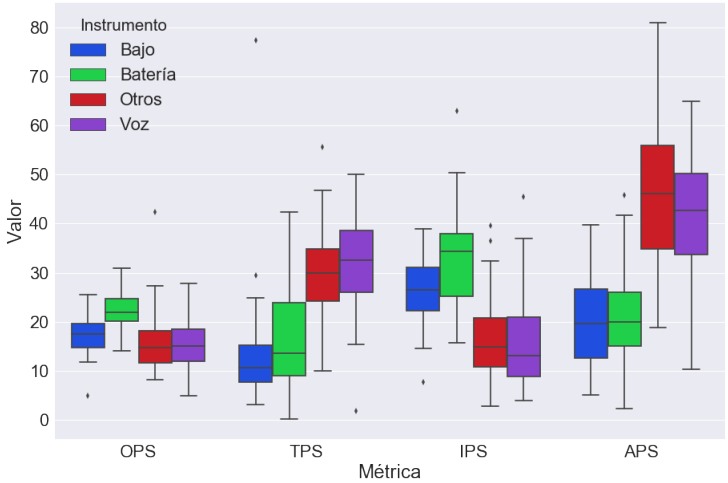

Figura 6.3: Diagrama de caja y bigotes del OPS, TPS, IPS y APS obtenidos mediante PEASS en el sistema final.

tanto como, por ejemplo, en el caso de la voz. Sin embargo, el índice TPS, que se asocia con la calidad en términos de preservación de la fuente a separar, es inferior en la batería y elevado en la voz. Los valores de APS sugieren que las fuentes voz y otros son las que presentan artefactos más notorios.

6.4 ANÁLISIS CUALITATIVO DEL SISTEMA

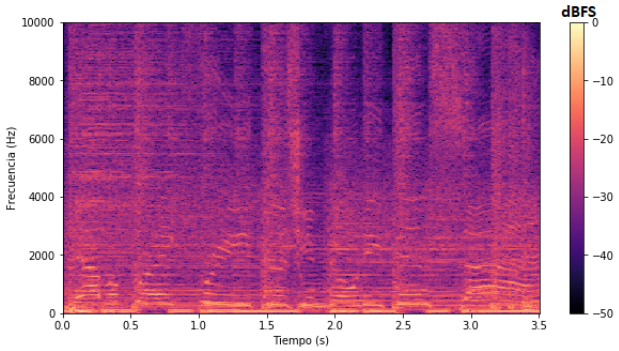

Figura 6.4: Espectrograma de un fragmento de la canción “Too Much - BKS”.

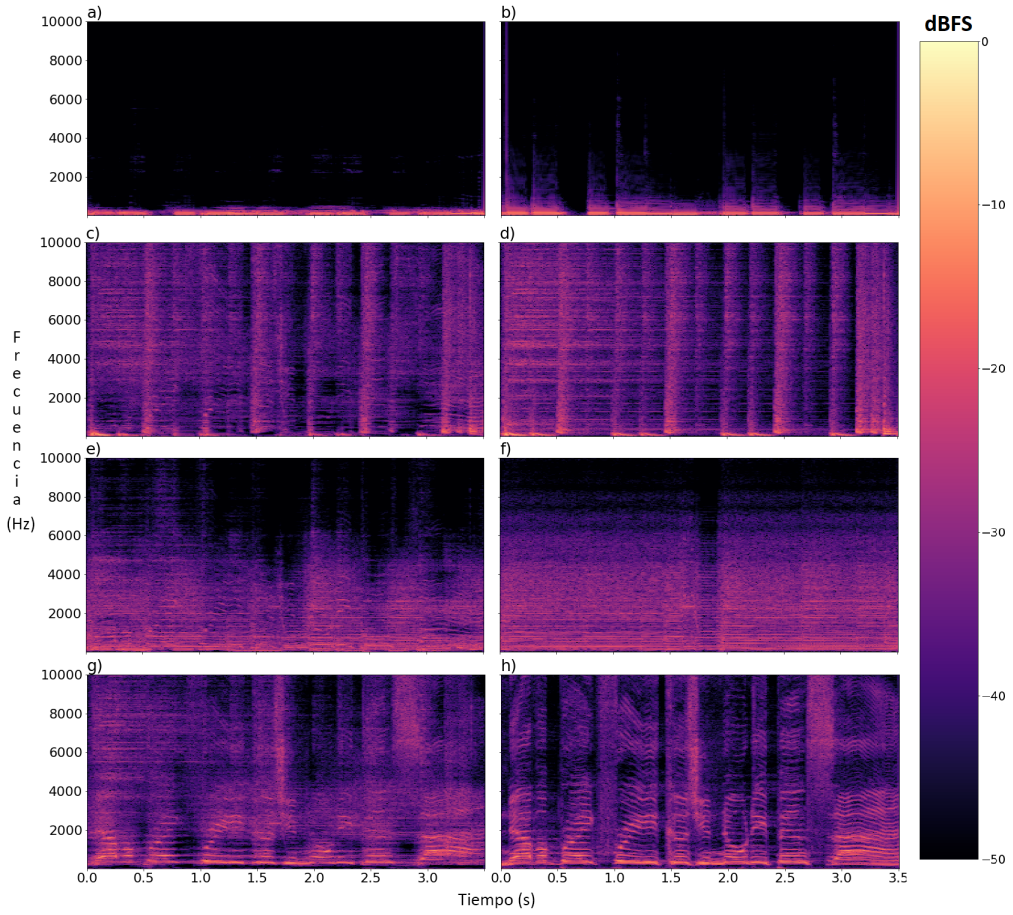

Figura 6.5: Ejemplo de separación de fuentes musicales mediante el sistema realizado.

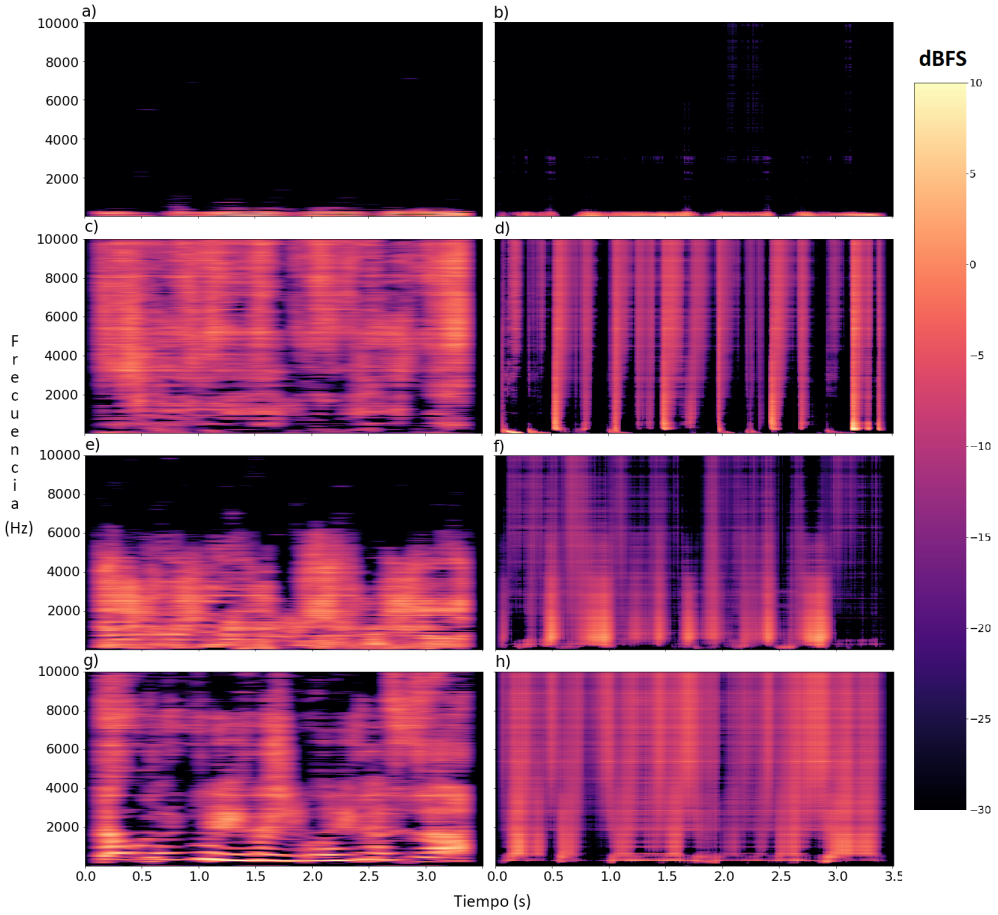

Figura 6.6: Salidas de los decodificadores de la red neuronal implementada.

En la Figura 6.4 se muestra el espectrograma de un fragmento de la canción “Too Much” de BKS[2], la cual se encuentra en el conjunto de prueba de la base de datos DSD100. En la Figura 6.5 se muestran los resultados obtenidos al utilizar el sistema desarrollado para separar la mezcla de la Figura 6.4. En la columna izquierda se observan los espectrogramas de las fuentes separadas y en la derecha los de las verdaderas correspondiendo a) y b) al bajo, c) y d) a la batería, e) y f) a otros, y g) y h) a la voz.

Al separar el bajo, se ve que si bien se mantienen las componentes de menor frecuencia correspondientes a la fundamental y los primeros armónicos, el ataque de las notas, al igual que un gran número de armónicos superiores, se pierden. Esto es razonable ya que el bajo es normalmente la señal con menor nivel dentro de una mezcla. También, sus componentes armónicas suelen quedar enmascaradas por la frecuencia fundamental de la voz y los golpes de batería. Esta dificultad para separar el bajo se ve reflejada en la Tabla 5.7, al ser la fuente que exhibe un menor SDR en prácticamente todos los sistemas de separación.

El espectrograma de la batería separada preserva las líneas verticales, correspondientes a los golpes. Sin embargo, también se pueden percibir curvas horizontales generadas prin-

[2]Disponible en http://www.cambridge-mt.com/ms-mtk.htm#BKS_TooMuch

cipalmente por interferencias de la voz. En el caso de la categoría otros, se nota la pérdida de una parte del contenido en alta frecuencia y la interferencia de la voz. El espectrograma correspondiente a la voz estimada muestra que se alcanza una buena separación de la frecuencia fundamental y de los primeros armónicos, aunque los armónicos de mayor orden no son tan definidos. Esto puede deberse a la poca cantidad de capas presentes en la rama especializada en capturar componentes armónicos. Por otra parte, se puede observar que se conservan de manera aproximada las formantes de la voz.

En la Figura 6.6 se muestran las salidas de los decodificadores de la red convolucional dual utilizada en el sistema final (Figura 4.3). Las subfiguras a) y b) corresponden al bajo, c) y d) a la batería, e) y f) a otros, y g) y h) a la voz. En la columna izquierda se ven las salidas de las capas D_{Ai2}, cuya finalidad es reconstruir los patrones armónicos del espectrograma. Se puede distinguir, principalmente en la voz, la presencia de líneas horizontales relacionadas con la frecuencia fundamental y los componentes armónicos. En la columna derecha se observan las salidas de las capas D_{Pi2}, cuya finalidad es reconstruir los patrones percusivos del espectrograma. Se ve, principalmente en la batería, cómo las salidas revelan líneas verticales correspondientes a eventos percusivos. Mediante la inspección de las salidas de los decodificadores se puede comprobar que la rama añadida en la red convolucional dual efectivamente reconstruye componentes armónicos, mientras que la otra rama genera los componentes percusivos.

Algunas debilidades del sistema que han sido detectadas en pruebas de escucha son:

- Confusión de solos de guitarra con voz cantada.
- Frecuencias medias y altas del bajo asignadas a la categoría otros. Esto es más notorio cuando el bajo ejecuta notas en el rango superior del registro o realiza slap.
- Consonantes de la voz cantada interfiriendo con la batería.
- Parte percusiva de las fuentes, como ataques y rasguidos con púa, interfiriendo en la batería.

