Universidad Politécnica
de Madrid

Escuela Técnica Superior de

Ingenieros Informáticos

Máster Universitario en Inteligencia Artificial

Trabajo Fin de Máster

Eliminación de la Reverberación en Archivos de Audio Usando Deep Learning

Autor: Alberto Casado Garfia

Tutor: Francisco Serradilla García
Tutor: Lino García Morales

Madrid, Julio 2020

Este Trabajo Fin de Máster se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

Trabajo Fin de Máster
Máster Universitario en Inteligencia Artificial
Título: Eliminación de la Reverberación en Archivo de Audio Usando Deep Learning
Julio 2020

Autor: Alberto Casado Garfia

Tutor:
Francisco Serradilla García
Departamento de Inteligencia Artificial
ETSI Informáticos
Universidad Politécnica de Madrid
Tutor:
Lino García Morales
Departamento de Ingeniería Audiovisual y Comunicaciones
ETSI Informáticos
Universidad Politécnica de Madrid

Resumen

La reverberación es un fenómeno acústico que reduce la inteligibilidad de los diálogos de voz y la calidad de las grabaciones. Por esta razón se han realizado multitud de trabajos con el objetivo de eliminar la reverberación de las grabaciones de audio.

En los últimos años se ha comenzado a usar técnicas de Deep Learning obteniendo resultados prometedores. Este trabajo continúa esta línea probando la capacidad de la arquitectura Demucs, la cual está diseñada para separar pistas de música, para la nueva tarea de separar el audio emitido originalmente por la fuente, de las reverberaciones creadas por la estancia.

Palabras clave: Eliminación de la reverberación, Deep Learning, Audio, Reverberación

Abstract

The reverberation is an acoustic phenomenon which reduces the intelligibility of the voice recordings and the quality of the audio recordings. For this reason, new works have been made to remove the reverberation of the audio recordings, the process of dereverberation.

In the last years, Deep Learning technics have begun to be used with promising results. This work continues this line using the architecture Demucs, which is designed to separate music tracks, to implement the new task of separate the original audio, which comes from the audio source, from the reverberation created by the room.

Keywords: Dereverberation, Deep Learning, Audio, Reverberation

Tabla de contenidos

Índice de figuras

1 Introducción

En este trabajo se pretende usar técnicas de Deep Learning para resolver el problema de la eliminación de la reverberación en grabaciones de audio.

La reverberación es un fenómeno acústico que afecta a la calidad de las grabaciones, produciendo problemas en multitud de ámbitos. Las ondas sonoras rebotan en las paredes de las habitaciones creando ecos que interfieren con el sonido original. Este fenómeno es más común en grandes estancias y habitaciones sin amueblar y puede afectar a la grabación de audio o en la realización de videoconferencias.

Por esta razón, en los últimos años es en el realce del habla y en el reconocimiento automático del habla donde en donde más se realizan esfuerzos para resolverlo, ya que la reverberación decrementa la inteligibilidad de los diálogos de voz.

Una gran parte de este esfuerzo ha sido puesta en la eliminación del ruido, pero se siguen haciendo esfuerzos en reducir y eliminar la reverberación de las grabaciones de audio. Sin embargo, las características de la reverberación han impedido crear un método que elimine la reverberación en un ámbito general.

Este último aspecto está cambiando con la llegada del Aprendizaje Automático y más concretamente del Deep Learning aplicado a generación de audio. Los métodos focalizados en teoría de señales están dando paso a arquitecturas de redes neuronales o combinaciones de ambas técnicas que se entrenan usando conjuntos de datos.

1.1 Objetivos

El objetivo de este trabajo es implementar un modelo de red neuronal que resuelva el problema de la eliminación de la reverberación en una grabación de audio que no haya sido usado antes para esta tarea. La red deberá ser entrenada y evaluada para comprobar su capacidad de realizar la eliminación de la reverberación.

Se ha elegido usar como base la red Demucs[1], la cual esta entrenada para separar pistas de música. La intuición detrás de esta elección es la siguiente, si imaginamos la reverberación como un proceso aditivo, en el que a la señal original de la fuente se le suma una reverberación, este proceso es muy parecido al que ya realiza la red. En el caso de la música separa pistas y en la nueva tarea separa la reverberación del audio original.

Esta red no ha sido usada en otros trabajos con este objetivo y aunque redes similares se han probado con el mismo objetivo[2], [3], existen bastantes

diferencias entre las arquitecturas e implementaciones como para considerar que los resultados obtenidos pueden ser muy diferentes.

La red Demucs esta implementada usando Pytorch, por lo que este trabajo será realizado usando enteramente este framework.

2 Estado del Arte

2.1 Conocimientos previos

2.1.1 Sonido

El sonido es una onda de mecánica que aparece cuando una fuente de sonido hace que las partículas cercanas comiencen a vibrar[4]. Este movimiento se propaga gradualmente por el medio, creando ondas de presión que se alejan de la fuente.

Las vibraciones pueden ser capturadas por un receptor pudiendo ser tratados como una señal, como podemos ver en la Figura 1.

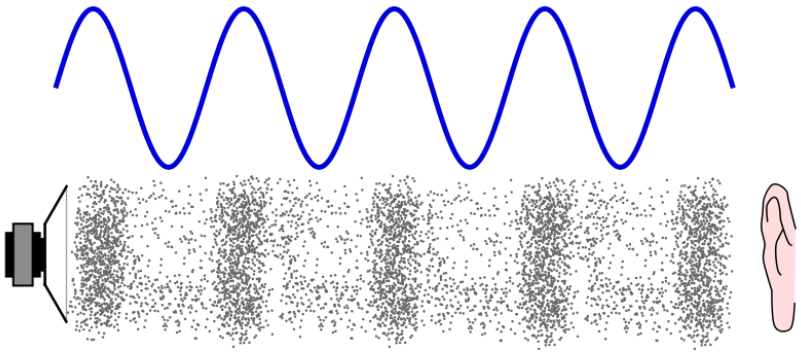


Figura 1 Representación de la vibración del aire(abajo) por una fuente sonora (a la izquierda) y su correspondiente señal acústica (arriba)

En una onda con una única frecuencia, la amplitud es la distancia hasta la cresta de la onda y el periodo el tiempo que tarda en repetirse la onda. La frecuencia es la inversa del periodo y se mide en Hertzios, la cantidad de periodos transcurridos un segundo.

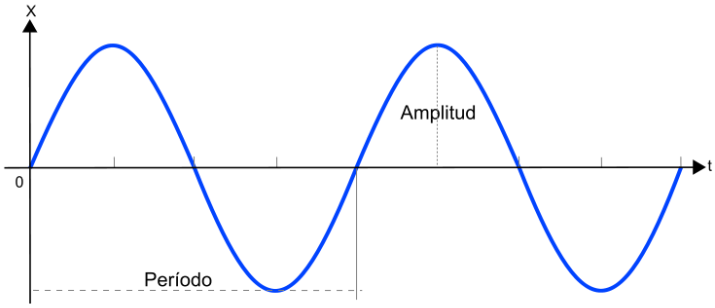


Figura 2 Gráfica de una señal con un tono puro, con la amplitud en el eje vertical y el tiempo en el horizontal

Sin embargo, en la realidad las ondas sonoras son más complejas, como podemos ver en la Figura 3, representada igualmente con el desplazamiento o amplitud de la onda en el eje vertical y en tiempo en el eje horizontal. Esta representación es la más utilizada y en la que se codifica generalmente el sonido, como veremos más adelante.

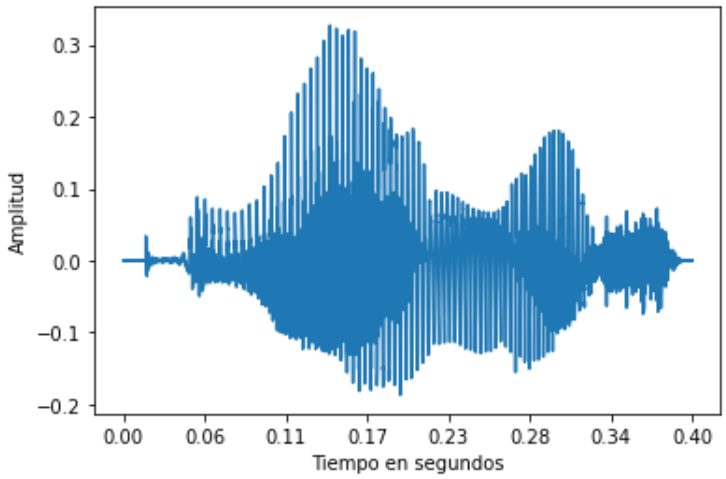


Figura 3 Gráfica donde se muestra la onda sonora de una voz humana

La presión sonora está determinada por la amplitud de la señal y se expresa en Pascales en el sistema internacional, sin embargo, es más común usar decibelios.

El decibelio es una medida adimensional que expresa la relación entre dos valores en una escala logarítmica, pero que es ampliamente usado en acústica para compar presiones.

En caso de trabajar con niveles de presiones sonoras se define como:

$$Lp = 10 \cdot log_{10}\left(\frac{P^2}{{P_{ref}}^2}\right)(dB) = 20 \cdot log_{10}\left(\frac{P}{P_{ref}}\right)(dB)$$

Donde P es la presión sonora medida, P_{ref} la presión de referencia y Lp el nivel de presión medido en decibelios.

Comparando la presión de una onda sonora que se desea medir con un valor de convenio muy bajo, normalmente el umbral mínimo de percepción del oído humano, 20 micropascales, obtenemos la definición de decibelio que es usada para describir la intensidad de fenómenos acústicos. En la Figura 4 podemos ver una gráfica donde en el eje vertical tenemos la intensidad del mismo sonido que en la figura 3 y en el eje horizontal el tiempo.

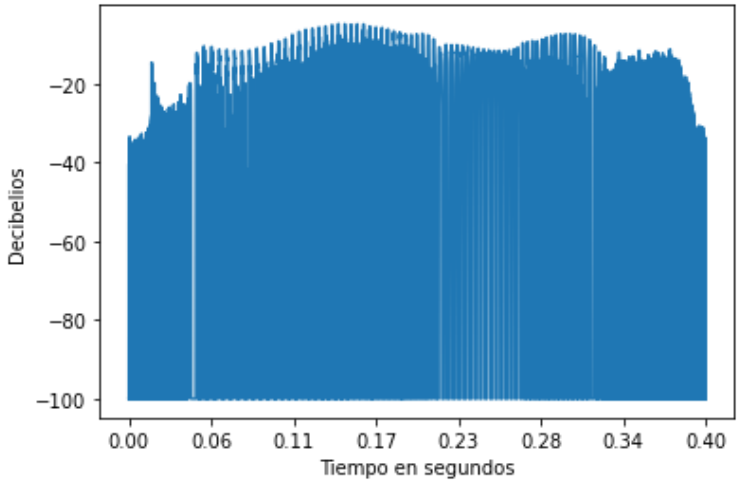


Figura 4 Intensidad del sonido de la Figura 3 en decibelios. La transformación ha sido realizada usando Pytorch Audio.

Existen otras medidas como la potencia o la intensidad sonoras que también se expresan en esta medida.

La potencia sonora, medida en vatios, mide la cantidad de energía sonora emitida por unidad de tiempo. Depende únicamente del sonido producido por la fuente.