Por último, si bien la separación de fuentes en el caso monofónico es considerada de mayor dificultad que en el caso estereofónico, los resultados de la Tabla 5.10 muestran que el sistema desarrollado no presenta una disminución marcada de la calidad de separación. Esto indicaría que los patrones aprendidos por la red neuronal convolucional no dependen fuertemente de la información estereofónica. Algunos ejemplos de separación de fuentes realizados sobre canciones del conjunto de prueba se encuentran disponibles en el sitio web de la tesis[3].

[3]Sitio web de la tesis: https://mrpep.github.io/Fast-Music-Source-Separation/

CAPÍTULO 7: CONCLUSIONES

En este estudio se ha diseñado y mejorado en etapas sucesivas un sistema de separación de fuentes musicales. El sistema se basó en trabajos previos con redes neuronales convolucionales aplicadas a la separación de fuentes pero presenta una serie de innovaciones. Estas son, principalmente, la incorporación de funciones de activación ReLU a la salida de las capas convolucionales, el uso compartido de pesos en las etapas decodificadoras, la incorporación de ramas paralelas de filtrado enfocadas al procesamiento de señales armónicas y percusivas, y el promediado de las ventanas contextuales de salida.

El sistema desarrollado utiliza una sola red neuronal para separar las cuatro fuentes y posee relativamente pocas capas convolucionales. De esta manera, se logra una separación veloz de las fuentes musicales que podría aprovecharse en un sistema con requerimientos de baja latencia.

Se obtuvieron resultados objetivos comparables al estado del arte actual. La evaluación objetiva fue realizada mediante el conjunto de herramientas BSS_Eval y también empleando las herramientas PEASS, las cuales proporcionan índices perceptuales. Si bien el sistema fue diseñado para operar sobre señales estereofónicas, la calidad de separación es similar en el caso monofónico.

Como parte del trabajo de investigación se desarrolló código de computadora con el que es posible separar las pistas de bajo, batería, voz y otros instrumentos a partir de una mezcla. El código es de acceso libre y gratuito esperando que sea de utilidad para investigaciones futuras.

Los sistemas de separación de fuentes musicales actuales aún no alcanzan resultados de calidad aplicables a las ciencias de la grabación. Sin embargo, se espera que en los próximos años mejore la calidad de las separaciones posibilitando su uso en restauración y remasterización de grabaciones antiguas y en la obtención de pistas de acompañamiento para músicos, entre otros.

CAPÍTULO 8: LÍNEAS FUTURAS DE INVESTIGACIÓN

El uso de la STFT como forma de representación del audio, si bien simplifica las arquitecturas de red neuronal empleadas para realizar separaciones, conlleva una serie de problemáticas. El uso de una ventana fija no resulta adecuado para todos los tipos de señales presentes en la música. A su vez, descartar la información de fase puede constituir una pérdida de información relevante. Es por esto que el sistema desarrollado podría mejorar si se utilizan otras técnicas para representar el audio, tales como Wavelets, transformada Q Constante, uso de ventanas adaptativas, entre otras. Otra posible mejora es considerar la percepción humana del sonido, por lo cual encontrar maneras de representar el audio similares a como lo hace una persona podría conllevar mejoras en la calidad percibida de separación. Recientemente, la arquitectura de red Wavenet [36] ha permitido trabajar directamente con las muestras de audio. En lugar de utilizar la STFT como forma de representar el audio a la entrada de la red neuronal convolucional dual desarrollada, se podría emplear una red neuronal Wavenet que obtenga una representación más adecuada de la señal a partir de sus muestras.

Existe una tendencia generalizada a utilizar redes neuronales convolucionales con un número cada vez mayor de capas y conexiones, lo cual en líneas generales contribuye a mejorar el desempeño de los sistemas. Una posible mejora a la arquitectura de red convolucional dual desarrollada consistiría en añadir más capas en la rama armónica. Por otra parte, las capas convolucionales le dan un mismo peso a todas las regiones del espectrograma. Si bien en procesamiento de imágenes la propiedad de invariancia a la traslación de los filtros convolucionales es beneficiosa y tiene sentido, en audio esto no es tan así, ya que un patrón en baja frecuencia no es equivalente al mismo patrón pero en alta frecuencia. En consecuencia, el sistema desarrollado también podría beneficiarse dividiendo el espectrograma de entrada en varias regiones, con el fin de aplicar un procesamiento distinto a las frecuencias bajas, medias y altas. Otra posible mejora podría ser utilizar capas localmente conectadas en lugar de filtros convolucionales convencionales.

Si bien en esta investigación se utilizó una sola red neuronal para separar las cuatro fuentes, el uso de una red neuronal para cada fuente a separar podría ser conveniente. Emplear múltiples redes neuronales permitiría utilizar arquitecturas y formas de representación de la señal distintas para cada instrumento musical. Por ejemplo, la red especializada en separar batería podría incorporar formas de filtro similares a las utilizadas en la rama especializada en patrones percusivos y una ventana de análisis de la STFT corta para una localización precisa en el tiempo. A su vez, se podría añadir otro sistema que tome como entrada las fuentes

estimadas por la red convolucional dual y mejore su calidad reconstruyendo información perdida y refinando las estimaciones.

El uso de técnicas de aprendizaje de máquina requiere una gran cantidad de datos para obtener una buena calidad de separación. Futuros trabajos podrían enfocarse en crear bases de datos más amplias y de géneros musicales específicos, así como sistemas de separación que utilicen técnicas de aprendizaje no supervisado y no requieran de datos organizados en pistas. Además, en el marco de la separación de fuentes musicales, es conveniente disponer de herramientas que permitan una adecuada evaluación objetiva del desempeño de los algoritmos de separación de fuentes musicales. Esto es importante para conocer qué estrategias producen los mejores resultados y realizar una correcta toma de decisiones en la etapa de diseño del sistema.

ANEXO I: DESCENSO POR GRADIENTE Y PROPAGACIÓN HACIA ATRÁS DEL ERROR

I.1 OPTIMIZACIÓN MEDIANTE DESCENSO POR GRADIENTE

En el proceso de aprendizaje de una red neuronal se busca minimizar una función de costo C. Esta es función de los parámetros de la red neuronal (pesos y umbrales). Se deberán encontrar los valores de los pesos y umbrales tal que hagan mínima la función de costo $C(W, B)$. Realizar la optimización de manera analítica requiere la inversión de matrices de dimensiones muy altas, por lo cual se vuelve un problema intratable. Es por esto que se recurre a métodos numéricos, siendo el descenso por gradiente el más utilizado.

Para encontrar los valores de los vectores de pesos W y umbrales B que minimicen $C(W, B)$ debemos conocer cómo varía C al variar los pesos w_i y umbrales b_i. Para simplificar la notación, los vectores W y B se agrupan en un solo vector P, el cual contiene todos los parámetros de la red neuronal. La variación de C ante cambios en P se expresa:

$$\Delta C = \frac{\partial C}{\partial p_1}\Delta p_1 + \frac{\partial C}{\partial p_2}\Delta p_2 + ... + \frac{\partial C}{\partial p_N}\Delta p_N \tag{I.1}$$

Lo cual se puede reexpresar de forma vectorial como:

$$\Delta C = \nabla C \Delta P \tag{I.2}$$

, en donde ∇C es el gradiente de C respecto a P.

Para minimizar C, ΔC deberá ser negativo. Si tomamos:

$$\Delta P = -\eta \nabla C \tag{I.3}$$

, en donde η es la tasa de aprendizaje, ΔC será siempre menor o igual a 0:

$$\Delta C = \nabla C - \eta \nabla C = -\eta \|\nabla C\|^2 \tag{I.4}$$

La tasa de aprendizaje η es un hiperparámetro importante en la optimización. Un valor muy pequeño supondrá una gran cantidad de iteraciones ralentizando el proceso de entrenamiento, mientras que un valor muy alto puede provocar que no se converja al mínimo.

A su vez, la técnica de descenso por gradiente no garantiza la convergencia a un mínimo global, por lo que si la función de costo no es convexa, es probable que se converja a un mínimo local.

Normalmente, debido a limitaciones computacionales, el cálculo del gradiente no se realiza sobre la base de datos completa sino que sobre muestras de la misma. Si se calcula el gradiente en base a una muestra por vez, el algoritmo se denomina descenso por gradiente estocástico. Si en cambio el gradiente se computa con N>1 muestras en cada iteración, se denomina descenso por gradiente mediante mini-lotes. Cuanto mayor el tamaño del lote (batch size), menor la varianza introducida en la estimación del gradiente. El ruido introducido por la varianza puede llegar a ser beneficioso si ayuda al algoritmo a escapar de un mínimo local. A su vez, el descenso por gradiente tradicional es lento para entrenar redes neuronales profundas. En cambio, se utilizan algoritmos basados en momento, tales como el momento acelerado de Nesterov [91], ADAGrad [92] y Adam [82], los cuales tienen en cuenta los gradientes pasados para actualizar los parámetros.

I.2 PROPAGACIÓN HACIA ATRÁS DEL ERROR

Para calcular el gradiente de la función de costo respecto a pesos y umbrales se utiliza la técnica de propagación hacia atrás del error, la cual permite con un bajo costo computacional actualizar los parámetros de la red neuronal. A continuación, se deriva matemáticamente el mecanismo.

Se denotará b_j^L al umbral de la neurona j en la capa L. Los umbrales forman un vector b^L. Se denomina w_{jk}^L a los pesos que interconectan la neurona k en la capa $L-1$ con la neurona j en la capa L, los cuales forman una matriz W^L. Las activaciones de la capa L serán:

$$a_j^L = \sigma(\sum_k w_{jk}^L a_k^{L-1} + b_j^L) \tag{I.5}$$

, en donde σ es la función de activación. Expresado de manera vectorial se tiene:

$$a^L = \sigma(w^L a^{L-1} + b^L) \tag{I.6}$$

La entrada efectiva a la neurona j de la capa L es:

$$z_j^L = \sum_k w_{jk}^L a_k^{L-1} + b_j^L \tag{I.7}$$

Se define el error en la neurona j de la capa L como:

$$\delta_j^L = \frac{\partial C}{\partial z_j^L} \tag{I.8}$$

Aplicando regla de cadena en la expresión I.8 se tiene:

$$\delta_j^L = \frac{\partial C}{\partial z_j^L} = \frac{\partial C}{\partial a_j^L}\frac{\partial a_j^L}{\partial z_j^L} = \frac{\partial C}{\partial a_j^L}\sigma'(z_j^L) \tag{I.9}$$

, lo cual vectorialmente equivale a:

$$\delta^L = \nabla_a C \odot \sigma'(z^L) \tag{I.10}$$

, en donde $\odot$ es el producto elemento a elemento, denominado producto de Schur o Hadamard.