La intensidad sonora representa la cantidad de energía sonora que atraviesa una unidad de área perpendicular a la dirección hacia donde se propaga el sonido en un periodo de tiempo. Se mide en W/m^2. Aunque en este trabajo se hable de intensidad de un sonido, este término no se referirá a esta medida en concreto, sino a la amplitud de la señal que vimos antes.

Mientras que la distancia del desplazamiento de las partículas en la vibración del aire genera la amplitud de la señal, la velocidad con la que este desplazamiento hace vibrar estas partículas produce las frecuencias de la señal. Los sonidos con mayor frecuencia, mayor número de repeticiones, son percibidos como agudos y los de menor frecuencia como graves.

Una señal acústica, como la que vemos en la Figura 1 puede separarse en diferentes señales más simples, cada una de ellas con una única frecuencia y amplitud, como la que aparece en la Figura 2. La suma de todas ellas volverá a generar la señal original. Este proceso de separación se realiza a usando la Transformada de Fourier, la cual convierte el eje temporal al dominio de la frecuencia.

Para una señal de entrada como en la Figura 3 la transformada de Fourier devolverá una señal donde el eje horizontal será cada una de las frecuencias que componen la señal original y el eje vertical su amplitud, como vemos en la Figura 5[5]. Este gráfico es el usado normalmente en los reproductores de música o ecualizadores de audio.

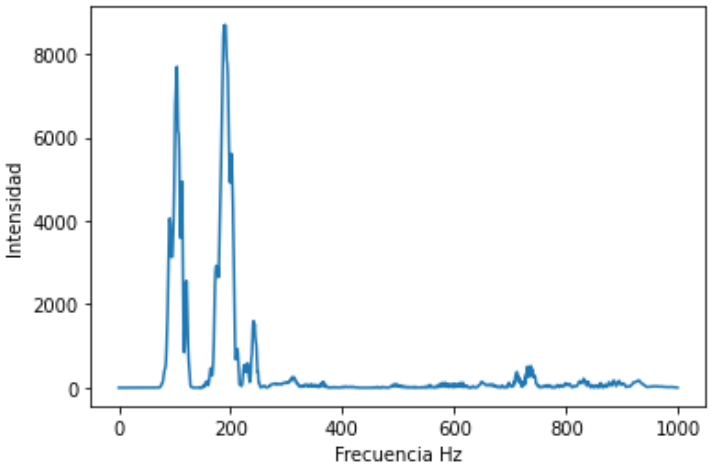


Figura 5 Gráfica en el que se visualiza el sonido de la Figura 3 tras aplicar la FFT. En el eje vertical se dispone la intensidad o amplitud y en el eje horizontal las frecuencias.

En computación se usa más comúnmente una variante llamada Transformada Rápida de Fourier o FFT, la cual tiene un menor coste computacional. La transformada es reversible usando su función inversa, lo que permite trabajar con la onda en el dominio que más sea conveniente en el modelo.

Aunque la transformada puede aplicarse a cualquier duración de audio, tiene más sentido usarse en longitudes cortas para observar cómo varia las frecuencias de una señal durante el tiempo, por ejemplo, para detectar las notas de una canción. Este procedimiento se denomina Transformada de Fourier de Tiempo Reducido o STFT.

Si al igual que hacemos en una convolución vamos desplazando una ventana sobre una señal de audio y en cada paso aplicamos la Transformada de Fourier obtenemos un Espectrograma de Frecuencias como el que vemos en la Figura 6.

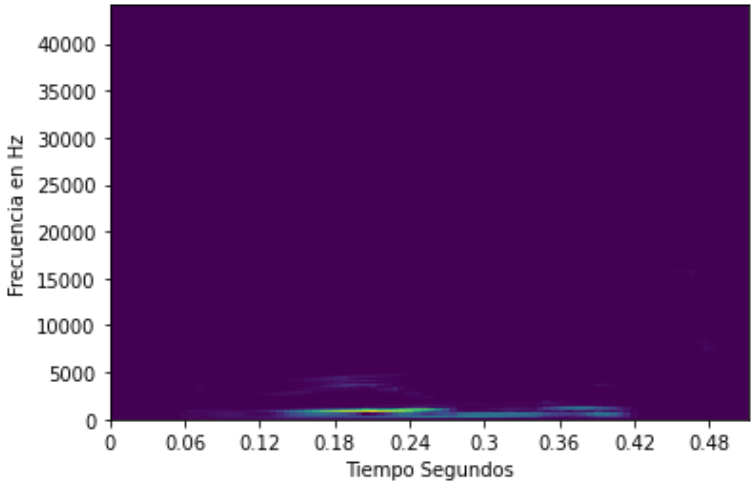


Figura 6 Espectrograma de frecuencias del sonido de la Figura 3

Al igual que en una convolución, al crear un espectrograma existen múltiples parámetros como el tamaño de la ventana, el avance de la ventana en cada paso

o el tamaño de la muestra que lee el FFT, permitiendo una mayor o menor resolución.

En la Figura 6 se puede observar que no hay frecuencias mayores de 5000 Hz debido a que la voz humana tiene un rango de 125 Hz a 8kHz mientras que el espectrograma se extiende hasta 40kHz. Para su uso en acústica se reescala esta matriz usando la escala logarítmica o la escala de Mel, como podemos ver en la Figura 7

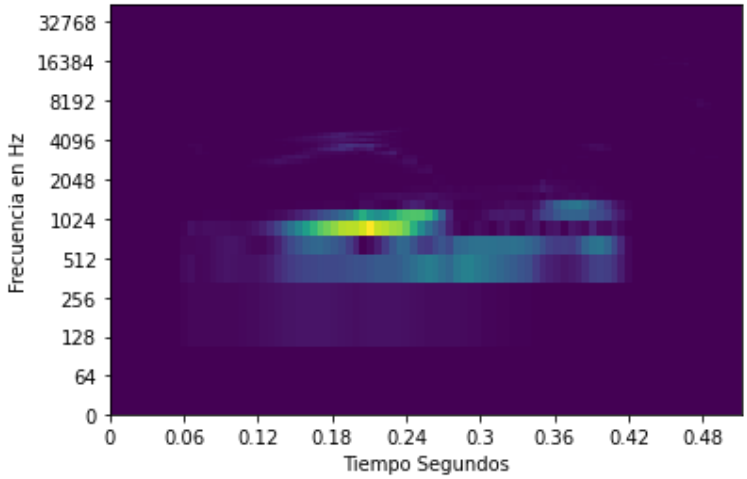


Figura 7 Frecuencias en la escala logarítmica

La escala de Mel imita como la percepción no lineal del oído humano siendo menos discriminativa con bajas frecuencias que con altas. Podemos convertir medidas de frecuencia en Hertzios f a los Mels m usando la siguiente ecuación:

$$m = 2595\, log_{10}(1 + \frac{f}{700})$$

A un espectrograma en la escala de Mel se le puede aplicar otras transformaciones para obtener un espectrograma usado por bastantes algoritmos llamado Coeficientes Cepstrales en las Frecuencias de Mel o MFCC.

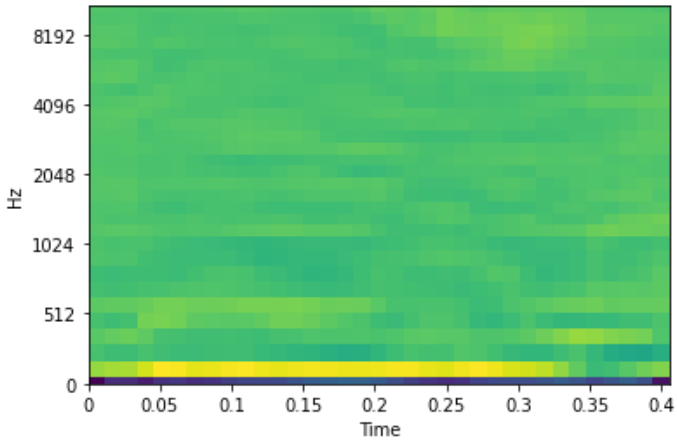


Figura 8 Frecuencias en un MFCC

Todas estas representaciones permiten digitalizar el audio que se ha capturado analógicamente, pero la forma más sencilla y utilizada es la de muestrear periódicamente una señal y almacenar su amplitud en un vector[6], como podemos ver en la Figura 3.

El formato WAV realiza exactamente este proceso. Otros formatos como MP3, FLAC, OGG comprimen el audio para reducir el tamaño de los audios en disco, por lo que necesitan ser descomprimidos por un decodificador para convertirse en un vector que podamos procesar.

El parámetro más importante para procesar esta lista es la tasa de muestreo, la cual indica cuantos elementos del vector se van a procesar en un segundo. Su inversa, la frecuencia de muestreo nos indica cuanto tiempo hay entre los elementos del vector, es decir el periodo en el que fueron muestreados, generalmente tiene una frecuencia de 44100Hz.

La profundidad de bits que mide cuantos bits se ha usado para almacenar cada elemento del vector. A mayor profundidad, mayor precisión para almacenar la amplitud.

Un audio puede tener varios canales por haber sido grabado por varios receptores. En caso de usar un audio con dos canales en estéreo se usará un vector por cada canal.

2.1.2 Reverberación

Desde que se realizan grabaciones de audio se han creado métodos para mejorar la calidad del audio obtenido, reduciendo o eliminando efectos indeseados que deterioran el audio que se ha grabado, ya sea de voces, instrumentos u otros sonidos.

De todos estos casos, el realce del habla es el que más atención ha recibido debido a su multitud de aplicaciones en mejora de la comprensión del habla, como llamadas en dispositivos móviles o en reconocimiento de voz en asistentes inteligentes.

Los tres ejes principales del realce del habla, que son igualmente aplicables al resto de casos de mejora de calidad del audio, son, según se expuso en el International Workshop on Acoustic Echo and Noise Control[7] la cancelación del eco, la reducción del ruido y la eliminación de la reverberación, siendo esta última la más difícil de resolver según el consenso y que sigue sin estar solventada en la totalidad de los casos.

El ruido es el conjunto de todos aquellos sonidos recogidos por el micrófono que no provienen de la fuente que queremos grabar. Se han desarrollado multitud de técnicas con el fin de eliminar distintos tipos de ruido de las grabaciones o reducir el ruido en espacios cerrados.

El eco y la reverberación están más interrelacionados. Cuando se realiza una grabación en un espacio cerrado las ondas sonoras rebotan contra las paredes, el techo, el suelo y otros objetos dentro de la estancia, generando replicas más atenuadas. Como el sonido se propaga por el aire a una velocidad de 340 m/s, estas replicas son grabadas por el micrófono con un retardo apreciable por el oído humano, que es de 50ms para conversaciones y 80ms al escuchar música.

Se denomina eco al fenómeno acústico de una única reflexión del sonido, el cual produce un sonido igual, pero de menor volumen. En cambio, la reverberación es la suma de todas las reflexiones del sonido grabadas por el micrófono, por lo que también se puede definir como la unión de todos los ecos producidos. Este sonido es muy distinto al de un único eco y al sonido original, pues las ondas de sonido se mezclan con diferentes retardos, creando un sonido difuso que se desvanece con el tiempo.