Si uno vuelve a aplicar regla de la cadena en I.9 pero derivando respecto a las entradas efectivas en vez de las activaciones se tiene:

$$\delta_j^L = \sum_k \frac{\partial C}{\partial z_k^{L+1}} \frac{\partial z_k^{L+1}}{\partial z_j^L} \tag{I.11}$$

A su vez, el primer factor en la sumatoria es el error en la capa siguiente, y desarrollando z_k^{L+1} se tiene:

$$z_k^{L+1} = \sum_j w_{kj}^{L+1} a_j^L + b_k^{L+1} = \sum_j w_{kj}^{L+1} \sigma(z_j^L) + b_k^{L+1} \tag{I.12}$$

, por ende:

$$\frac{\partial z_k^{L+1}}{\partial z_j^L} = w_{kj}^{L+1} \sigma'(z_j^L) \tag{I.13}$$

Reemplazando I.13 en I.11 se obtiene:

$$\delta_j^L = \sum_k \delta_k^{L+1} w_{kj}^{L+1} \sigma'(z_j^L) \tag{I.14}$$

La ecuación I.14 se puede expresar de manera vectorial como:

$$\delta^L = ((w^{L+1})^T \delta^{L+1}) \odot \sigma'(z^L) \tag{I.15}$$

Usando la expresión I.9 es posible computar el error en la capa de salida, y luego utilizando la expresión I.15 se puede calcular el error en cada una de las capas ya que permite obtener el error en una capa a partir del error en la siguiente. Ahora es necesario obtener expresiones para los pesos y umbrales:

$$\frac{\partial C}{\partial b_j^L} = \frac{\partial C}{\partial a_j^L} \frac{\partial a_j^L}{\partial z_j^L} \frac{\partial z_j^L}{\partial b_j^L} = \delta_j^L \frac{\partial z_j^L}{\partial b_j^L} \tag{I.16}$$

$$\frac{\partial C}{\partial w_{jk}^L} = \frac{\partial C}{\partial a_j^L} \frac{\partial a_j^L}{\partial z_j^L} \frac{\partial z_j^L}{\partial w_{jk}^L} = \delta_j^L \frac{\partial z_j^L}{\partial w_{jk}^L} \tag{I.17}$$

Derivando la expresión de la entrada efectiva I.7 respecto a los pesos y umbrales se tiene:

$$\frac{\partial z_j^L}{\partial b_j^L} = 1 \tag{I.18}$$

$$\frac{\partial z_j^L}{\partial w_{jk}^L} = a_k^{L-1} \tag{I.19}$$

Reemplazando I.18 en I.16 se obtiene:

$$\frac{\partial C}{\partial b_j^L} = \delta_j^L \tag{I.20}$$

,y reemplazando I.19 en I.17 se tiene:

$$\frac{\partial C}{\partial w_{jk}^L} = \delta_j^L a_k^{L-1} \tag{I.21}$$

En el proceso de entrenamiento se toman M muestras x_i del conjunto de entrenamiento y se calculan las entradas efectivas z^L y las activaciones a^L. Se producen salidas y se calcula el error en las neuronas de la última capa mediante la aplicación de la ecuación I.9. Luego se calcula de manera sucesiva el error en cada capa mediante la ecuación I.15. Esto es lo que se denomina propagación hacia atrás del error. Luego se ajustan los umbrales y pesos mediante las siguientes ecuaciones:

$$b^L = b^L - \frac{\eta}{M} \sum_x \delta^{x,L} (a^{x,L-1})^T \tag{I.22}$$

$$w^L = w^L - \frac{\eta}{M} \sum_x \delta^{x,L} \tag{I.23}$$

, las cuales se obtienen al combinar las ecuaciones I.20 y I.21 con la I.3, realizando un promedio de los gradientes en las M muestras.

De las expresiones obtenidas pueden obtenerse varias conclusiones importantes a la hora de entrenar una red neuronal:

- Observando la ecuación I.21, si las activaciones de una capa son bajas, los pesos de la capa siguiente variarán poco, por lo cual el aprendizaje será lento. De esto se desprende que al usar activaciones ReLU, cuando la salida vale cero debido a que la entrada efectiva es negativa, los pesos asociados de la capa siguiente dejan de actualizarse.

- También observando I.21, si el error σ_j^L es bajo el aprendizaje será lento. En las expresiones del error I.9 y I.15 se ve que un factor es $\sigma'(z_j^L)$. Si la función de activación posee puntos de saturación (donde la derivada tiende a 0), el aprendizaje puede volverse lento. Esto ocurre, por ejemplo, en las funciones de activación logística y tangente hiperbólica.

- Inspeccionando la ecuación I.15, se observa que el error en la primer capa es el producto de los errores de las demás capas. A medida que el error se propaga hacia las

entradas, el número de factores multiplicándose en el cálculo del error es mayor. Si alguno de los factores posee un valor cercano a 0, o varios poseen valores menores a 1, el error resultante tenderá a 0 y la actualización de los pesos será prácticamente nula. Esto provoca que las capas cercanas a la entrada aprendan de una manera más lenta que las cercanas a la salida. Este problema, denominado desvanecimiento del gradiente, se da principalmente en redes neuronales con un número elevado de capas, y se potencia si se usan funciones de activación que saturen. También puede ocurrir que el gradiente explote si hay factores mayores a 1. Una forma de evitar este último problema es realizando recorte de gradiente [83], el cual consiste en limitar el valor máximo que pueden tomar los gradientes.

ANEXO II: RESULTADOS COMPLETOS

Tabla II.1: Resultados obtenidos con distintos tamaños de red neuronal.

Fuente	Métrica	Modelo			
		A	B	C	D
Bajo	SDR	2,16	2,28	2,26	**2,37**
	SIR	1,48	1,76	2,23	**2,44**
	SAR	**7,24**	6,81	6,42	6,36
	ISR	7,98	7,34	**8,40**	8,22
Batería	SDR	3,66	**3,95**	3,87	3,91
	SIR	5,75	6,48	6,66	**6,99**
	SAR	5,67	**6,07**	5,50	5,35
	ISR	7,76	8,00	8,53	**8,68**
Otros	SDR	2,36	**2,46**	2,30	2,35
	SIR	3,75	3,57	**5,04**	4,91
	SAR	3,97	**4,22**	2,63	2,76
	ISR	4,42	**5,20**	4,35	4,69
Voz	SDR	2,79	**2,97**	2,84	2,87
	SIR	4,59	5,23	4,96	**5,28**
	SAR	6,10	5,86	**6,27**	5,97
	ISR	8,54	8,85	10,21	**10,23**

Tabla II.2: Resultados obtenidos al aplicar distintas transformaciones a los datos de entrada.

Fuente	Métrica	Representación		
		Abs	Log2	Log10
Bajo	SDR	1.20	**1.31**	0.79
	SIR	0.35	**0.51**	-0.46
	SAR	**6.61**	5.69	6.38
	ISR	7.60	7.46	**7.91**
Batería	SDR	**5.37**	5.32	4.87
	SIR	8.03	**8.33**	7.24
	SAR	**7.61**	7.30	7.35
	ISR	9.55	**10.01**	9.63
Otros	SDR	**2.55**	2.49	2.19
	SIR	4.54	**5.12**	4.30
	SAR	**4.38**	3.62	3.29
	ISR	4.29	**4.61**	4.21
Voz	SDR	3.08	**3.13**	2.70
	SIR	4.80	5.71	**5.73**
	SAR	**5.26**	4.52	3.23
	ISR	**8.85**	8.55	6.67

Tabla II.3: Resultados obtenidos al aplicar dropout.

Fuente	Métrica	Sin Dropout	Dropout	
			A	B
Bajo	SDR	**1.31**	1.21	1.08
	SIR	0.51	0.14	**0.58**
	SAR	5.69	**6.69**	5.80
	ISR	**7.46**	7.32	7.46
Batería	SDR	**5.32**	5.12	5.23
	SIR	8.33	7.38	**9.35**
	SAR	7.30	**7.64**	6.56
	ISR	**10.01**	9.09	9.08
Otros	SDR	**2.49**	2.45	2.34
	SIR	**5.12**	3.67	3.08
	SAR	3.62	**5.17**	4.47
	ISR	**4.61**	4.17	4.15
Voz	SDR	**3.13**	3.00	2.27
	SIR	**5.71**	4.51	2.30
	SAR	4.52	5.95	**7.08**
	ISR	8.55	7.58	**8.76**

Tabla II.4: Resultados obtenidos al aplicar promediado temporal en las salidas.

Fuente	Métrica	Sin Promediar	Promediado
Bajo	SDR	1.31	**1.41**
	SIR	**0.51**	0.30
	SAR	5.69	**6.61**
	ISR	7.46	**7.48**
Batería	SDR	5.32	**5.41**
	SIR	**8.33**	7.79
	SAR	7.30	**8.02**
	ISR	**10.01**	9.89
Otros	SDR	2.49	**2.61**
	SIR	**5.12**	5.05
	SAR	3.62	**4.50**
	ISR	**4.61**	4.41
Voz	SDR	3.13	**3.36**
	SIR	**5.71**	5.48
	SAR	4.52	**5.36**
	ISR	**8.55**	8.34

Tabla II.5: Resultados obtenidos en las arquitecturas de red evaluadas.

Fuente	Métrica	Original	Uso de Panorama	HPSS en canales	Red Convolucional Dual	Red Convolucional Dual + Descriptores de Fase
Bajo	SDR	1.31	1.23	**1.54**	1.39	1.19
	SIR	**0.51**	0.25	-0.07	0.44	0.39
	SAR	5.69	6.49	**6.72**	6.43	6.65
	ISR	7.46	7.29	6.16	6.97	**7.90**
Batería	SDR	**5.32**	5.12	4.73	5.28	5.29
	SIR	**8.33**	7.20	5.80	7.61	7.56
	SAR	7.30	7.86	**8.63**	7.97	8.03
	ISR	10.01	9.80	9.42	9.81	**10.20**
Otros	SDR	2.49	2.33	2.32	**2.83**	2.31
	SIR	5.12	4.49	4.10	5.49	**6.39**
	SAR	3.62	4.29	3.91	**4.98**	3.77
	ISR	4.61	4.12	3.90	**4.71**	3.33
Voz	SDR	3.13	3.21	3.06	**3.66**	3.55
	SIR	5.71	5.10	4.06	**5.75**	5.39
	SAR	4.52	5.97	6.21	6.18	**6.82**
	ISR	8.55	8.48	8.25	9.51	**10.37**

Tabla II.6: Resultados del sistema final obtenidos sobre el conjunto de prueba de la base de datos DSD100.

Fuente	Métrica	Sistema Final	
		Sin DA	Con DA
Bajo	SDR	2.71	**2.83**
	SIR	2.97	**3.28**
	SAR	**7.68**	7.51
	ISR	**7.90**	7.79
Batería	SDR	4.11	**4.31**
	SIR	5.89	**7.96**
	SAR	**6.90**	5.81
	ISR	**8.54**	7.86
Otros	SDR	2.78	**3.01**
	SIR	**5.37**	3.72
	SAR	4.53	**6.27**
	ISR	4.63	**6.07**
Voz	SDR	3.53	**3.66**
	SIR	5.32	**6.16**
	SAR	**7.47**	7.04
	ISR	**10.08**	9.01

Tabla II.7: Métricas del sistema final obtenidas mediante BSS_Eval para las mezclas del conjunto de prueba.

ID de	SDR				SIR				SAR				ISR			
Canción	Bajo	Batería	Otros	Voz	Bajo	Batería	Otros	Voz	Bajo	Batería	Otros	Voz	Bajo	Batería	Otros	Voz
1	5.01	0.53	2.34	0.55	7.71	-1.30	2.99	-1.01	9.41	4.43	4.63	6.39	7.64	6.53	3.77	6.78
2	5.34	5.34	1.00	4.14	14.93	7.26	-1.33	5.33	7.08	8.67	3.23	8.08	6.73	12.95	3.59	11.95
3	0.91	5.01	2.84	5.19	-0.49	7.87	2.55	6.94	6.78	7.18	6.11	8.23	9.22	7.98	5.21	8.81
4	3.77	2.79	1.29	4.69	6.74	4.66	-0.76	7.84	7.42	3.39	5.23	6.98	5.66	6.71	5.31	7.58
5	2.59	3.98	3.82	2.79	2.15	8.55	7.35	3.46	7.71	4.99	5.34	8.65	7.97	5.99	5.68	15.66
6	3.13	6.66	2.67	7.02	3.22	11.47	2.66	10.15	6.38	8.59	5.30	9.79	7.23	9.65	5.83	11.78
7	5.71	4.54	2.37	6.80	7.42	8.69	2.87	12.99	8.69	5.11	4.70	8.36	9.93	7.73	6.84	10.36
8	2.89	6.04	4.32	5.55	2.88	10.89	6.67	7.86	6.84	7.40	6.70	8.22	8.54	8.93	8.20	10.05
9	3.53	4.00	2.75	5.27	3.79	10.93	3.40	8.49	10.78	5.13	5.70	7.32	10.28	6.01	4.74	9.56
10	-1.84	1.56	3.79	4.27	-4.85	5.36	4.05	6.10	3.40	3.69	7.26	7.39	5.22	5.62	7.94	7.46
11	-0.09	4.82	3.73	1.30	-3.45	6.22	6.36	0.61	7.65	8.91	6.77	4.67	5.48	7.14	5.16	5.03
12	0.62	3.82	1.16	3.34	-2.91	6.91	-0.90	6.88	8.26	6.04	6.64	3.93	5.29	5.20	5.88	5.30
13	4.10	1.27	2.85	5.54	5.84	0.16	3.70	7.54	7.02	3.73	5.26	9.02	7.33	5.35	5.71	10.86
14	1.24	3.78	-1.42	3.32	0.68	11.55	-6.07	4.90	5.40	4.14	3.63	8.19	4.25	5.21	3.79	8.91
15	1.05	3.39	3.46	6.21	-1.07	7.88	4.28	8.42	5.29	3.21	6.38	9.74	5.33	5.36	5.46	12.36
16	3.34	7.54	3.62	4.46	4.66	12.04	3.99	9.40	5.61	10.22	7.10	5.79	8.31	12.36	7.17	6.68
17	3.18	4.97	4.59	2.12	2.51	6.11	6.60	2.33	7.86	8.25	7.59	4.97	6.18	8.65	7.84	6.32
18	0.88	2.21	3.75	5.60	-2.36	2.54	4.70	9.75	9.01	4.30	8.52	6.65	5.88	5.16	5.16	9.37
19	3.14	5.07	4.07	4.90	3.74	8.87	5.70	7.99	6.35	6.36	6.92	7.30	8.52	10.22	7.64	8.83
20	5.94	-1.23	3.04	2.35	9.26	-3.81	5.77	3.14	8.54	1.02	4.74	7.43	9.58	3.35	4.41	10.51
21	1.63	2.24	3.34	5.41	0.66	5.46	4.92	10.80	8.68	3.29	5.98	7.47	9.39	4.55	6.20	9.32
22	0.80	7.64	3.84	1.85	-1.51	12.29	6.65	4.04	7.64	10.32	6.63	7.66	7.52	10.82	5.34	9.89
23	2.52	6.62	2.70	4.72	5.85	11.96	1.99	5.89	4.17	8.74	6.00	9.23	3.41	10.32	5.40	13.13
24	7.67	1.39	0.26	4.53	15.98	5.37	-0.65	6.66	12.97	-0.45	7.30	7.25	9.28	3.29	6.73	8.66