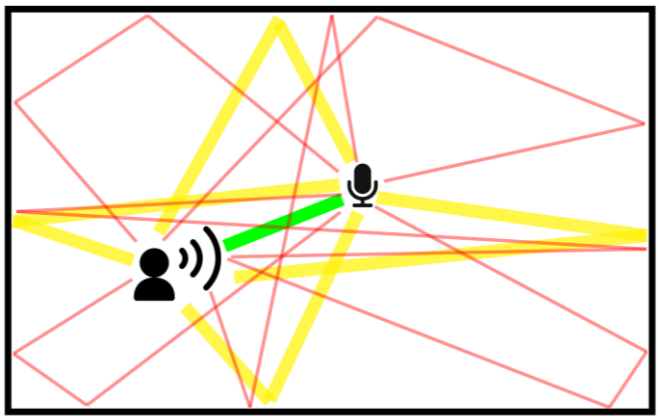


Figura 9 Aproximación de reflexiones de una reverberación, siendo la trayectoria más directa y por tanto la señal original la verde, las amarillas las reflexiones primordiales que solo han tenido una reflexión y las rojas las reflexiones con 2 reflexiones.

En la figura 9 podemos ver la trayectoria de las reflexiones que alcanzan el micrófono. Generalmente en las grabaciones es deseable únicamente capturar la primera onda recibida, aquella que ha realizado el recorrido más directo hasta el receptor, y desechar el resto de las reflexiones, en el caso de la ilustración, la línea verde.

Según transcurre el tiempo llegan al micrófono más reflexiones que han rebotado más veces en las paredes, por lo que están más atenuadas. Este fenómeno hace que haya diferencias entre las primeras reflexiones de una reverberación y las siguientes.

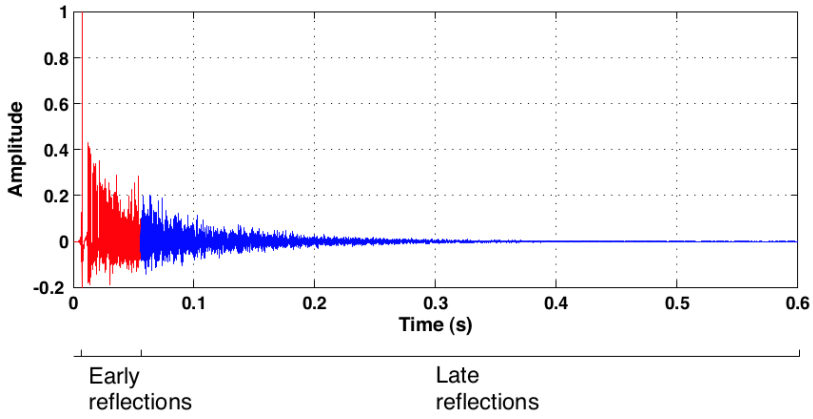


Figura 10 Diferentes partes de la reverberación

Cada parte de la reverberación produce diferentes efectos, las reflexiones más tempranas o primordiales son las que le dan una sensación de que el sonido es más "brillante" o "oscuro", generando un efecto denominado coloración, mientras que las reflexiones más tardías o la cola de la reverberación proporcionan una sensación de lejanía del sonido y es lo que normalmente asociamos a la reverberación. Muchos de los métodos de eliminación de la reverberación solo se centran en esta parte de la reverberación que es más fácil de detectar y eliminar.

Un parámetro muy importante es el tiempo de reverberación o RT_{60} el cual mide el tiempo en el que el volumen de la reverberación baja de los 60db y deja de ser apreciable. Es una propiedad que depende de la estancia donde se realice la grabación, ya que se producirá distintas reverberaciones, debido tanto a la forma de la habitación como la posición del micrófono y las fuentes de sonido generarán diferentes sonidos.

Por ejemplo, mientras que una habitación pequeña y amueblada el sonido rebotará rápidamente en los muebles creando muy poca reverberación, ya que los muebles absorberán la energía de las ondas sonoras, en el interior de una gran iglesia las ondas sonoras rebotarán contra las paredes desnudas creando una reverberación típica de estos espacios que se prolongará durante varios segundos, por lo que tendrá un tiempo de reverberación mayor que la habitación.

Por esta misma razón se añade reverberación digitalmente a grabaciones para dar una sensación inmersiva de espacio en películas y videojuegos. También es el motivo por el cual la reverberación ha sido estudiada en arquitectura para reducir su efecto en espacios cerrados.

Además, existen otras características que pueden hacer variar las reverberaciones[4], como la frecuencia de las vibraciones emitidas por la fuente. Los sonidos más agudos hacen vibrar el aire con frecuencias mayores, lo cual requiere de una mayor energía. Esto produce que los sonidos agudos se atenúen antes que los sonidos graves, por lo que en grandes espacios como en el ejemplo

de la catedral, las reverberaciones serán ecos únicamente de las frecuencias más graves.

Los materiales porosos absorben la energía de la vibración del aire, reduciendo la energía de las ondas que rebotan en ella y por tanto también la reverberación. Estos materiales pueden absorber mejor ciertas vibraciones, eliminando parte de las frecuencias de las reverberaciones. Otra propiedad de los materiales es su resonancia, la cual produce que materiales comiencen a vibrar generando nuevas vibraciones en otras frecuencias.

A parte del tamaño de la sala, la forma de la sala también puede generar diferentes efectos. Mientras que los sonidos agudos suelen ser más direccionales, los sonidos graves de bajas frecuencias se propagan con una mayor amplitud, pueden atravesar objetos más voluminosos y doblar esquinas. Este efecto, que también tiene que ver con la atenuación de los sonidos agudos, es el que produce que, al salir de un local con música, solo oigamos las frecuencias graves.

La distancia entre el receptor y la fuente también es muy importante porque grandes distancias producen que las reverberaciones tengan un volumen similar a la primera onda mezclándolas más uniformemente, mientras que a cortas distancias las reverberaciones tienen un volumen menor, por lo que afectan en menor medida al audio final.

Estos efectos son los causantes de que la eliminación de la reverberación sea un proceso difícil y complicado, en el que se deben de tener en cuenta una gran cantidad de factores. En la Figura 11 podemos ver como el mismo sonido con dos reverberaciones distintas produce dos señales muy diferentes, como podemos ver en su espectrograma de frecuencias.

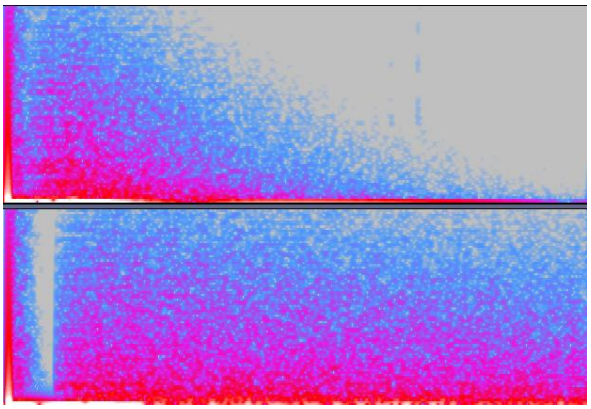


Figura 11 Espectrogramas del mismo sonido ante diferentes reverberaciones.

Para evitar la reverberación es común usar cámaras anecoicas, habitaciones aisladas del ruido con paredes diseñadas para absorber los sonidos e impedir las reflexiones de ondas acústicas. También se puede evitar en menor manera usando acondicionamiento acústico en las paredes de una habitación.

Figura 12 Prueba de sonido en una cámara anecoica

La reverberación afecta a la comprensión del habla reduciendo el entendimiento de las palabras, por lo que es un gran problema en la comunicación. También afecta a la grabación de instrumentos, ya que generalmente se prefiere obtener un sonido limpio y posteriormente añadir una reverberación artificial si se desea. Sin embargo, no siempre se puede disponer de un acondicionamiento acústico, por lo que se han creado algoritmos para eliminar la reverberación de las grabaciones.

La eliminación de la reverberación un proceso muy complejo debido a que diferentes formas de la estancia y materiales de las paredes generan reverberaciones muy distintas, haciendo de este proceso un problema difícil de resolver en su totalidad.

2.2 Métodos de eliminación de la reverberación

La eliminación de la reverberación se ha tratado usando generalmente procesamiento de señales y ha sido en los últimos años con la revolución de la inteligencia artificial y el Deep Learning cuando estas técnicas han comenzado a usarse para resolver este problema.

Mientras que los enfoques anteriores usan complicados modelos matemáticos, el uso de Deep Learning permite resolver el problema únicamente usando pares de sonidos con reverberación y sin ella, solucionando este problema sin la necesidad de un conocimiento extenso en la materia de procesamiento de señales.

Por ello, en esta revisión se realizará un breve repaso de los métodos anteriores, un repaso a como el Deep Learning se ha aplicado al procesamiento de audio y finalmente una revisión a los trabajos realizados con la unión de ambos campos.

2.2.1 Procesamiento de señales aplicado a la eliminación de la reverberación.

En la figura 13, podemos ver un sistema que modela como el sonido de la fuente $s(n)$ se propaga a través de diferentes canales acústicos $H_m(Z)$ y es recibido por

todos los micrófonos del sistema $x_m(n)$.[7] Durante el proceso cada canal puede verse afectado por ruidos $v_m(n)$ que se acoplan a la señal.

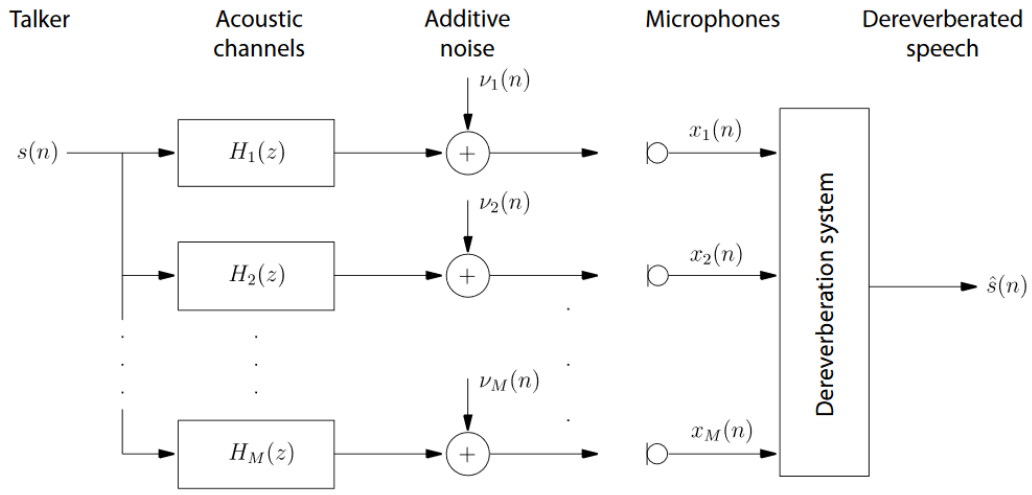


Figura 13 Modelo genérico de reverberación y eliminación de la reverberación

La señal $x_m(n)$ observada por el micrófono además de ruido, puede contener reverberación, la cual podemos expresar como.

$$x_m(n) = \sum_{i=0}^{\infty} h_{m,i}(n)s(n-i)$$

Donde la señal registrada por el micrófono, sin tener en cuenta el ruido, es la superposición de la señal que llega directamente al micrófono y todas sus infinitas reflexiones que también son grabadas por el micrófono. Estas señales se propagan por diferentes canales acústicos $h_{m,i}(n)$ que atenúan y modifican la señal original señal $s(n-i)$.

Posteriormente, aun en la figura 13, un sistema de eliminación de la reverberación recoge todas las señales obtenidas por los micrófonos y genera una señal $\hat{s}(n)$ que se aproxime a la señal original de la fuente $s(n)$

Este sistema que se pretende crear es dependiente de los procesos que le anteceden, la generación de la reverberación, por lo que debe conocer los parámetros que hacen que varíe esta reverberación.