25	7.27	7.62	2.31	6.71	11.88	13.17	2.61	13.16	8.96	9.34	5.30	8.09	11.07	13.13	6.28	9.80
26	3.90	6.42	4.50	0.94	5.39	12.07	8.08	1.31	6.44	7.67	6.66	6.47	7.74	10.69	6.44	10.01
27	0.27	3.61	2.45	5.86	-0.05	8.12	3.58	7.48	7.31	4.42	3.69	10.01	12.91	5.17	4.06	11.47
28	5.44	2.79	5.59	3.64	6.94	6.08	8.87	5.22	9.52	3.40	7.64	7.38	8.70	4.30	9.15	10.76
29	4.45	6.03	3.63	0.70	7.31	11.85	5.20	3.58	7.49	7.68	6.12	8.84	7.29	11.58	6.56	10.44
30	3.17	6.95	2.24	7.58	4.57	14.99	1.74	14.16	6.72	8.17	6.68	9.40	5.32	10.21	7.68	11.66
31	-2.00	3.84	2.20	0.71	-8.41	8.60	1.83	4.49	5.99	5.93	4.18	6.86	3.32	5.40	5.37	9.06
32	3.87	4.09	2.70	2.72	3.53	6.69	2.09	4.23	8.49	5.66	7.52	4.73	9.20	8.01	5.33	4.36
33	5.62	1.33	3.89	5.41	10.62	2.38	5.36	8.12	7.35	3.41	6.65	7.44	8.73	7.27	9.43	10.61
34	6.78	4.14	3.89	3.83	9.42	8.87	4.66	6.09	10.73	4.55	7.30	6.71	9.78	6.46	7.51	10.82
35	1.74	3.37	3.67	7.03	1.38	6.31	6.85	9.91	6.83	3.68	5.87	9.75	10.11	6.49	5.43	13.05
36	-2.02	3.66	2.53	-7.72	-1.53	4.98	3.24	-4.80	9.92	5.78	7.20	2.87	12.66	10.13	6.19	-3.10
37	8.67	3.40	3.59	0.78	12.79	6.83	5.13	-0.60	10.55	5.13	7.26	6.44	14.43	8.58	5.10	7.67
38	1.23	3.37	2.03	1.74	-1.49	8.60	0.91	1.39	6.49	3.27	5.52	7.72	4.75	5.83	4.12	8.58
39	2.37	2.60	2.75	2.56	1.81	5.22	2.77	4.06	10.53	3.00	6.80	5.64	8.24	4.99	4.53	6.09
40	8.87	4.30	4.56	0.39	12.16	10.03	6.30	3.44	12.24	4.22	7.23	7.94	13.71	7.37	7.60	10.39
41	4.19	3.81	3.90	4.36	5.22	9.98	5.56	6.77	8.77	4.69	6.48	7.29	8.61	6.48	7.98	11.98
42	2.93	5.26	3.19	7.17	2.61	11.43	3.53	10.49	7.25	6.85	6.19	9.66	7.26	7.80	5.65	13.98
43	-4.23	1.98	4.08	1.71	-7.27	2.21	5.32	7.27	8.58	3.94	8.95	0.35	7.72	3.73	6.02	2.30
44	2.24	9.69	0.32	5.36	1.39	16.63	-2.93	10.47	5.27	11.35	4.78	6.52	4.50	12.90	6.07	11.14
45	0.10	3.92	4.23	3.24	-1.90	7.81	6.17	3.95	5.12	5.00	8.12	6.25	6.71	7.56	7.56	7.32
46	2.88	7.95	0.19	5.34	4.94	11.18	-2.02	10.40	3.44	10.42	3.91	7.42	5.36	16.23	5.17	7.72
47	1.18	3.22	3.03	2.35	-1.00	6.12	2.23	2.81	4.49	3.58	7.13	5.52	3.74	7.04	5.69	5.62
48	-0.44	8.64	4.94	1.51	-2.89	12.96	8.08	8.32	4.13	11.16	10.06	-1.04	6.30	14.71	7.93	2.62
49	4.92	3.63	4.28	2.75	5.41	6.81	6.71	4.30	9.43	4.15	6.38	6.48	11.17	6.29	6.57	9.41
50	0.80	5.67	3.79	4.66	-0.27	10.05	6.71	5.77	4.41	7.16	6.39	8.40	5.77	9.38	5.15	11.61

Tabla II.8: Métricas de energía del sistema final obtenidas mediante PEASS para las mezclas del conjunto de prueba.

ID de	Bajo				Batería				Otros				Voz			
Canción	SDR	SIR	ISR	SAR	SDR	SIR	ISR	SAR	SDR	SIR	ISR	SAR	SDR	SIR	ISR	SAR
1	4.58	5.89	7.28	17.15	0.66	-1.46	5.48	12.52	2.38	3.05	3.85	15.89	-1.03	-3.05	5.44	18.97
2	5.28	12.23	6.37	14.89	3.17	4.53	10.22	15.69	-2.84	-4.64	1.72	15.53	0.97	0.78	7.80	19.76
3	1.13	-0.98	8.65	14.10	5.05	6.80	7.31	14.43	2.81	1.89	5.16	17.38	5.28	6.30	8.04	21.40
4	3.83	5.35	5.56	17.00	2.72	2.63	5.25	13.60	1.29	-1.42	4.92	18.44	4.68	6.48	6.78	21.83
5	1.94	0.80	7.84	18.05	4.22	8.47	5.12	16.48	4.00	6.32	5.35	17.64	3.66	3.21	13.80	21.40
6	3.28	2.84	6.91	12.76	5.64	8.59	7.99	14.95	2.83	2.23	5.58	16.77	4.45	6.62	9.10	21.84
7	5.78	5.99	9.61	17.22	4.58	7.51	6.50	14.04	2.85	2.13	7.17	16.85	6.48	8.91	8.65	19.31
8	2.14	1.23	7.12	13.40	3.42	5.41	6.76	12.77	7.61	9.06	8.77	20.88	4.05	5.13	8.51	17.57
9	2.99	2.43	9.96	22.26	3.89	7.80	4.88	15.93	2.65	2.85	4.53	16.95	5.24	6.56	8.16	18.72
10	-5.95	-5.12	1.65	12.78	-3.06	0.50	2.07	12.27	3.98	4.60	7.78	18.55	1.15	3.27	4.83	19.00
11	-0.43	-3.96	5.28	16.64	4.49	4.24	6.50	16.03	3.95	6.30	4.95	18.34	-0.75	-2.25	3.95	17.89
12	0.03	-3.83	5.24	15.86	3.73	6.71	4.58	12.59	1.07	-1.56	5.47	17.68	3.70	5.07	5.18	18.08
13	4.00	4.03	6.87	15.51	1.22	-0.54	4.33	13.69	2.80	2.88	5.80	17.56	6.21	7.83	9.56	21.69
14	-2.62	-1.53	1.58	12.90	5.00	9.98	5.84	12.70	-1.27	-5.48	3.58	12.99	3.49	5.74	7.53	18.39
15	1.12	-1.17	5.06	15.06	3.08	4.86	4.32	11.02	3.62	3.77	5.55	17.30	5.89	6.00	10.44	20.12
16	3.17	2.75	7.34	12.59	8.35	11.86	11.00	18.27	3.35	3.13	7.20	17.18	4.63	7.09	6.08	16.45
17	1.99	0.61	5.60	13.96	5.19	6.15	7.76	14.42	5.41	6.82	7.77	20.87	2.18	0.93	5.63	17.61
18	-0.22	-2.61	4.07	16.96	-0.17	-0.54	4.27	11.08	4.22	4.30	5.08	17.42	-1.02	2.37	4.98	15.96
19	2.21	1.90	7.12	12.36	7.05	11.10	9.43	16.01	3.08	3.23	7.40	17.18	4.46	6.47	6.73	18.03
20	5.82	6.85	7.57	19.76	-1.24	-4.03	2.29	13.29	3.11	4.27	3.98	17.24	2.51	2.04	9.61	20.13
21	1.95	0.51	8.24	17.04	1.33	3.26	3.25	12.59	3.30	3.93	5.98	17.57	6.59	9.60	8.70	20.38
22	0.77	-1.68	7.67	14.26	8.44	12.17	10.62	17.77	4.68	6.73	5.85	19.00	4.54	4.24	9.61	21.18
23	2.43	4.73	3.12	12.91	7.11	11.15	9.26	17.36	2.78	1.67	5.31	16.66	3.34	3.41	11.55	20.67
24	7.32	12.29	8.52	23.55	1.73	3.53	2.95	13.79	1.99	0.76	7.78	21.74	3.85	3.64	7.33	21.42

25	7.03	8.64	10.44	16.21	7.48	10.97	10.56	17.37	2.01	1.06	6.21	17.20	6.56	9.50	8.59	21.15
26	3.99	4.03	7.39	14.44	6.54	10.42	8.83	16.06	4.82	7.34	6.21	17.81	0.24	-0.18	8.02	18.95
27	-1.88	-0.61	8.48	15.43	3.51	5.76	4.40	12.33	1.67	0.60	3.79	15.56	6.07	7.24	10.05	20.83
28	5.51	6.72	8.38	18.34	2.46	5.08	4.08	13.37	7.24	9.93	9.37	22.74	2.88	2.06	9.04	20.06
29	4.56	5.30	6.86	17.44	5.49	8.77	7.94	16.93	5.15	6.71	7.21	20.69	0.11	2.27	7.75	20.25
30	3.12	3.69	5.06	18.27	7.33	11.77	8.63	18.82	2.87	1.92	7.16	20.18	8.36	11.93	10.55	20.07
31	-2.37	-8.31	3.10	16.04	3.97	7.88	4.78	14.35	2.81	2.67	5.28	14.80	-2.54	2.37	4.43	18.31
32	3.92	2.96	8.97	16.19	-0.32	1.96	4.69	13.01	3.16	4.21	5.25	19.03	-2.28	-1.13	2.16	17.24
33	5.90	9.29	7.98	15.65	1.27	2.00	5.82	14.35	4.88	3.46	8.53	19.05	6.34	7.50	10.09	17.42
34	4.62	6.74	7.93	16.77	2.37	5.63	5.24	11.94	4.61	5.72	7.63	17.74	4.00	3.98	9.66	17.85
35	-3.96	-0.90	4.66	15.32	1.45	2.03	5.20	12.43	3.62	5.47	5.12	17.93	6.61	8.10	10.40	22.43
36	4.82	6.56	7.82	17.40	3.94	5.02	8.37	12.53	3.80	6.03	5.79	18.41	-18.65	-12.05	-8.05	15.37
37	8.78	10.51	13.62	14.46	4.98	7.20	8.20	13.22	3.58	5.02	5.15	18.21	2.49	1.70	7.58	19.03
38	-1.08	-2.69	3.77	13.89	2.25	6.60	4.50	10.85	0.11	1.27	2.69	14.92	2.57	2.87	8.34	18.42
39	3.13	2.66	8.10	23.31	1.53	2.68	3.63	14.37	2.67	1.96	4.33	19.25	1.50	0.89	5.11	18.70
40	8.51	9.52	12.74	20.06	4.35	8.25	5.95	13.51	5.21	7.41	7.41	20.22	-9.64	-2.54	1.44	19.30
41	3.18	3.01	7.74	15.63	3.58	6.53	5.11	12.43	4.73	5.58	7.43	18.53	2.28	2.89	7.32	19.12
42	1.38	0.91	6.12	15.61	3.10	7.31	5.63	14.27	3.99	4.53	6.11	18.76	-0.01	2.22	8.42	21.43
43	-3.70	-7.05	6.78	17.23	2.03	1.48	3.48	13.09	4.26	4.81	6.09	18.41	0.55	1.90	1.67	15.57
44	2.57	2.11	4.11	12.46	7.81	10.46	10.62	16.48	2.63	0.79	6.16	16.84	4.97	6.53	9.05	17.96
45	0.26	-2.28	6.34	16.17	4.17	6.93	6.50	13.72	6.27	8.35	7.63	22.26	3.34	2.69	6.61	18.59
46	3.03	3.55	5.87	12.31	8.24	11.35	12.48	19.26	-0.07	-3.04	4.67	15.31	5.52	9.36	7.08	18.92
47	1.13	-1.42	3.23	12.97	1.95	3.81	5.40	11.20	2.71	1.60	5.85	16.50	-0.74	0.13	3.50	16.78
48	-0.28	-3.35	6.11	11.58	8.83	12.02	12.60	17.90	5.05	7.58	7.35	22.59	-0.69	-0.40	1.87	13.71
49	4.83	4.44	10.63	18.33	3.68	5.96	5.39	14.02	4.37	5.78	6.16	18.19	2.28	2.89	7.32	19.12
50	0.69	-1.27	5.63	14.66	5.65	9.04	8.05	15.37	3.79	5.63	5.01	17.28	4.39	4.05	10.35	20.81

Tabla II.9: Métricas de puntaje perceptual del sistema final obtenidas mediante PEASS para cada mezcla del conjunto de prueba.

ID de	Bajo				Batería				Otros				Voz			
Canción	OPS	TPS	IPS	APS	OPS	TPS	IPS	APS	OPS	TPS	IPS	APS	OPS	TPS	IPS	APS
1	17.32	9.60	29.00	13.54	14.97	24.92	15.68	34.64	15.09	20.96	14.69	19.63	13.00	38.99	8.68	50.66
2	17.13	6.48	29.35	6.96	25.29	38.05	23.30	45.73	18.26	20.08	11.67	25.72	15.08	29.99	14.47	49.49
3	17.96	6.63	32.13	14.33	23.41	8.69	30.55	20.02	13.18	12.90	6.13	24.80	13.77	20.43	12.62	28.86
4	25.44	24.91	32.09	39.75	22.72	14.07	30.76	30.73	9.99	9.88	4.63	18.80	20.24	29.04	15.91	48.09
5	16.88	7.43	22.64	28.96	25.19	0.12	21.21	2.22	11.20	27.60	17.98	34.89	10.37	48.67	12.09	62.15
6	21.65	13.73	32.39	27.78	26.51	16.23	41.96	17.61	18.59	26.71	15.43	45.93	20.33	34.26	19.22	41.37
7	17.91	8.67	23.76	20.38	28.99	24.90	38.18	36.66	12.21	32.15	9.39	57.37	18.49	36.73	21.08	36.42
8	20.38	20.59	31.20	21.75	18.27	12.87	24.17	24.21	42.42	45.80	39.52	62.56	16.97	35.31	16.09	44.36
9	18.14	10.67	20.54	26.94	22.51	15.12	36.73	18.56	13.93	22.12	10.55	44.08	20.32	32.38	20.20	53.35
10	16.33	11.09	38.98	17.46	20.45	24.21	37.88	21.52	13.87	29.93	17.27	56.32	17.21	15.40	14.96	30.01
11	12.82	15.34	15.70	5.58	24.65	21.04	30.90	31.69	25.40	31.03	23.46	34.59	10.72	46.62	7.58	24.94
12	16.96	3.07	19.07	8.57	20.12	10.74	28.72	26.68	12.25	21.87	6.20	44.71	20.24	19.00	21.11	24.58
13	20.53	4.54	27.04	19.10	13.99	1.53	23.08	13.41	15.23	21.58	11.36	36.07	10.26	26.20	14.90	52.90
14	16.76	19.96	26.06	29.70	22.56	10.81	45.13	14.24	15.11	41.52	11.76	54.43	27.75	47.60	37.03	55.01
15	22.70	6.84	29.56	20.26	19.83	9.12	26.81	15.09	17.49	25.80	17.35	33.80	14.99	22.63	9.35	43.32
16	13.77	13.17	35.77	10.15	30.85	38.20	38.63	41.75	18.35	35.26	20.78	52.16	17.60	34.29	24.44	35.66
17	23.31	10.29	34.32	12.31	25.38	30.41	37.88	16.59	13.09	27.31	18.62	53.30	10.74	37.48	6.64	43.68
18	18.48	3.38	18.90	11.30	20.48	13.03	24.13	20.03	27.31	32.02	29.44	39.49	24.81	41.39	31.09	31.53
19	14.89	24.56	24.66	25.57	16.61	6.55	24.53	19.90	11.02	33.58	14.34	46.16	19.31	17.21	20.45	32.42
20	21.23	11.64	22.98	29.59	23.45	14.34	42.42	14.94	11.28	19.93	9.30	33.72	11.89	31.11	6.20	53.27
21	14.27	9.32	7.67	38.74	24.04	15.30	36.84	23.27	10.74	27.17	18.15	53.22	26.14	32.80	26.31	42.08

22	11.91	5.13	21.05	16.96	20.86	10.67	24.57	25.21	17.29	32.09	22.05	44.88	14.76	30.58	8.70	35.79
23	19.17	3.07	30.78	9.02	26.59	26.55	35.53	31.33	17.79	24.88	14.02	46.25	12.58	41.36	7.19	52.89
24	18.38	13.04	22.85	13.57	22.92	9.92	36.97	17.95	8.42	36.80	8.68	66.08	8.77	16.31	7.63	32.97
25	17.17	14.40	25.44	29.45	22.01	12.79	35.18	20.19	15.52	28.50	12.27	54.70	19.47	34.35	26.75	26.30
26	17.56	8.24	37.71	21.49	17.80	9.08	31.97	15.97	18.22	37.41	20.60	60.90	14.03	50.07	8.49	53.93
27	12.80	77.48	24.74	22.25	19.91	27.25	29.17	23.64	16.84	24.20	14.22	24.24	14.68	37.00	28.68	53.97
28	25.43	21.38	37.01	29.03	21.31	6.84	33.64	16.36	20.47	46.64	32.40	72.08	15.84	31.04	10.64	39.23
29	25.10	14.70	31.69	26.97	14.27	10.04	21.37	7.57	13.90	30.94	23.54	53.52	9.24	26.42	13.48	49.84
30	24.10	8.52	25.88	19.37	20.92	5.19	41.47	10.47	9.29	55.62	5.30	81.01	24.33	36.40	21.31	50.45
31	17.72	6.90	28.08	8.71	24.92	21.12	42.54	18.43	17.70	33.01	25.45	49.65	13.35	44.76	13.08	48.56
32	18.17	13.26	22.12	8.47	15.22	18.15	18.62	24.88	19.35	29.98	22.29	40.97	16.39	18.68	11.10	26.63
33	15.29	14.46	31.47	24.97	20.53	24.69	36.47	17.48	23.46	46.71	28.38	64.10	10.46	37.73	13.10	32.32
34	12.26	8.89	26.40	6.70	20.31	9.01	36.53	12.58	12.59	40.15	14.97	62.52	11.97	46.89	9.50	46.26
35	11.78	17.89	28.77	20.44	18.95	6.19	33.09	16.79	14.20	31.03	14.46	48.74	18.17	30.15	22.07	52.44
36	19.72	23.51	35.99	25.14	22.24	42.41	35.47	13.17	20.75	33.54	25.78	34.36	4.84	46.55	6.28	19.48
37	4.88	29.38	25.25	29.85	29.17	27.34	63.02	26.36	8.21	46.06	2.71	77.73	9.43	26.23	3.85	64.95
38	16.05	13.53	30.94	10.71	22.03	14.67	39.07	14.57	11.04	31.63	10.25	53.49	18.25	36.40	11.04	51.95
39	17.70	10.32	17.44	18.50	19.44	3.35	24.62	17.62	9.84	12.26	7.02	23.54	12.67	16.81	5.21	37.73
40	16.65	13.47	20.05	19.80	20.84	2.73	39.37	10.61	11.23	24.77	13.86	36.53	9.31	50.12	13.88	37.87
41	13.84	9.05	26.64	5.02	29.85	39.37	44.62	25.13	21.92	37.36	22.86	46.73	16.62	32.26	11.74	41.92
42	19.57	15.84	24.44	19.78	21.41	7.68	35.59	20.60	18.33	23.45	18.20	29.49	11.37	43.17	16.77	44.92
43	12.32	10.53	14.56	28.09	22.00	10.56	16.07	34.80	9.42	24.51	5.36	66.87	16.40	1.80	36.08	10.31
44	19.59	17.77	26.92	27.14	24.77	39.63	50.41	21.62	17.89	30.45	16.44	48.11	16.35	47.43	25.69	39.11
45	14.66	3.59	19.86	9.13	15.84	2.04	20.83	14.79	18.18	45.39	36.52	57.68	12.53	20.67	6.11	36.76
46	15.87	15.64	29.09	18.83	21.01	19.37	30.53	8.95	13.16	18.03	7.17	28.45	14.32	25.00	12.57	49.70
47	20.79	9.04	32.16	17.09	24.86	22.53	36.06	26.25	15.44	24.75	14.33	36.16	18.78	33.44	13.09	49.05

48	14.23	3.29	21.44	15.77	21.23	8.94	27.44	27.34	10.02	39.65	17.11	76.17	20.43	23.58	45.54	12.81
49	20.16	9.49	27.50	24.04	21.83	4.56	30.66	14.12	12.92	27.81	17.44	45.10	16.62	32.26	11.74	41.92
50	11.68	5.86	21.01	14.55	25.38	21.98	36.01	28.30	14.24	22.28	12.78	34.77	10.89	25.99	4.27	47.43