Por esta razón muchas de las técnicas intentan modelar cómo un impulso sonoro de la fuente genera una respuesta en los micrófonos, lo que normalmente se conoce en la literatura cómo Respuesta a Impulsos Acústicos, o AIR del inglés Acoustic Impulse Response.

La Respuesta a Impulsos Acústicos usa las funciones respuesta a impulso con los conocimientos acústicos que muestran cómo se comporta el sonido según la geometría de la estancia, la absorción de las paredes, etc.

Desde este campo se han creado multitud de enfoques que tratan de resolver problemas de diferentes maneras, por lo que una forma de revisarlos es categorizándolos.

1. **Conformación de haces** *Beamforming:*

   Cuando se dispone de múltiples micrófonos se pueden usar para diferenciar las reverberaciones del sonido original según la orientación y la posición de los micrófonos.

   La técnica más sencilla es Delay-And-Sum Beamformer[8], en la que las señales obtenidas de los micrófonos se suman de forma ponderada y con ciertos retardos. La ecuación Estas ponderaciones y retardos se deben especificar manualmente.

   Estas técnicas dan los mejores resultados cuando las reverberaciones se concentran en una región. Sin embargo, la necesidad de usar varios micrófonos hace que no puedan ser usadas en un ámbito general como por ejemplo aplicaciones móviles.

2. **Realce del habla** *Speech Enhancement:*

   Cuando se desea eliminar la reverberación de grabaciones de voces podemos usar ciertas características específicas como el rango de frecuencias en las que se encuentra la voz o picos en las ondas acústicas. Algunas de estas técnicas también hacer uso de varios micrófonos para aumentar su eficiencia.

   Al ser específicos para grabación de voz, no pueden utilizarse para uso general, aunque son muy utilizados debido a la gran utilidad.

3. **Deconvolución Ciega** *Blind deconvolution:*

   Usan los modelos de Respuesta a Impulsos Acústicos AIR para modelar la reverberación de la estancia y poder crear un filtro para invertir el efecto de la reverberación. Estos métodos suelen ser muy complejos ya que o se deben especificar los parámetros de AIR o el algoritmo debe inferirlos a partir de las grabaciones. Suelen usar varios micrófonos para mejorar la eliminación de la reverberación integrando sus respuestas en el modelo AIR.

   A pesar de tener un uso general su complejidad y coste computacional no los hace propicios para muchas aplicaciones.

También se pueden clasificar según como se enfrentan al problema, como diferenció el experto Emanuël A.P. Habets[7]

1. **Cancelación de la Reverberación:**
   Se modela el modelo acústico AIR estimando sus parámetros a partir de las señales obtenidas para reconstruir la señal original. En esta categoría aparecen aquellos que son clasificados como deconvolución ciega.

2. **Supresión de la Reverberación:**
   En esta categoría se presentan los métodos que entienden la reverberación como un proceso aditivo en el que a la señal original se le suma la señal de reverberación. En esta categoría residen los métodos de beamforming también llamados de Spatial Processing y Spectral

Enhancement, aquellos que aprovechan las características de la reverberación para suprimirla, entre los que se encuentran los de realce del habla.

3. **Estimación directa**
   Esta nueva categoría se añadió posteriormente[9], por lo que no aparece en su explicación en el libro Dereverberation[7] y contiene todos aquellos métodos que estiman la señal original tratando el sistema acústico como una incógnita.
   En esta categoría abarca todos los métodos de Deep Learning que mencionaremos más adelante y procesos de predicción lineal.

En conclusión, se puede observar que se han realizado un gran número de técnicas muy diferentes, algunas requiriendo más de un micrófono o saber ciertos parámetros de la estancia donde se realiza la grabación. Muchas de ellas necesitan menos recursos que los métodos de Deep Learning que veremos a continuación, pero carecen de la generalidad que puede conseguirse con el Deep Learning si se entrenan con una red con un diverso conjunto de datos.

2.2.2 Deep Learning aplicado a sonido

Las redes neuronales y en concreto el Deep Learning son una de las herramientas más utilizadas en el ámbito de la inteligencia artificial en la actualidad.

Desde su aparición se han aplicado a diferentes tipos de datos, desde datos tabulares a imágenes con la incorporación de las redes convolucionales, a textos y variables continuas con las redes recurrentes y a sonido y vídeo usando varios de estos métodos. Al mismo tiempo han incrementado las salidas que pueden generar, desde una clasificación o una predicción a redes que pueden generar nuevos datos, como las redes generativas que crean nuevas imágenes.

En las aplicaciones de audio se han usado diferentes tipos de arquitecturas con diferentes tipos de entradas.[10]

Redes Convolucionales

Aunque en algunos primeros proyectos se usaron redes densas para procesar audio como entrada [4], pronto se comenzaron a usar redes convolucionales por sus mejores resultados. La señal de audio puede ser procesada directamente con convoluciones de una dimensión o transformándola a un espectrograma de frecuencia y usando convoluciones bidimensionales.

Redes recurrentes

Mientras que las redes convolucionales bidimensionales solo pueden ver una ventana de los datos, las redes recurrentes pueden aceptar audios de diferentes tamaños. Las redes recurrentes además de tener salidas y entradas tienen conexiones a sí mismas que afectarán a las siguientes iteraciones. Este

mecanismo confiere a la red una especie de memoria sobre las iteraciones pasadas.

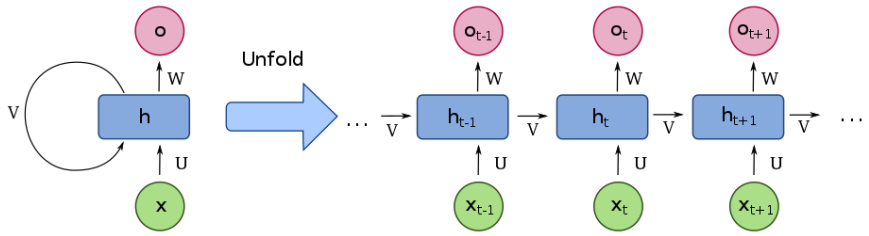


Figura 14 Esquema de una red recurrente básica, donde la salida de la conexión v se vuelve a introducir a la red en la siguiente iteración.

Arquitectura LSTM

Una de las arquitecturas más utilizadas de este tipo son las LSTM, cuyo acrónimo proviene de Memoria a Corto y Largo Plazo. En la Figura 15 podemos observar una representación de la estructura interna de una red LSTM.

En estas redes la salida de la red h_t se propaga a la siguiente iteración de la secuencia por la entrada h_{t+1}. Esta conexión representa la memoria a corto plazo.

Las conexiones c_t representan la memoria a largo plazo, la cual solo puede ser modificada a través de estructuras llamadas puertas y que finalmente producirá la salida h_t y la salida de la red.

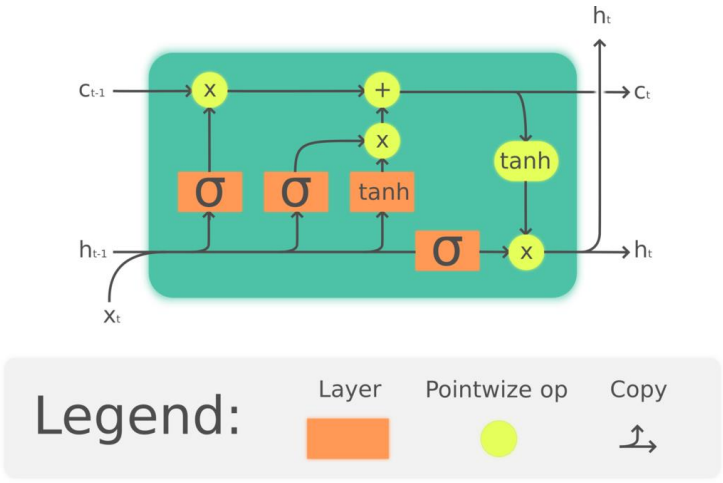


Figura 15 Esquema de una red LSTM

El estado de memoria a corto plazo se suele denominar hidden state, mientras que el de largo plazo, cell state.

Al comienzo de la iteración se concatena la entrada de la red x con la memoria a corto plazo h_{t-1} en la siguiente memoria a corto plazo.

La primera puerta que se aplica permite a la red descartar información de la memoria a largo plazo, y en el dibujo se encuentra en la primera conexión de la izquierda. Por esta razón se le da el nombre de *forget gate*. Primero se aplica una capa con una función de activación sigmoide representada por el símbolo σ. Gracias a esta función de activación los valores devueltos estarán en el rango

de 0 a 1. Al multiplicar la conexión de la memoria a largo plazo por este rango valores los filtramos, permitiendo a la red decidir que debe permanecer y que anular convirtiéndolo en 0. Este mecanismo es usado en las demás puertas.

Esta puerta se puede describir como:

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

Donde σ representa la activación sigmoide de la capa, W_f el valor de sus pesos y b_f su bias.

La siguiente puerta, a la mitad del dibujo, añade información a la memoria a largo plazo, por lo que se llama *input gate*. Los valores de la memoria a corto plazo son pasados por una capa con una función de activación de tangente hiperbólica, que devuelve un rango de -1 a 1 y simplemente este resultado se suma a la memoria a largo plazo.

$$\hat{C}_t = \tanh(W_C[h_{t-1}, x_t] + b_C)$$

En esta ecuación W_c son los pesos de la capa con activación de tangente hiperbólica e b_c su bias.

Antes de que suceda la suma el resultado se filtra con el mismo mecanismo que en la puerta anterior para refinar que se añade a la memoria.

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i) * \hat{C}_t$$

En esta última ecuación obtenemos la salida de la puerta input donde W_i e b_i son los pesos y bias de la capa con activación sigmoide que multiplica a la capa con activación tangente hiperbólica.

La memoria a largo plazo o estado de celda es actualizada por ambas puertas

$$C_t = f_t \cdot C_{t-1} + i_t$$

Finalmente, los valores de la memoria a largo plazo son pasados a la siguiente iteración y una copia de ellos se le aplica una función tangente hiperbólica para que tengan un rango de -1 a 1. Una vez hecho esto se filtran con el mismo método y el resultado de esta operación se convierte en la salida de la red y la entrada de la memoria a corto plazo de la siguiente iteración.

$$h_t = \sigma(W_o[h_{t-1}, x_t] + b_o) * \tanh(C_t)$$

Arquitectura GRU

Una mejora de esta arquitectura es la red GRU, la cual contiene menos parámetros entrenables y presenta rendimientos similares.

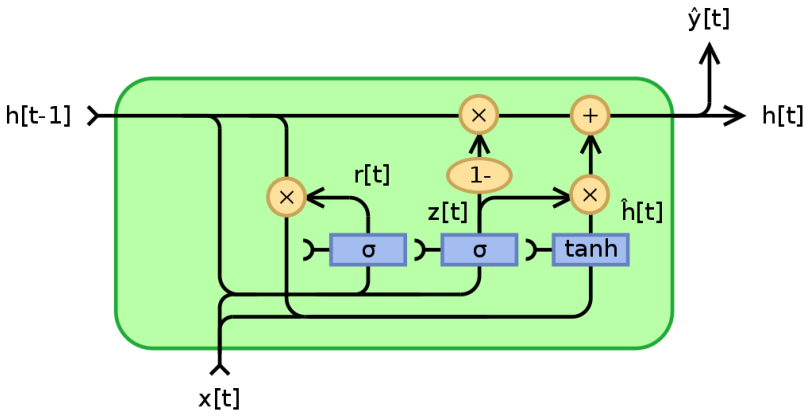


Figura 16 Esquema de una red GRU

Esta red únicamente tiene una conexión consigo misma. Al igual que en las LSTM las capas con función de activación sigmoide filtran la información, pero en este caso la entrada de estas capas es la concatenación de la conexión recurrente y la entrada de la red.