ANEXO III: CÓDIGO IMPLEMENTADO

MisCapas.py

```
from keras import backend as k
from keras.layers import  Multiply, Add
from keras.layers.core import Lambda
import numpy as np
import tensorflow as tf

def divide_layer(inputs):

    """Toma como argumento una lista de 2 entradas (inputs), y devuelve la división
        elemento a elemento de ambas."""

    eps = np.finfo(float).eps
    out = tf.div(abs(inputs[0])+eps,abs(inputs[1])+eps)
    return out

def softmask(layermultiout,layertomask):

 """Función para calcular máscaras suaves y aplicarlas.
 Argumentos:
 layermultiout: lista de capas con las cuales se calcula la máscara suave a partir de sus
     salidas.
 layertomask: lista de capas en las que se aplica la máscara suave en la salida.
 """

    bass = layermultiout[0]
    drums = layermultiout[1]
    others = layermultiout[2]
    vocals = layermultiout[3]

    mixture = Add()([bass,drums,others,vocals])

    vocbranch = Lambda(divide_layer)([vocals,mixture])
    drumsbranch = Lambda(divide_layer)([drums,mixture])
    othersbranch = Lambda(divide_layer)([others,mixture])
    bassbranch = Lambda(divide_layer)([bass,mixture])

    vocbranch = Multiply()([layertomask,vocbranch])
    drumsbranch = Multiply()([layertomask,drumsbranch])
    othersbranch = Multiply()([layertomask,othersbranch])
    bassbranch = Multiply()([layertomask,bassbranch])

    outlist = [bassbranch,drumsbranch,othersbranch,vocbranch]
    return outlist

def stacklayers(inputs):

 """Capa cuya salida consiste de la salida de las capas en inputs concatenadas."""

    return k.stack([inputs[0],inputs[1],inputs[2],inputs[3]],axis = 4)
```

```
48
49 def unstacklayers(inputs):
50
51  """Revierte el proceso de la función stacklayers"""
52
53     return [inputs[:,:,:,0],inputs[:,:,:,1],inputs[:,:,:,2],inputs[:,:,:,3]]
54
55 def log2emphasis(inputs):
56
57  """Capa que aplica el logaritmo en base 2 de la entrada + 1"""
58
59     log2value = np.log2(2)
60     emphasized = Lambda(lambda x: k.log(x+1)/log2value)(inputs)
61
62     return emphasized
```

MisCallbacks.py

```
1  import keras
2  import keras.backend as k
3  import pickle
4  import tensorflow as tf
5
6  def ConfigurarTF():
7
8      """Configura Tensorflow para un uso eficiente de la memoria de la GPU."""
9
10     config = tf.ConfigProto()
11     config.gpu_options.allow_growth = True
12     k.tensorflow_backend.set_session(tf.Session(config = config))
13
14 def LeerModelo(modelo,weightfile,optimizerfile = None):
15
16     """Permite cargar los parámetros de la red neuronal para continuar el entrenamiento o
           realizar predicciones.
17     Argumentos:
18     modelo: es el modelo compilado de Keras al cual se le cargaran los parámetros.
19     weightfile: es el archivo .hdf5 que contiene los pesos sinápticos.
20     optimizerfile: es el archivo .pkl que contiene los momentos del optimizador. Es
           necesario si se quiere continuar el entrenamiento, pero no lo es para realizar
           predicciones.
21     """
22
23     modelo.load_weights(weightfile)
24     modelo._make_train_function()
25
26     if optimizerfile != None:
27
28         with open(optimizerfile,'rb') as f:
29             weight_values = pickle.load(f)
30         modelo.optimizer.set_weights(weight_values)
31
32     return modelo
33
34
35 class GuardarModelo(keras.callbacks.Callback):
```

```
36
37      """Clase que permite guardar al finalizar cada época de entrenamiento los pesos
            sinápticos de la red en un archivo .hdf5 y el estado del optimizador en un .pkl."""
38
39      def __init__(self,filepath):
40
41          self.filepath = filepath
42
43      def on_epoch_end(self, epoch, logs = None):
44
45          filepath = self.filepath.format(epoch=epoch + 1, **logs)
46          #Guardo los pesos sinápticos:
47          self.model.save_weights(filepath,overwrite = True)
48          #Guardo parámetros del optimizador (necesarios si quiero reanudar entrenamiento)
49          symbolic_weights = getattr(self.model.optimizer,'weights')
50          weight_values = k.batch_get_value(symbolic_weights)
51          with open('optimizador.pkl','wb') as f:
52              pickle.dump(weight_values,f)
53
54
55  def CustomLossFunction(yTrue,yPred):
56
57      """Función de costo propuesta. Toma como argumentos el espectrograma verdadero y el
            predicho, y devuelve el error."""
58
59      BassTrue = yTrue[:,:,:,0]
60      BassPred = yPred[:,:,:,0]
61      DrumsTrue = yTrue[:,:,:,1]
62      DrumsPred = yPred[:,:,:,1]
63      OthersTrue = yTrue[:,:,:,2]
64      OthersPred = yPred[:,:,:,2]
65      VocalsTrue = yTrue[:,:,:,3]
66      VocalsPred = yPred[:,:,:,3]
67
68      basemse = k.mean(k.square(BassPred-BassTrue))+k.mean(k.square(DrumsPred-DrumsTrue))+k.
            mean(k.square(VocalsPred-VocalsTrue))+0.5*k.mean(k.square(OthersPred-OthersTrue))
69      diffmse = k.mean(k.square(BassPred-DrumsPred)+k.square(BassPred-OthersPred)+k.square(
            BassPred-VocalsPred)+
70                       k.square(DrumsPred-VocalsPred)+k.square(DrumsPred-OthersPred)+k.square
                             (VocalsPred-OthersPred),axis=-1)
71      othvocmse = k.mean(k.square(VocalsPred-OthersTrue),axis=-1)
72      othersmse = k.mean(k.square(VocalsPred-OthersTrue) + k.square(DrumsPred-OthersTrue) + k
            .square(BassPred-OthersTrue) ,axis=-1)
73      recons = k.mean(k.square(BassTrue + DrumsTrue + OthersTrue + VocalsTrue - BassPred -
            DrumsPred - OthersPred - VocalsPred),axis=-1)
74
75      alpha = 0.001
76      beta = 0.01
77      betav = 0.03
78      err = basemse - alpha*diffmse - beta*othersmse - betav*othvocmse + 0.01*recons
79
80      return err
```

ModeloDoble.py

```
1   from MisCallbacks import CustomLossFunction, VocalsError,DrumsError,BassError,OthersError,
```

```
    MetricInterference,MetricOthVoc,MetricOthers,MetricRecons
from MisCapas import softmask, stacklayers, unstacklayers, log2emphasis
from keras.layers import Input, Add, BatchNormalization, Concatenate
from keras.layers.convolutional import Conv2D, Conv2DTranspose
from keras.layers.core import Reshape, Dense, Flatten, Lambda
from keras.models import Model
from keras.optimizers import Adam

def CompileModel():

    """Función que compila el modelo de red neuronal implementado en Keras."""

    #Hiperparámetros de la subred percusiva:
    NVFiltPerc = 64
    NHFiltPerc = 64
    NReshapePerc = 1024

    #Encoder percusivo:
    stft_input = Input(shape = (1025,21,2,), dtype = 'float32')
    pvconv = Conv2D(NVFiltPerc,(1025,1),activation = "relu",use_bias = False)
    phconv = Conv2D(NHFiltPerc,(1,6),activation = "relu",use_bias = False)
    pflatter = Flatten()
    pencoder = BatchNormalization()(stft_input)
    pencoder = pvconv(pencoder)
    pencoder = BatchNormalization()(pencoder)
    pencoder = phconv(pencoder)
    pencoder = BatchNormalization()(pencoder)
    pencoder = pflatter(pencoder)

    #Hiperparámetros de la subred armónica:
    NVFiltHarm = 64
    NHFiltHarm = 32
    NReshapeHarm = 1536

    #Encoder armónico:

    hhconv = Conv2D(NHFiltHarm,(1,21),activation = "relu",use_bias = False)
    hvconv = Conv2D(NVFiltHarm,(82,1),activation = "relu",use_bias = False,strides = (41,1)
        )
    hflatter = Flatten()
    hencoder = BatchNormalization()(stft_input)
    hencoder = hhconv(hencoder)
    hencoder = BatchNormalization()(hencoder)
    hencoder = hvconv(hencoder)
    hencoder = BatchNormalization()(hencoder)
    hencoder = hflatter(hencoder)

    #Espacio Latente:
    latentspace = Concatenate()([hencoder,pencoder])
    latentspace = Dense(1024,activation = "relu",use_bias = False)(latentspace)

    #Capas de convolución transpuesta con pesos atados entre si (decoder percusivo):
    psharedHDeconv2D = Conv2DTranspose(NVFiltPerc,(1,6),activation = "relu")
    psharedVDeconv2D = Conv2DTranspose(2,(1025,1),activation = "relu")

    #Decodificadores paralelos percusivos para cada instrumento:
```

```
56      pbassbranch = Dense(NReshapePerc,activation = "relu")(latentspace)
57      pbassbranch = Reshape((1,16,NHFiltPerc))(pbassbranch)
58      pbassbranch = psharedHDeconv2D(pbassbranch)
59      pbassbranch = psharedVDeconv2D(pbassbranch)
60
61      pdrumsbranch = Dense(NReshapePerc,activation = "relu")(latentspace)
62      pdrumsbranch = Reshape((1,16,NHFiltPerc))(pdrumsbranch)
63      pdrumsbranch = psharedHDeconv2D(pdrumsbranch)
64      pdrumsbranch = psharedVDeconv2D(pdrumsbranch)
65
66      pothersbranch = Dense(NReshapePerc,activation = "relu")(latentspace)
67      pothersbranch = Reshape((1,16,NHFiltPerc))(pothersbranch)
68      pothersbranch = psharedHDeconv2D(pothersbranch)
69      pothersbranch = psharedVDeconv2D(pothersbranch)
70
71      pvocbranch = Dense(NReshapePerc,activation = "relu")(latentspace)
72      pvocbranch = Reshape((1,16,NHFiltPerc))(pvocbranch)
73      pvocbranch = psharedHDeconv2D(pvocbranch)
74      pvocbranch = psharedVDeconv2D(pvocbranch)
75
76      #Salidas de los decodificadores paralelos de la subred percusiva:
77
78      pbass = psharedVDeconv2D.get_output_at(0)
79      pdrums = psharedVDeconv2D.get_output_at(1)
80      pothers = psharedVDeconv2D.get_output_at(2)
81      pvocals = psharedVDeconv2D.get_output_at(3)
82
83      poutput = Lambda(stacklayers)([pbass,pdrums,pothers,pvocals])
84
85      #Capas de convolución transpuesta con pesos atados entre si (decoder armónico):
86      hsharedVDeconv2D = Conv2DTranspose(32,(82,1),activation = "relu",strides = (41,1))
87      hsharedHDeconv2D = Conv2DTranspose(2,(1,21),activation = "relu")
88
89      #Decodificadores paralelos armónicos para cada instrumento:
90      hbassbranch = Dense(NReshapeHarm,activation = "relu")(latentspace)
91      hbassbranch = Reshape((24,1,NVFiltHarm))(hbassbranch)
92      hbassbranch = hsharedVDeconv2D(hbassbranch)
93      hbassbranch = hsharedHDeconv2D(hbassbranch)
94
95      hdrumsbranch = Dense(NReshapeHarm,activation = "relu")(latentspace)
96      hdrumsbranch = Reshape((24,1,NVFiltHarm))(hdrumsbranch)
97      hdrumsbranch = hsharedVDeconv2D(hdrumsbranch)
98      hdrumsbranch = hsharedHDeconv2D(hdrumsbranch)
99
100     hothersbranch = Dense(NReshapeHarm,activation = "relu")(latentspace)
101     hothersbranch = Reshape((24,1,NVFiltHarm))(hothersbranch)
102     hothersbranch = hsharedVDeconv2D(hothersbranch)
103     hothersbranch = hsharedHDeconv2D(hothersbranch)
104
105     hvocbranch = Dense(NReshapeHarm,activation = "relu")(latentspace)
106     hvocbranch = Reshape((24,1,NVFiltHarm))(hvocbranch)
107     hvocbranch = hsharedVDeconv2D(hvocbranch)
108     hvocbranch = hsharedHDeconv2D(hvocbranch)
109
110     hbass = hsharedHDeconv2D.get_output_at(0)
111     hdrums = hsharedHDeconv2D.get_output_at(1)
```

```
112     hothers = hsharedHDeconv2D.get_output_at(2)
113     hvocals = hsharedHDeconv2D.get_output_at(3)
114 
115     houtput = Lambda(stacklayers)([hbass,hdrums,hothers,hvocals])
116 
117     #Fusión de las salidas de los decodificadores armónico y percusivo:
118     houtput = Lambda(lambda x: 2**x-1)(houtput)
119     poutput = Lambda(lambda x: 2**x-1)(poutput)
120     totaloutput = Add()([houtput,poutput])
121 
122     sourceoutputs = Lambda(unstacklayers)(totaloutput)
123 
124     #Cálculo y aplicación de la máscara suave:
125     salida = softmask(sourceoutputs,stft_input)
126     finaloutput = Lambda(stacklayers)(salida)
127     finaloutput = Lambda(log2emphasis)(finaloutput)
128 
129     #Definición del modelo de Keras:
130     modelodoble = Model(inputs = stft_input,outputs = finaloutput)
131     #Especificación del optimizador:
132     opt = Adam(lr = 0.01,clipvalue = 0.9,amsgrad = True)
133     #Se compila el modelo utilizando como función de pérdida la propuesta. También se
            especifican errores a mostrar durante el entrenamiento con el fin de monitorear el
            progreso.
134     modelodoble.compile(loss = CustomLossFunction,optimizer = opt,metrics = [VocalsError,
            DrumsError,BassError,OthersError,MetricInterference,MetricOthVoc,MetricOthers,
            MetricRecons])
135 
136     return modelodoble
```