Este filtrado se realiza sobre la conexión recurrente h_{t-1} usando la propia conexión recurrente y la entrada.

$$r_t = \sigma(W_r[h_{t-1}, x_t] + b_r)$$

Luego vuelve a ser concatenado con la entrada x_t pasado a una capa con función de activación tangente hiperbólica.

$$\hat{h}_t = \tanh\left(W_{\hat{h}}[r_t \cdot h_{t-1}, x_t] + b_{\hat{h}}\right)$$

Posteriormente la salida de esta capa se le aplica otro filtrado calculado como:

$$z_t = \sigma(W_z[h_{t-1}, x_t] + b_z)$$

Finalmente será sumado a lo que resultará la salida de la red y la conexión recurrente, en algo parecido a lo que antes era el *input gate*.

$$h_t = (1 - z_t) \cdot h_{t-1} + z_t \cdot \hat{h}_t$$

Para ahorrar parámetros se realiza el filtrado de la *forget gate* con el opuesto de este último filtrado.

A través de todos estos mecanismos, tanto la arquitectura LSTM como GRU pueden retener información, procesarla y eliminarla.

Las redes recurrentes pueden ser bidireccionales, para ello se crean dos capas de redes recurrentes y a cada una de ellas se le muestra la secuencia en orden

o en orden inverso. Finalmente se suman sus dos salidas. De esta forma se crean las redes BiLSTM y BiGRU.

Se han usado tanto redes recurrentes unidireccionales como bidireccionales para procesar señales de audio, generalmente usando el modelo LSTM. También se han desarrollado versiones de LSTM para procesar espectrogramas, las Frequency LSTM y las Frecuency and Time LSTM. También pueden procesar la salida de redes convolucionales para crear Redes Convolucionales Recurrentes o CRNN.

Debido a la capacidad de las redes recurrentes de trabajar con símbolos como texto, también son usadas para procesar archivos MIDI, en donde en vez de usar la señal de audio de una canción se usan las partituras como símbolos. Existen múltiples formas de codificar estas partituras, cada una obteniendo diferentes resultados.

Secuencia a Secuencia

Estos métodos generan una secuencia a partir de otra secuencia de entrada, permitiendo hacer conversiones como la eliminación de la reverberación que se pretende hacer en este trabajo. A partir de audios con reverberación estas redes pueden generar audios sin reverberación. Estas redes suelen tener la estructura base de un autoencoder.

Un autoencoder es una red neuronal que a partir de unos datos de entrada intenta generar otros de salida iguales o con bastante relación. La red está compuesta por dos partes, una serie de capas con cada vez menos neuronas por cada llamada codificador y luego una capa simétrica con cada vez más neuronas por cada llamada decodificador.

El propósito de esta estructura es que la información de la entrada tenga que abstraerse según atraviesa el codificador, al tener que representarse con menos neuronas y posteriormente reconstruirse en el decodificador. Justo entre estas dos partes, en la mitad se alcanza la mayor abstracción y esto provee de un gran número de utilidades, como separar el codificador y el decodificador para crear sistemas de compresión.

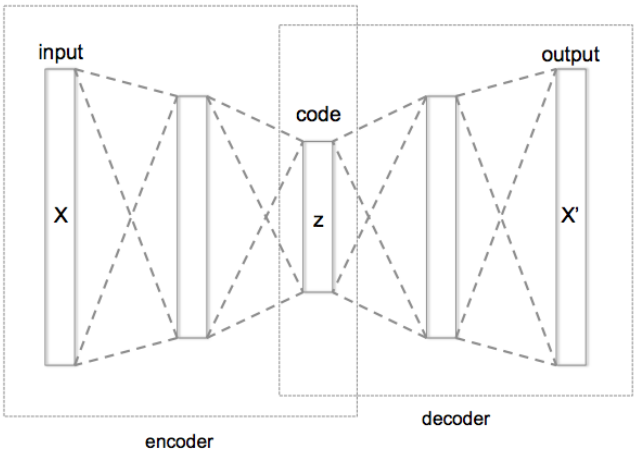


Figura 17 Estructura de un autoencoder

Una U-Net es un autoencoder en donde la salida de cada capa, además de con la siguiente, está conectada con su simétrica, lo que permite a la red abstraer ciertos parámetros y dejar pasar la información. Estas conexiones se denominan *skip connections*. Las U-Nets suelen estar formadas por capas convolucionales.

Suelen tener un mejor rendimiento que el autoencoder cuando se desea convertir los datos de entrada en otros diferentes. Las U-Net usan redes convolucionales en sus capas.

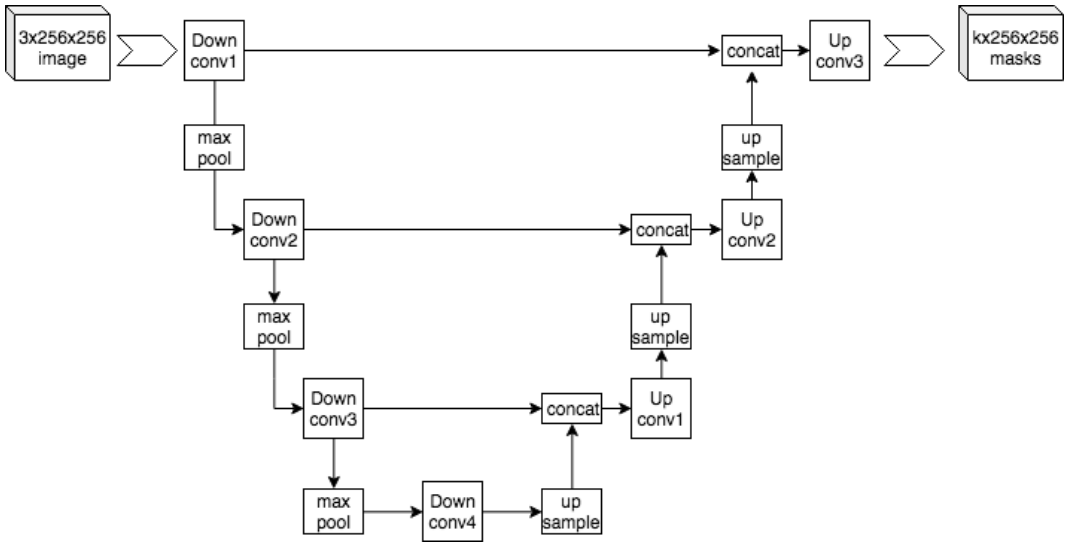


Figura 18 Estructura de una U-Net que produce imágenes de 256x256

Esta transformación de secuencia a secuencia puede tener muchas utilidades, una de ellas es la separación de una canción en las diferentes pistas que la forman, como es el caso de la red Demucs[1] que usa un U-Net con una red recurrente BiLSTM y redes convolucionales unidimensionales para procesar una señal de audio de una canción y separarla.

También podemos incluir en esta categoría aquellas redes recurrentes que generen nuevas secuencias a partir de la entrada.

Redes Generativas Antagónicas

Más conocidas por su acrónimo inglés GAN (Generative Adversarial Networks), las redes generativas antagónicas son arquitecturas neuronales famosas por su generación de imágenes realistas. Estas redes permiten generar nuevas imágenes a partir de datos de entrenamientos que intentará imitar.

Sin embargo, estas redes también pueden aplicarse en el ámbito del audio, usando como entrada espectrogramas de audio, partituras o la señal de audio. En este último caso la red no procesará una imagen sino un vector con la señal de audio, por lo que se deberán realizar ajustes en el modelo, aunque la estructura general siga siendo la misma.

Una GAN está compuesta de dos redes, una red generativa que crea las imágenes y un discriminador que evalúa el generador. La Figura 19 muestra una representación de esta arquitectura.

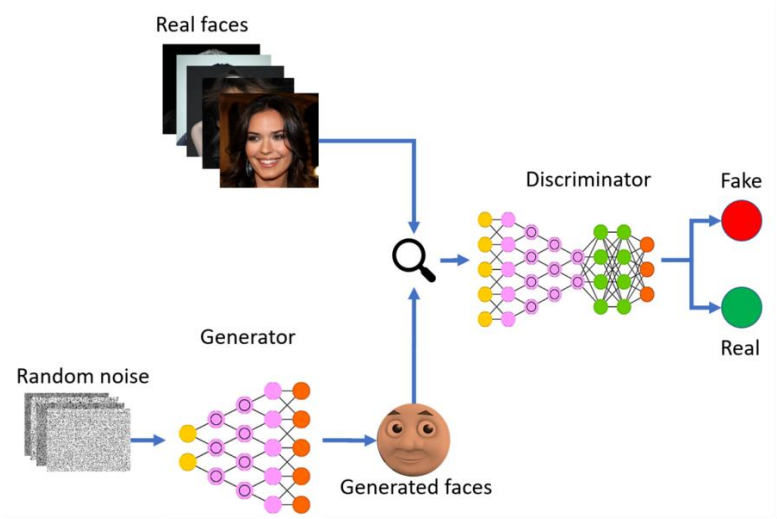


Figura 19 Representación de una Red Generativa Antagónica que genera caras

Primero el generador crea nuevas imágenes usando como entrada números aleatorios. Estas imágenes generadas son presentadas junto a las del conjunto de datos de entrenamiento al discriminador. La tarea del discriminador es la de diferenciar que imágenes han sido creadas por el generador.

Cuando el discriminador falla y clasifica mal una imagen, el error se retropopaga por su red como en una red neuronal normal. Pero cuando el discriminador acierta, es la red generativa la que recibe el error y aprende.

De esta forma ambas redes compiten y aprenden, la red generativa cada vez crea imágenes más similares al conjunto de datos de entrenamiento para engañar al discriminador y el discriminador aprende a diferenciar cada vez mejor las imágenes del generador de las reales.

Finalmente, una vez entrenada la red, generalmente se usa únicamente el generador para crear nuevas imágenes muy similares a las del conjunto de datos.

Un ejemplo de una red GAN aplicada a audio, en especial la generación de música es MuseGAN[11] en el que se usan redes convolucionales sobre

partituras MIDI de piano roll, donde las filas de la imagen representan notas y las columnas el tiempo.

Conditional Generative Adversarial Networks
El sistema de competición de las GAN también se puede usado para convertir unas imágenes en otras imágenes como realizan las redes de secuencia a secuencia. Estas arquitecturas se denominan conditional GAN o cGAN ya que las imágenes generadas están condicionadas por otras.

La estructura generador-discriminador permanece igual, pero en este caso la red generativa es del tipo secuencia a secuencia y recibe como entrada una imagen.

Al igual que en las redes que generan secuencia a secuencia, serán necesarios pares de imágenes. Tras realizar la generación de imágenes, los pares de imágenes generados se mezclarán con los reales. El discriminador recibirá el par de imágenes y los clasificará en reales o generados por el generador, al igual que en el caso de las GAN. Realizando estos cambios el generador será dependiente de las imágenes de entrada.

Un ejemplo de cGan son las redes Pix2Pix[12] las cuales permiten realizar conversiones de fotos diurnas a nocturnas o mapas a fotos de satélite. Esta arquitectura usa como generador una red U-Net.