trainmodel.py

```
1  from BatchGenerator import DataGenerator
2  from ValidationGenerator import ValidationDataGenerator
3  from keras.callbacks import TensorBoard
4  from MisCallbacks import GuardarModelo
5  
6  def trainmodel(model):
7      """Función que configura el entrenamiento del modelo y lo ejecuta."""
8      #Se usan generadores los cuales levantan los lotes de datos para entrenar la red:
9      training_generator = DataGenerator()
10     validation_generator = ValidationDataGenerator()
11 
12     #Guardado de pesos y estado del optimizador en cada época:
13     filepath = "weights-{epoch:02d}.hdf5"
14     checkpoint = GuardarModelo(filepath)
15 
16     #Despliegue de estadísticas de entrenamiento en Tensorboard
17     tbCallBack = TensorBoard(log_dir = './Graph', histogram_freq = 0, write_graph = True,
           write_images = False)
18     callbacklist = [checkpoint,tbCallBack]
19 
20     #Mediante esta función se entrena la red:
21     model.fit_generator(generator=training_generator,validation_data=validation_generator,
22                         epochs = 20, callbacks = callbacklist)
```

train.py

```
"""Script que ejecuta el entrenamiento del modelo. Permite leer un modelo ya entrenado y continuar el entrenamiento."""

import ModeloDoble
import trainmodel
from MisCallbacks import LeerModelo, ConfigurarTF

ConfigurarTF()
model = ModeloDoble.CompileModel()
#Comentar esta linea si el entrenamiento es desde cero:
model = LeerModelo(model,'weights-11.hdf5','optimizador.pkl')
trainmodel.trainmodel(model)
```

separate.py

```
import leeraudio
import tkinter as tk
from tkinter import filedialog
import ModeloDoble
import scipy.io.wavfile as wavfile
import scipy.signal as signal
import numpy as np

#Ventana para abrir archivos de audio:
root = tk.Tk()
root.withdraw()
file_path = filedialog.askopenfilename()
#Se lee el archivo seleccionado:
[fs,audiomixture] = leeraudio.ReadAudio(file_path)
#Se compila el modelo de Keras y se cargan los pesos:
model = ModeloDoble.CompileModel()
model.load_weights('weights-01.hdf5')

eps = np.finfo(float).eps
print("Analizando la señal")

#Se calcula la STFT, aplica el log2 de la magnitud y acondiciona el formato del tensor de entrada a la red:
[f,t,mixturestftl] = signal.stft(audiomixture[:,0]/(2**15-1),fs,window = 'hann',nperseg = 2048, noverlap = 2048-512)
[f,t,mixturestftr] = signal.stft(audiomixture[:,1]/(2**15-1),fs,window = 'hann',nperseg = 2048, noverlap = 2048-512)
magnitudestftin = np.array([np.log2(1+np.abs(mixturestftl)),np.log2(np.abs(mixturestftr)+1)])
sizestft = np.shape(magnitudestftin)
nframes = sizestft[2]
magnitudestftin = np.transpose(magnitudestftin,(1,2,0))
magnitudestftin = np.reshape(magnitudestftin,(1,1025,nframes,2))

#La red neuronal genera las salidas en un bucle:
print("Realizando la separación")
stftpredicted = np.empty((1025,nframes,2,4))
for i in np.arange(10,nframes-11,3):
    print("\r" + str(np.round(i/(nframes-11)*100,decimals = 1)) + "%",end='')
    prediction = model.predict(magnitudestftin[:,:,i-10:i+11,:])
```

```
37      #Se promedian las salidas:
38      stftpredicted[:,i-10:i+11,:,:] = stftpredicted[:,i-10:i+11,:,:] + (1/7)*prediction
            [0,:,:,:,:]
39
40  #Se vuelven a generar máscaras suaves y aplican en la STFT original:
41  stftpredicted = 2**stftpredicted - 1
42  denmask = np.sum(stftpredicted,axis = 3)
43  softmasks = np.zeros(np.shape(stftpredicted))
44  sources = np.zeros(np.shape(stftpredicted))
45  for i in range(4):
46      softmasks[:,:,:,i] = np.divide(stftpredicted[:,:,:,i],denmask+eps)
47      sources[:,:,:,i] = np.multiply(softmasks[:,:,:,i],np.transpose(np.array([np.abs(
            mixturestftl),np.abs(mixturestftr)]),(1,2,0)))
48
49  magnitudestftoutl = sources[:,:,0,:]
50  magnitudestftoutr = sources[:,:,1,:]
51
52  #Fases mezcla
53  phasestftl = np.angle(mixturestftl)
54  phasestftr = np.angle(mixturestftr)
55  j = np.complex(0,1)
56
57  #Se invierte la STFT de cada fuente y se guardan los resultados en archivos .wav
58  filename = file_path.split('/')
59  filename = filename[-1]
60  filename = filename.split('.')
61  filename = filename[0]
62  print("\nRealizando la inversión de la STFT")
63  for n, instrument in enumerate([filename+"_Bass.wav",filename+"_Drums.wav",filename+"_Other
        .wav",filename+"_Vocals.wav"]):
64
65      [t,xl] = signal.istft((np.multiply(magnitudestftoutl[:,:,n],np.exp(phasestftl*j))), fs
            = fs, window = 'hann', nperseg = 2048,noverlap = 2048-512,time_axis = 1, freq_axis
            = 0)
66      [t,xr] = signal.istft((np.multiply(magnitudestftoutr[:,:,n],np.exp(phasestftr*j))), fs
            = fs, window = 'hann', nperseg = 2048,noverlap = 2048-512,time_axis = 1, freq_axis
            = 0)
67      x = np.array([xl,xr])
68      x = x.astype('float32')
69      shx = np.shape(x)
70      tframes = shx[1]
71      x = np.reshape(x,(2,tframes))
72      x = np.transpose(x)
73      x = x
74      wavfile.write(instrument,fs,x)
```

BIBLIOGRAFÍA

[1] Cherry, E. C., "Some Experiments on the Recognition of Speech, with One and with Two Ears", *The Journal of the Acoustical Society of America*, 25, 5, 975-979, 1953.

[2] Simpson, A. J. R., Roma, G. y Plumbley, M. D., "Deep Karaoke: Extracting Vocals from Musical Mixtures Using a Convolutional Deep Neural Network", *CoRR*, abs/1504.04658, 2015.

[3] FitzGerald, D., "Upmixing from mono - A source separation approach", *2011 17th International Conference on Digital Signal Processing (DSP)*, 1-7, 2011.

[4] Esquef, P. A. A., Välimäki, V. y Karjalainen, M., "Restoration and Enhancement of Solo Guitar Recordings Based on Sound Source Modeling", *J. Audio Eng. Soc*, 50, 4, 227-236, 2002.

[5] Paulus, J. y Virtanen, T., "Drum transcription with non-negative spectrogram factorisation", *2005 13th European Signal Processing Conference*, 1-4, 2005.

[6] Herbst, C. T., Hertegard, S., Zangger-Borch, D. y Lindestad, P.-Å., "Freddie Mercury—acoustic analysis of speaking fundamental frequency, vibrato, and subharmonics", *Logopedics Phoniatrics Vocology*, 42, 1, 29-38, 2017.

[7] Beranek, L., *Acustica*, Segunda Edición, 1969.

[8] Sethares, W. A., *Rhythm and Transforms*, 1st, 2007.

[9] Manolakis, D. e Ingle, V., *Applied Digital Signal Processing*, 2011.

[10] Gabor, D., "Theory of communication. Part 1: The analysis of information", *Journal of the Institution of Electrical Engineers - Part III: Radio and Communication Engineering*, 93, 26, 429-441, 1946.

[11] Torrésani, B., "An Overview of Wavelet Analysis and Time-Frequency Analysis (a minicourse)", *International Workshop on Self-Similar Systems*, vol. JINR, E5-99-38, 9-34, 1998.

[12] Gerkmann, T., Krawczyk-Becker, M. y Roux, J. L., "Phase Processing for Single-Channel Speech Enhancement: History and recent advances", *IEEE Signal Processing Magazine*, 32, 2, 55-66, 2015.

[13] Auger, F. y Flandrin, P., "Improving the readability of time-frequency and time-scale representations by the reassignment method", *IEEE Transactions on Signal Processing*, 43, 5, 1068-1089, 1995.

[14] Banno, H., Takeda, K. e Itakura, F., “The effect of group delay spectrum on timbre”, *Acoustical Science and Technology*, 23, 1, 1-9, 2002.

[15] Prodeus, A., Didkovskyi, V., Didkovska, M. y Kotvytskyi, I., “On peculiarities of evaluating the quality of speech and music signals subjected to phase distortion”, *2017 IEEE 37th International Conference on Electronics and Nanotechnology (ELNANO)*, 455-460, 2017.

[16] Lipshitz, S. P., Pocock, M. y Vanderkooy, J., “On the Audibility of Midrange Phase Distortion in Audio Systems”, *J. Audio Eng. Soc*, 30, 9, 580-595, 1982.

[17] Griffin, D. y Lim, J., “Signal estimation from modified short-time Fourier transform”, *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 32, 2, 236-243, 1984.

[18] Cruz, P., *Inteligencia artificial con aplicaciones a la ingeniería*, 2011.

[19] Mcculloch, W. y Pitts, W., “A Logical Calculus of Ideas Immanent in Nervous Activity”, *Bulletin of Mathematical Biophysics*, 5, 127-147, 1943.

[20] Rosenblatt, F., “The Perceptron: A Probabilistic Model for Information Storage and Organization in The Brain”, *Psychological Review*, 65-386, 1958.

[21] Minsky, M. y Papert, S., *Perceptrons: An Introduction to Computational Geometry*, 1969.

[22] Rumelhart, D. E., Hinton, G. E. y Williams, R. J., “Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1”, ed. por Rumelhart, D. E., McClelland, J. L. y PDP Research Group, C., 1986, cap. Learning Internal Representations by Error Propagation, 318-362.

[23] Hornik, K., “Approximation capabilities of multilayer feedforward networks”, *Neural Networks*, 4, 2, 251-257, 1991.

[24] Russell, S. y Norvig, P., *Artificial Intelligence: A Modern Approach*, 3rd, 2009.

[25] Ruder, S., “An overview of gradient descent optimization algorithms”, *CoRR*, abs/1609.04747, 2016.

[26] Goodfellow, I., Bengio, Y. y Courville, A., *Deep Learning*, 2016.

[27] Bishop, C. M., *Pattern Recognition and Machine Learning (Information Science and Statistics)*, 2006.

[28] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. y Salakhutdinov, R., “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”, *Journal of Machine Learning Research*, 15, 1929-1958, 2014.

[29] Ioffe, S. y Szegedy, C., “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”, *CoRR*, abs/1502.03167, 2015.