2.2.3 Deep Learning aplicado a eliminación de la reverberación

Desde el 2000 se sabía que las redes neuronales tendrían una gran utilidad en el realce del habla[13] y tras obtener buenos resultados en reducción de ruido aparecieron redes neuronales para el realce de voz en 2014[14]. Posteriormente se comenzaron a realizar los primeros trabajos especializados en tratar la reverberación con redes neuronales como Han et Al.[15] donde procesaba un mapa espectral de la frecuencia con una red profunda con 3 capas ocultas.

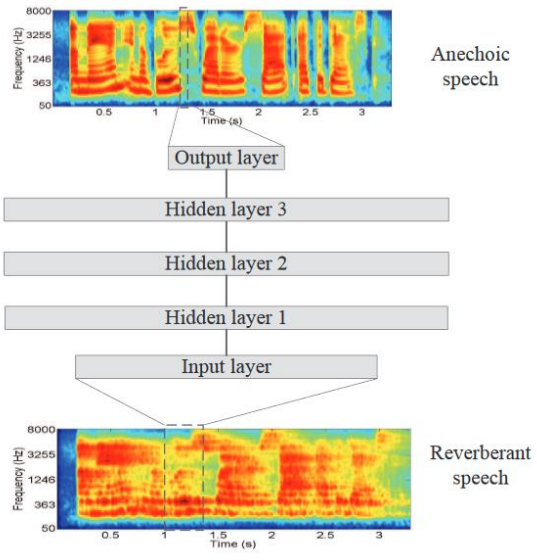


Figura 20 Red neuronal densa propuesta por Han et Al en 2014

También en 2013 se creó la competición Dereverberation Challenge[16] con el objetivo de progresar las investigaciones en esta cuestión y proveyó un dataset para entrenar a los algoritmos y realizar benchmarks

Figura 21 Logo de la competición Reverb Challenge

Este dataset contiene pares de voces con y sin reverberación grabados desde 1 a 8 micrófonos. La competición concluyó en 2014 con la participación de 27 papers con 49 sistemas, donde se presentaron los ganadores en un workshop. Posteriormente se ha seguido usando este dataset como benchmark

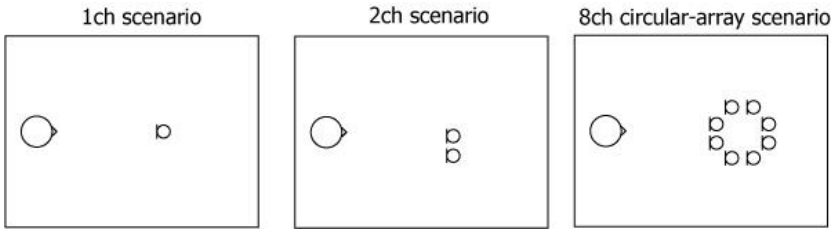


Figura 22 Tipos de grabaciones en el dataset de Dereverberation Chanllenge

También se han realizado más competiciones con menor repercusión como la Acoustic Characterization of Environments (ACE) Challenge[17] o la Automatic Speech recognition In Reverberant Environments (ASpIRE) Challenge[18].

Posteriormente se han realizado numerosos trabajos donde se han utilizado diferentes arquitecturas de redes neuronales para solventar este problema usando tanto los nuevos avances en Deep Learning como los conocimientos previos en procesamiento de señales.

Otros estudios con cGAN han obteniendo buenos resultados[2], [3], [19] y se han comparándo con otros trabajos que usaban DNN, CNN y LSTM [2].

Generalmente el uso de GANs se justifica debido a que al usar una red discriminadora no hace falta usar métricas como MSE o L1, que pueden producir problemas en el entrenamiento[20]. El aprendizaje se realiza debido a que el discriminador aprende que pares de audio con y sin reverberación son los reales.

También se han propuesto medidas intermedias como tomar como métrica de error la suma ponderada del MSE y el error del discriminador.[19]

Erns et Al.[3] realizaron un estudio donde comparaban trabajos anteriores con dos modelos propuestos de Deep Learning, que usaban un espectrograma como entrada. Estos eran una red U-Net y una red Pix2Pix. Ambas soluciones daban buenos resultados, mejorando el estado del arte en ciertos datasets, siendo la U-Net en promedio mejor cuando usaba kernels asimétricos en las convoluciones, de mayor tamaño en el eje del tiempo que en el de la frecuencia.

También se menciona que en la arquitectura GAN el discriminador no conseguía progresar, por lo que el entrenamiento se estancaba, tras un cierto número de iteraciones.

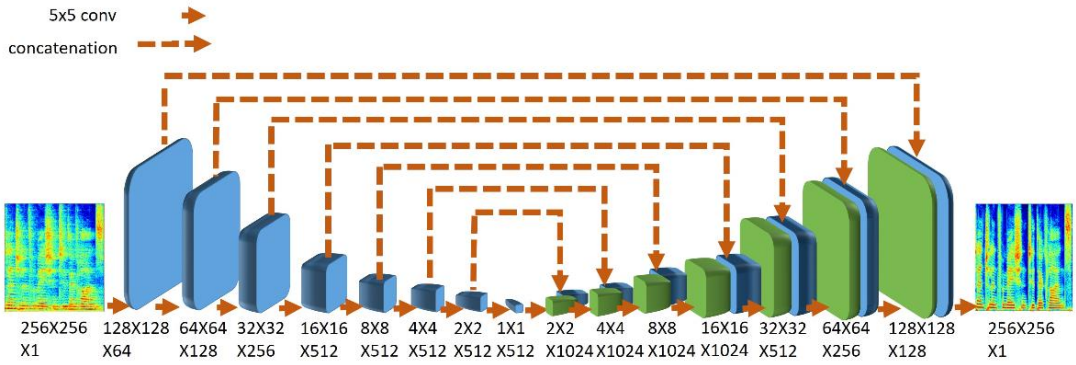


Figura 23 U-net usada por Erns et Al

Otros trabajos han optado por crear arquitecturas específicas para resolver el problema, como Chenxing Li Et Al.[19] donde utilizan una cGAN utilizando los espectrogramas, donde el generador está compuesto por 5 capas convolucionales, 2 capas BiLSTM y una densa.

Esta red genera una máscara que se multiplica con la señal original. El discriminador contiene 2 capas BiLSTM y una densa.

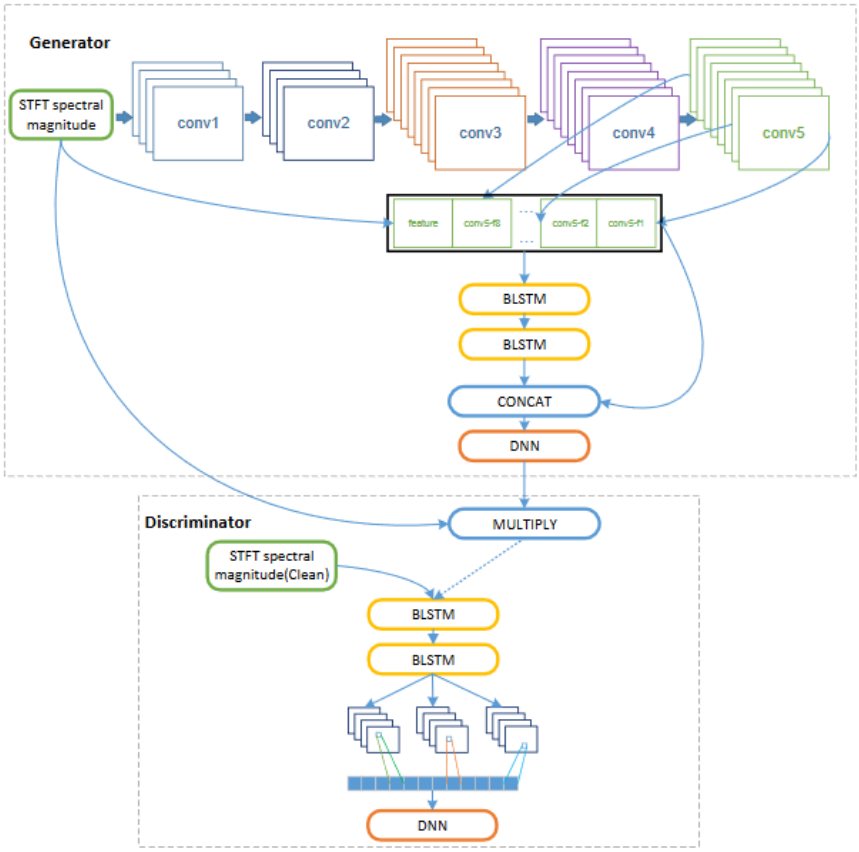


Figura 24 Modelo GAN creado por Chenxing Li Et Al.

Usando redes BiLSTM Cuchang Fan Et Al.[21] han creado un modelo que tras eliminar el ruido como habían realizado en proyectos anteriores, ahora usa las mismas redes BiLSTM para crear una máscara que multiplicada a la señal de entrada produzca una señal sin reverberación.

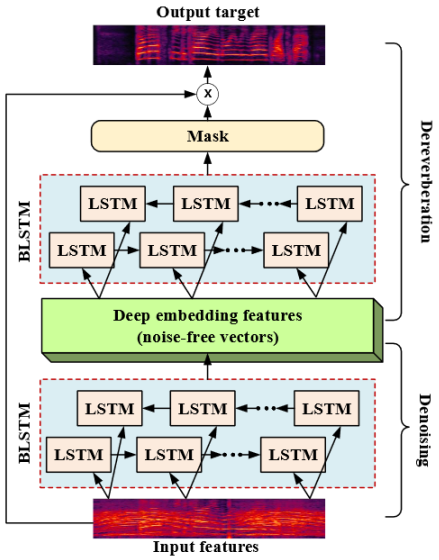


Figura 25 Arquitectura de Cuchang Fan Et Al. formada por capas BLSTM

También se han utilizado redes neuronales para mezclar las diferentes respuestas de diferentes modelos, como es el caso del sistema IDEA[22], o incluyendo una red neuronal al final de una concatenación de métodos de eliminación de la reverberación para refinar el resultado.[14]

Se han usado las redes neuronales[23] para reconocimiento por voz, usando autoencoders y en conjunto con otras técnicas como MFCC y X-Vector Extractor.

En otros problemas relacionados, como la localización de una fuente de sonido usando pares de micrófonos también se han usado redes neuronales[24]

Se puede comprobar que hay una gran diferencia entre las arquitecturas que se han presentado, esto es debido a la variedad de formas que existen para tratar el sonido y que no hay una forma específica que de mejores resultados.

2.3 Demucs

Demucs[1] es la red neuronal artificial creada por Facebook con el objetivo de separar una canción en sus diferentes fuentes, la batería, el bajo, la voz y el resto de los acompañamientos. Este trabajo de noviembre de 2019 consigue buenos resultados con respecto el estado del arte, superándolo en la mayoría de los casos. Esta red cuyo código se encuentra accesible en GitHub, está programada en Pytorch, ya que pertenece a Facebook.

La red esta tiene está inspirada en Wave-U-Net[25], siendo muy similar a las arquitecturas U-Net, presentando primero varias capas que codificadoras que comprimen la información y luego capas decodificadoras que vuelven a

recuperarla como vemos en la figura 26. También posee las *skip connections* características de las U-Net que enlazan el codificador con el decodificador.

Al igual que la Wave-U-Net, Demucs usa directamente la onda de sonido tanto para la entrada como para la salida generada. Podemos observar en la figura 26 como en la entrada se introduce la onda de una canción en Estéreo, la cual será codificada en 2 vectores, uno para el canal izquierdo y otro para el derecho, y comenzará a procesarse en las capas de codificación o encoders. Cuando llegue al centro del modelo, donde se encuentra el estado latente, una red lstm lo mezclará con los espacios latentes anteriores.

Finalmente, a partir de este espacio latente la secuencia de decodificadores generará una matriz donde en cada par de filas se encuentren dos vectores con una de las pistas en estéreo que se desea separar.

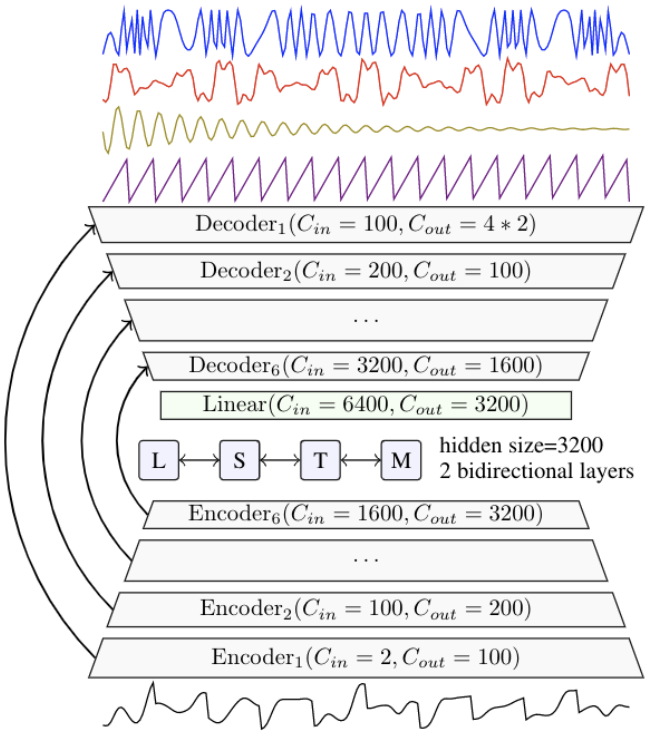


Figura 26 Arquitectura Demucs (La entrada se sitúa en la parte inferior del dibujo)

El número de pistas y canales de audio se debe especificar antes de crear el modelo, pero se puede variar según se necesite, en este caso 4 pares para cada pista de la canción. El modelo entrenado por Facebook usa 6 capas de codificación y 6 de decodificación, ya que deben de ser simétricas. Este parámetro también es modificable.

En general la mayoría de los hiperparámetros de la arquitectura de la red son modificables, permitiendo que se generé diferentes modelos sin muchas complicaciones.

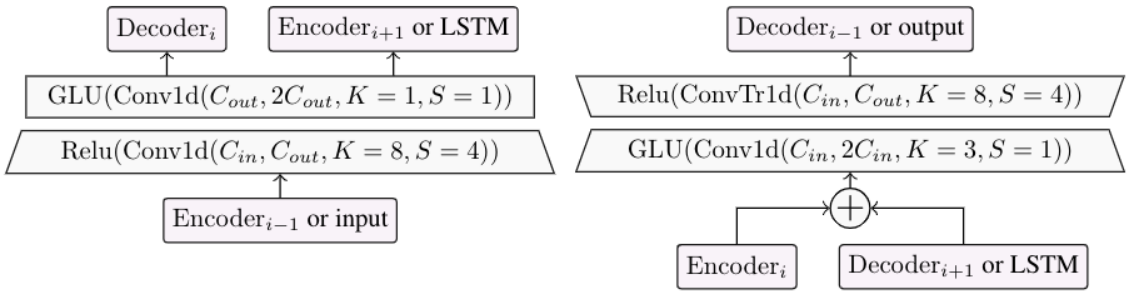


Figura 27 Arquitecturas de los codificadores y decodificadores de Demucs

Cómo podemos ver en la figura 27, las capas de codificación y decodificación están formadas por capas más simples. En la codificación se realiza una convolución unidimensional con un tamaño de kernel 8 y un stride de 4, seguida de una capa de activación Relu. Posteriormente se realiza la misma operación, pero en este caso se usa una capa de activación GLU y se generan dos salidas, una para la siguiente capa y otra para la capa en simétrico del decodificador.

La capa de activación GLU o Gated Linear Unit divide a la mitad la matriz de entrada y realiza el producto elemento a elemento de la primera parte por el sigmoide de la segunda, como podemos ver en la siguiente ecuación.

$$GLU(a, b) = a \otimes \sigma(b)$$

Esta capa permite dar más expresividad a la red y le ayuda a enmascarar el audio, necesario para separar pistas. Esta capa solo se encuentra en Pytorch, aunque se puede implementar fácilmente en otras librerías de Deep Learning.

La capa convolucional que precede a la capa GLU tiene un kernel de tamaño 1, añadiendo profundidad y más canales de forma muy poco costosa.

Los decodificadores tienen la misma arquitectura, pero con estos 2 bloques al revés, primero se concatena la salida de la capa anterior con la salida del codificador en simétrico, luego prosigue con una convolucional y una capa de activación GLU y posteriormente realiza otra convolucional con una activación Relu.

Los canales iniciales de la capa convolucional son de 100 en el caso de la versión original y de 64 en la versión light del modelo publicado posteriormente. La gran cantidad de canales fueron uno de los parámetros que más ayudaron a conseguir buenos resultados.

En el apartado de anexos es posible ver la estructura de Demucs que se obtiene al simplemente imprimir el modelo por pantalla.

El modelo se entrenó usando el repositorio MusDB, el cual contiene 150 canciones separadas por pistas. Estas pistas se dividieron en segmentos de 11 segundos de longitud y agruparon en lotes de 16 para el entrenamiento. Se aumentaron los datos con diferentes técnicas, como intercambiar los canales de estero, invertir la onda, desplazar temporalmente las pistas o incluso intercambiar pistas entre canciones de un mismo lote.

Se usó Adam como optimizador con una tasa de aprendizaje de 3e-4 y la función de coste L1 en vez de MSE, aunque este último cambio no conllevo una gran mejora. Se tardaron 240 iteraciones en entrenar sus 648.548.208 parámetros entrenables.

3 Desarrollo

3.1 Desarrollo cronológico

3.1.1 Preparación inicial

La idea inicial de este trabajo final surgió en una reunión con el tutor, a partir de la cual se inició un proceso de investigación sobre la temática de la eliminación de la reverberación con el objetivo de entender qué progresos se habían realizado en este ámbito, tener un conocimiento mayor del problema y no realizar un trabajo que se hubiese realizado anteriormente. Todos estos conocimientos se ven reflejados en el apartado de estado del arte de esta memoria.

Desde un comienzo se tuvo la intuición de usar la red Demucs debido a sus resultados en separación de pistas de música. Esta intuición, que es la misma que sostiene la supresión de ruido, se apoya en que la reverberación puede modelarse como un proceso aditivo en el cual a la señal original se le suma la señal de reverberación. Demucs permite separar pistas de una señal original, en este caso podríamos modificarla para separar el audio original de su reverberación.

Además, en el trabajo de Erns et Al[3] se usaba una arquitectura U-Net produciendo los mejores resultados con respecto a otras redes con las que se comparaba, incluyendo una cGan. Como Demucs es una U-Net modificada esta era una buena señal.

Al mismo tiempo se comenzó a probar la red Demucs y a inspeccionar su código en GitHub. A partir del paper de Demucs[1] y su implementación en GitHub[26] se realizó el apartado donde se explica cómo funciona la arquitectura. Este repositorio también contaba con muchos más archivos los cuales se dedicaban a procesar el audio, hacer aumentación de datos, argumentos para lanzar el programa, a entrenar el modelo de forma distribuida…

Uno de los mayores desafíos al utilizar esta red era el de entrenar una red de más de 500 millones de parámetros que había sido entrenada usando 32 Gigabytes de memoria RAM y más de 16 de GPU.

En este caso solo vamos a generar 1 pista, el audio sin reverberación, en vez de 4 pistas y además usaremos un único canal de audio en vez de dos como en estéreo, ya que la mayoría de los micrófonos graban en mono. Podríamos pensar que con estas modificaciones el número de parámetros de la red descendería enormemente, pero solo nos libramos de 6.407 parámetros de 648.548.208 parámetros.

Por esta razón utilizamos la arquitectura light de Demucs, con la que el tamaño se reduce a 265.675.393 parámetros, un 40% del tamaño original de Demucs.

Aún con esta reducción era imposible entrenar la red en local, pero sí en el servicio de Google Colab.

Google Colab es un servicio de Google que permite ejecutar y compartir cuadernos Jupyter escritos en Python de manera gratuita. Esta desarrollado principalmente para ejecutar algoritmos de Machine Learning, incluyendo aquellos que requieran de una gran potencia computacional como es el caso del Deep Learning. Además, Google Colab permite usar tanto GPU, lo cual acelera en gran medida el entrenamiento de las redes neuronales. Estas características lo hacían ideal para entrenar la red en la nueva tarea de eliminar la reverberación.

Para importar el resto del código de Demucs al Notebook de Colab simplemente este se clonaba de un repositorio de GitHub. El código de Demucs tuvo que ser modificado para poder accederse desde el Notebook.

Para entrenar a la red es necesario tener un conjunto de datos que cuente con pares de audios sin ninguna reverberación y con reverberación. En un primer momento se decidió usar los datos de entrenamiento de Reverb Challenge[16] y así poder comparar la red con otras que fueron evaluadas con el mismo dataset, pero fue imposible conseguir descargar los datos siguiendo las instrucciones de la web y tampoco se pudo contactar con ellos.

Finalmente se decidió usar 2 conjuntos de datos de grabaciones en habitaciones anecoicas, de los que posteriormente se les añadiría artificialmente una reverberación.

El primero de los conjuntos de datos es el Speech Database de Telecommunications & Signal Processing Laboratory Multimedia Signal Processing[27], el cual contiene grabaciones cortas en ingles de hombres, mujeres y niños en una cámara anecoica con una buena calidad de audio.

El otro conjunto de datos complementa a este último añadiendo grabaciones de instrumentos en una cámara anecoica, proporcionado por OpenAIR[28]. Estas grabaciones contienen canciones enteras y notas sueltas producidas por los instrumentos.

En primera instancia se consideró usar el software Audacity para generar los pares de todos estos audios con reverberación. Para ello se probaron 4 configuraciones de reverberación, voz 2, habitación mediana, habitación grande e iglesia. Además, se convirtieron todos los audios a mono, se normalizo su profundidad de bits a 32 y se exportaron en formato WAV.

Más tarde se ideó otro sistema para aplicar directamente la reverberación desde código sin usar Audacity.

Se uso el servicio de Google Drive para almacenar el dataset, debido a su fácil integración con Google Colab que permite usando un simple código montar en segundos la carpeta en el directorio de Colab y acceder fácilmente a los archivos desde el Notebook de entrenamiento.

3.1.2 Implementación del código de entrenamiento

Tras varias modificaciones se implementó una versión de prueba del código de entrenamiento, usando las clases del repositorio de Demucs para leer un único

archivo de audio, con el objetivo de probar si el código realizado era capaz de entrenar a la red.