[30] Glorot, X. y Bengio, Y., "Understanding the difficulty of training deep feedforward neural networks", *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, vol. 9, 249-256, 2010.

[31] He, K., Zhang, X., Ren, S. y Sun, J., "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification", *CoRR*, abs/1502.01852, 2015.

[32] Hubel, D. y Wiesel, T., "Receptive Fields and Functional Architecture of Monkey Striate Cortex", 195, 215-43, 1968.

[33] Fukushima, K., "Neocognitron: A Self-Organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position", *Biological Cybernetics*, 36, 193-202, 1980.

[34] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. y Jackel, L. D., "Backpropagation Applied to Handwritten Zip Code Recognition", *Neural Computation*, 1, 4, 541-551, 1989.

[35] Krizhevsky, A., Sutskever, I. e Hinton, G. E., "ImageNet Classification with Deep Convolutional Neural Networks", *Advances in Neural Information Processing Systems 25*, ed. por Pereira, F., Burges, C. J. C., Bottou, L. y Weinberger, K. Q., 2012, 1097-1105.

[36] Oord, A. van den, Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A. W. y Kavukcuoglu, K., "WaveNet: A Generative Model for Raw Audio", *CoRR*, abs/1609.03499, 2016.

[37] Piczak, K. J., "Environmental sound classification with convolutional neural networks", *2015 IEEE 25th International Workshop on Machine Learning for Signal Processing (MLSP)*, 1-6, 2015.

[38] Géron, A., *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*, 2017.

[39] Dumoulin, V. y Visin, F., *A guide to convolution arithmetic for deep learning*, cite arxiv:1603.07285, 2016.

[40] Grais, E. M. y Erdogan, H., "Single channel speech music separation using nonnegative matrix factorization and spectral masks", *2011 17th International Conference on Digital Signal Processing (DSP)*, 1-6, 2011.

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

[42] Grais, E. M. y Erdogan, H., "Single Channel Speech Music Separation Using Nonnegative Matrix Factorization with Sliding Windows and Spectral Masks", *INTERSPEECH*, 2011.

[43] A.S. Bregman, *Auditory scene analysis: The perceptual organization of sound*, 1990.

[44] Avendano, C., "Frequency-domain source identification and manipulation in stereo mixes for enhancement, suppression and re-panning applications", *2003 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (IEEE Cat. No.03TH8684)*, 55-58, 2003.

[45] Bonada, J., Loscos, A. y Vinyes, M., "Demixing Commercial Music Productions via Human-Assisted Time-Frequency Masking", *Audio Engineering Society Convention 120*, 2006.

[46] Mandel, M. I., Ellis, D. P. y Jebara, T., "An EM Algorithm for Localizing Multiple Sound Sources in Reverberant Environments", *Advances in Neural Information Processing Systems 19*, ed. por Schölkopf, B., Platt, J. C. y Hoffman, T., 2007, 953-960.

[47] Fitzgerald, D., "Harmonic/Percussive Separation using Median Filtering", *13th International Conference on Digital Audio Effects (DAFX10)*, 2010.

[48] Vincent, E., Gribonval, R. y Févotte, C., "Performance measurement in blind audio source separation.", *IEEE Trans. Audio, Speech & Language Processing*, 14, 4, 1462-1469, 2006.

[49] Févotte, C., Gribonval, R. y Vincent, E., *BSS_EVAL Toolbox User Guide – Revision 2.0*, type, INRIA, 2005, 19.

[50] Erruz, G., "Binaural source separation with convolutional neural networks", Master, Universitat Pompeu Fabra, 2017.

[51] Emiya, V., Vincent, E., Harlander, N. y Hohmann, V., "Subjective and Objective Quality Assessment of Audio Source Separation", *IEEE Transactions on Audio, Speech, and Language Processing*, 19, 7, 2046-2057, 2011.

[52] Hohmann, V., "Frequency analysis and synthesis using a Gammatone filterbank", *Acta Acustica united with Acustica*, 88, 3, 433-442, 2002.

[53] Huber, R. y Kollmeier, B., "PEMO-Q—A New Method for Objective Audio Quality Assessment Using a Model of Auditory Perception", *Trans. Audio, Speech and Lang. Proc.* 14, 6, 1902-1911, 2006.

[54] ITU-R Recommendation BS.1534-1, *Method for the subjective assessment of intermediate quality levels of coding systems*, inf. téc., Geneva, Switzerland: International Telecommunication Union, ene. de 2003.

[55] Cano, E., FitzGerald, D. y Brandenburg, K., "Evaluation of quality of sound source separation algorithms: Human perception vs quantitative metrics", *2016 24th European Signal Processing Conference (EUSIPCO)*, 1758-1762, 2016.

[56] Ward, D., Wierstorf, H., Mason, R. y Grais, E., "BSS Eval or Peass? Predicting the Perception of Singing-Voice Separation", *IEEE International Conference on Acoustic, Speech and Signal Processing (ICASSP)*, 2018.

[57] Emiya, V., Vincent, E., Harlander, N. y Hohmann, V., "Subjective and Objective Quality Assessment of Audio Source Separation", *IEEE Transactions on Audio, Speech, and Language Processing*, 19, 7, 2046-2057, 2011.

[58] Brown, G. y Cooke, M., "Perceptual Grouping of Musical Sounds: A Computational Model", 23, 107-132, 1994.

[59] Brown, G., Wang, D., Benesty, J., Makino, S. y Chen, J., *Separation of Speech by Computational Auditory Scene Analysis*, mar. de 2006.

[60] Lee, D. D. y Seung, H. S., "Algorithms for Non-negative Matrix Factorization", *Proceedings of the 13th International Conference on Neural Information Processing Systems*, 535-541, 2000.

[61] Joder, C., Weninger, F., Virette, D. y Schuller, B., "A comparative study on sparsity penalties for NMF-based speech separation: Beyond LP-norms", *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, 858-862, 2013.

[62] Lefèvre, A., Bach, F. y Févotte, C., "Itakura-Saito nonnegative matrix factorization with group sparsity", *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 21-24, 2011.

[63] Hayashi, A., Kameoka, H., Matsubayashi, T. y Sawada, H., "Non-negative periodic component analysis for music source separation", *2016 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA)*, 1-9, 2016.

[64] Virtanen, T., "Monaural Sound Source Separation by Nonnegative Matrix Factorization With Temporal Continuity and Sparseness Criteria", *IEEE Transactions on Audio, Speech, and Language Processing*, 15, 3, 1066-1074, 2007.

[65] Magron, P. y Virtanen, T., "Complex ISNMF: a Phase-Aware Model for Monaural Audio Source Separation", *ArXiv e-prints*, 2018.

[66] Grais, E., Umut Sen, M. y Erdogan, H., "Deep neural networks for single channel source separation", 2013.

[67] Roux, J. L., Hershey, J. R. y Weninger, F., "Deep NMF for speech separation", *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 66-70, 2015.

[68] Nugraha, A. A., Liutkus, A. y Vincent, E., "Multichannel Audio Source Separation With Deep Neural Networks", *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 24, 9, 1652-1664, 2016.

[69] Chandna, P., Miron, M., Janer, J. y Gómez, E., "Monoaural Audio Source Separation Using Deep Convolutional Neural Networks", 10169, 258-266, 2017.

[70] Jansson, A., Humphrey, E. J., Montecchio, N., Bittner, R. M., Kumar, A. y Weyde, T., "Singing Voice Separation with Deep U-Net Convolutional Networks", *ISMIR*, 2017.

[71] Takahashi, N. y Mitsufuji, Y., "Multi-scale Multi-band DenseNets for Audio Source Separation", *CoRR*, abs/1706.09588, 2017.

[72] Uhlich, S., Porcu, M., Giron, F., Enenkl, M., Kemp, T., Takahashi, N. y Mitsufuji, Y., "Improving music source separation based on deep neural networks through data augmentation and network blending", *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 261-265, 2017.

[73] Luo, Y., Chen, Z., Hershey, J., Le Roux, J. y Mesgarani, N., *Deep clustering and conventional networks for music separation: Stronger together*, mar. de 2017.

[74] Stoller, D., Ewert, S. y Dixon, S., "Adversarial Semi-Supervised Audio Source Separation applied to Singing Voice Extraction", *CoRR*, abs/1711.00048, 2017.

[75] Grais, E. M., Ward, D. y Plumbley, M. D., "Raw Multi-Channel Audio Source Separation using Multi-Resolution Convolutional Auto-Encoders", *CoRR*, abs/1803.00702, 2018.

[76] Park, S., Kim, T., Lee, K. y Kwak, N., "Music Source Separation Using Stacked Hourglass Networks", *CoRR*, abs/1805.08559, 2018.

[77] Takahashi, N., Goswami, N. y Mitsufuji, Y., "MMDenseLSTM: An efficient combination of convolutional and recurrent neural networks for audio source separation", *CoRR*, abs/1805.02410, 2018.

[78] Stoller, D., Ewert, S. y Dixon, S., "Wave-U-Net: A Multi-Scale Neural Network for End-to-End Audio Source Separation", *ArXiv e-prints*, 2018.

[79] Muth, J., Uhlich, S., Perraudin, N., Kemp, T., Cardinaux, F. y Mitsufuji, Y., "Improving DNN-based Music Source Separation using Phase Features", *ArXiv e-prints*, 2018.

[80] Liutkus, A., Stöter, F.-R., Rafii, Z., Kitamura, D., Rivet, B., Ito, N., Ono, N. y Fontecave, J., "The 2016 Signal Separation Evaluation Campaign", *Latent Variable Analysis and Signal Separation - 12th International Conference, LVA/ICA 2015, Liberec, Czech Republic, August 25-28, 2015, Proceedings*, 323-332, 2017.

[81] Pepino, L. y Bender, L., "Separación de fuentes musicales mediante redes neuronales convolucionales con múltiples decodificadores", *4° Jornadas de Acústica, Audio y Sonido*, 2018.

[82] Kingma, D. P. y Ba, J., "Adam: A Method for Stochastic Optimization.", *CoRR*, abs/1412.6980, 2014.

[83] Pascanu, R., Mikolov, T. y Bengio, Y., “Understanding the exploding gradient problem”, *CoRR*, abs/1211.5063, 2012.

[84] Pan, J., Liu, S., Sun, D., Zhang, J., Liu, Y., Ren, J. S. J., Li, Z., Tang, J., Lu, H., Tai, Y. y Yang, M., “Learning Dual Convolutional Neural Networks for Low-Level Vision”, *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[85] Pons, J., Lidy, T. y Serra, X., “Experimenting with musically motivated convolutional neural networks”, *2016 14th International Workshop on Content-Based Multimedia Indexing (CBMI)*, 1-6, 2016.

[86] Rousseeuw, P. y Hubert, M., “Robust statistics for outlier detection”, *Wiley Interdiscip. Rev. Data Min. Knowl. Discov.* 1, 73-79, 2011.

[87] Weninger, F., Le Roux, J., Hershey, J. y Watanabe, S., “Discriminative NMF and its application to single-channel source separation”, *Nuclear Physics A*, 865-869, 2014.

[88] Virtanen, T., Gemmeke, J. F., Raj, B. y Smaragdis, P., “Compositional Models for Audio Processing: Uncovering the structure of sound mixtures”, *IEEE Signal Processing Magazine*, 32, 2, 125-144, 2015.

[89] He, K., Zhang, X., Ren, S. y Sun, J., “Deep Residual Learning for Image Recognition”, *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 770-778, 2016.

[90] Huang, G., Liu, Z., Maaten, L. v. d. y Weinberger, K. Q., “Densely Connected Convolutional Networks”, *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2261-2269, 2017.

[91] Nesterov, Y., “A method of solving a convex programming problem with convergence rate O(1/sqr(k))”, *Soviet Mathematics Doklady*, 27, 372-376, 1983.

[92] Duchi, J., Hazan, E. y Singer, Y., “Adaptive Subgradient Methods for Online Learning and Stochastic Optimization”, *J. Mach. Learn. Res.* 12, 2121-2159, 2011.