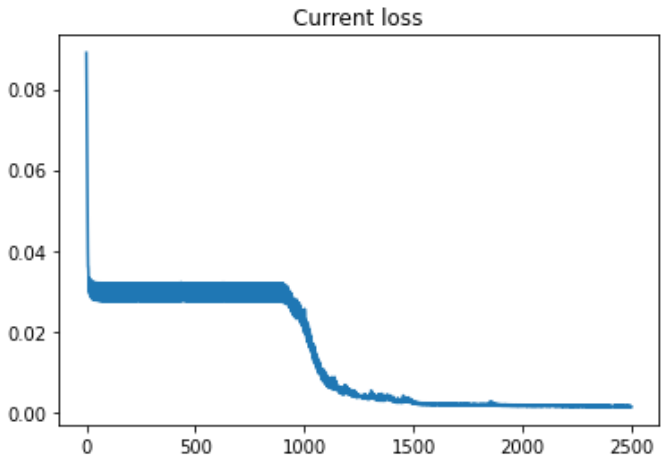


Figura 28 Gráfica de error L1 de la primera prueba donde se sobreentreno a la red

Después de 2500 iteraciones la red consiguió aprenderse el audio sin reverberación, demostrando que el código de entrenamiento funcionaba y que no había errores en la codificación del sonido. A pesar del bajo error obtenido, esta red no podía eliminar la reverberación, lo cual quedaba claro al introducir otro audio y escuchar su respuesta. La red se había sobreajustado al único dato de entrada, probando que el código de entrenamiento funcionaba.

Como la red no tiene ningún componente capa densa, no hay ningún tamaño fijo de entrada de datos. La convolución se va aplicando a todo el audio y al llegar al final se aplica un padding si el audio restante no ocupa el tamaño del kernel.

Sin embargo, para usar lotes los audios deben de tener la misma longitud, por lo que deben ser recortados. En el código de Demucs se recortan los audios usados en el entrenamiento en una longitud de aproximadamente 10 segundos.

3.1.3 Implementación de Augmented Reverbs

Para simplificar el uso de los Datasets se creó la clase Augmented Reverbs. En el código de Demucs existen clases que realizan funciones similares, pero se implementó una clase propia para controlar todas las características especiales del dataset de reverberación.

Esta clase permite acceder a los audios con un índice como si fuese un vector de Python. Para ello almacena las direcciones de los archivos y permitía a Pytorch cargarlos en el batch leyéndolos de disco.

Este sistema acabo siendo lento, ya que conllevaba leer de disco en cada batch y comenzó a generar errores cuando varios hilos de Pytorch intentaban leer el mismo archivo, por lo que se acabó optando por cargar todos los audios en memoria RAM ya que el entorno de Colab provee la suficiente.

Sin embargo, la lectura de todos los audios es una operación muy lenta debido a la decodificación de los archivos de audio, generalmente la lectura de todos

los audios tardaba alrededor de 25 minutos. Para evitar el coste de la decodificación simplemente se guardaron los conjuntos de datos almacenados en las clases Augmented Reverbs en archivos pickle, reduciendo el tiempo de carga a 15 segundos.

Otra función de la clase Augmented Reverbs es la de realizar aumentación de datos, la cual se realiza de diferentes maneras.

Cada vez que se pide un audio a través de un índice se retorna se retorna el audio indicado del conjunto de datos recortado a la duración de 10 segundos. Si el audio es dura menos de 10 segundos, se le añaden silencios por delante y por detrás con longitudes aleatorias. En cambio, si es mayor de 10 segundos, se escoge una parte aleatoriamente de 10 segundos de duración.

Luego se le superpone un número especificado de audios por encima de este usando el mismo método, de forma que cada audio que escucha la red es diferente.

Una vez realizado está aumentación de datos, usando la librería pysndfx se le añade una reverberación al audio creado. La reverberación generada por la librería depende de cuatro parámetros, la cantidad de reverberación, el tamaño de la habitación, el retardo de la reverberación y la atenuación a las frecuencias altas que se produce. Estos parámetros también pueden ser escogidos aleatoriamente en el rango especificado.

Al generar la reverberación directamente desde el código se prescindió de utilizar Audacity, permitiendo dar más flexibilidad a las pruebas y evitando el tener que rehacer el conjunto de datos y subirlo a Google Drive cada vez que se necesitase un cambio o descubriese un error.

Finalmente, los audios se normalizan restándole su media μ_x y dividiéndolo entre su desviación estándar σ_x como aparece en la siguiente formula.

$$normalización(x) = \frac{x - \mu_x}{\sigma_x}$$

Los datasets de entrenamiento, validación y test son generados usando tres instancias de la clase Augmented Reverbs. En la figura 30 se puede observar que la proporción de estos es de un 80% para el entrenamiento, 15% para validación y 5% de prueba. En esta figura se muestra la cantidad de audios de los conjuntos de datos previamente mencionados y uno posterior que se añadió más tarde, como se cuenta posteriormente.

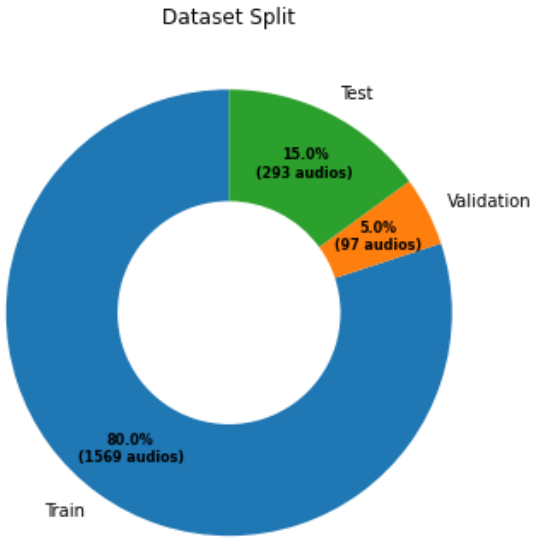


Figura 29 Proporción del conjunto de datos con los 3 conjuntos de datos

3.1.4 Realización de pruebas

En la figura 29 podemos observar una gráfica generada usando todos los audios sin batchs, obteniendo como respuesta un audio muy parecido al de entrada, pero aún con reverberación.

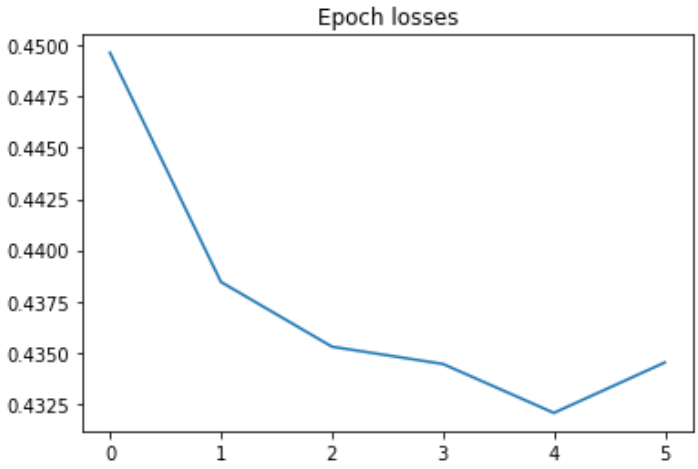


Figura 30: Gráfica del error L1 usando todos los audios para el entrenamiento

Tras probar la red con estos cambios se observó tras 50 épocas que el entrenamiento no superaba el 0.43 de error en entrenamiento y 0.32 en validación y que el audio generado era muy ruidoso. Se realizaron varias pruebas con diferentes ratios de aprendizajes sin obtener ninguna mejoría.

Para mejorar la red se cambió las redes LSTM del modelo DEMUCS por redes GRU y no se observó ninguna mejora en el error, pero sí un incremento en la velocidad de entrenamiento, debido a que las redes GRU cuentan con menos parámetros, en concreto hubo una reducción de 41.959.424 parámetros.

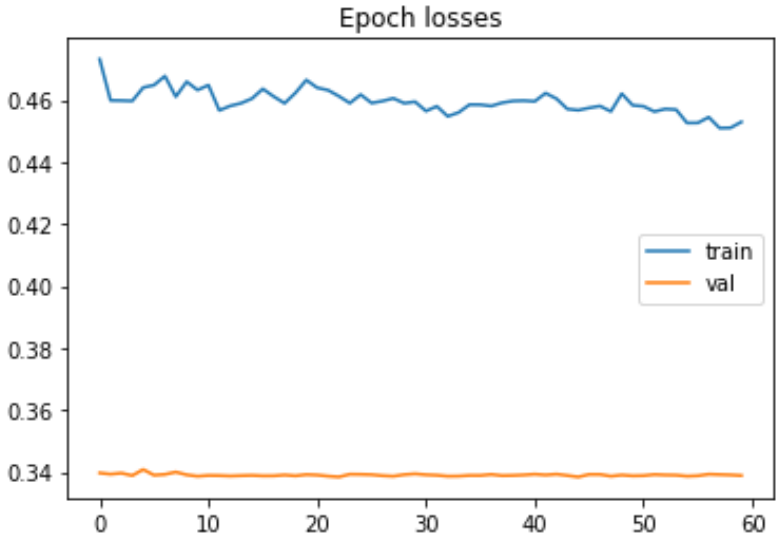


Figura 31 Gráfica del error L1 usando GRU y batch size 8

Posteriormente se implementó de nuevo el sistema de lotes de Demucs. Usando un batch size de 2 se obtuvieron mejores resultados, pero en el audio resultante seguía conteniendo gran parte de la reverberación.

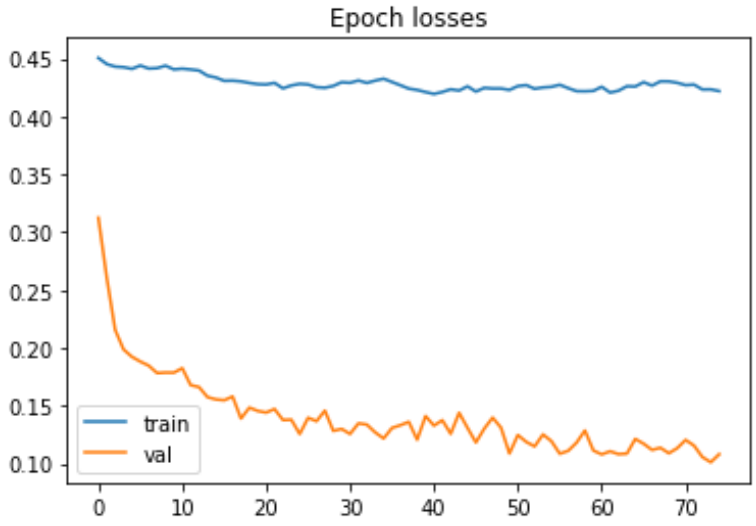


Figura 32 Gráfica de error L1 con batch size 2

Mientras que los lotes de 4 y de 8 tienen rendimientos similares, al usar un batch size de 16 el uso de GPU superaba los 15 Gigabytes y Colab no era capaz de entrenar la red.

Se puede observar en las figuras 31 y 32 que el conjunto de datos de validación produce muchos menos errores que el de entrenamiento, cuando generalmente debería ser al revés. Esto es debido a la decisión de no usar aumentación de datos sobre los datos de validación para que siempre se evalúen sobre el mismo dato.

Pero al quitar la aumentación de datos del conjunto de validación no se produce la superposición de audios, esto provoca que en los audios menores de 10

segundos haya más silencios. En los audios con silencios son más fáciles de quitar la reverberación y, por tanto, el error es menor.

Este efecto puede influenciar en los lotes de entrenamiento, donde hay una gran diferencia de error entre varios lotes.

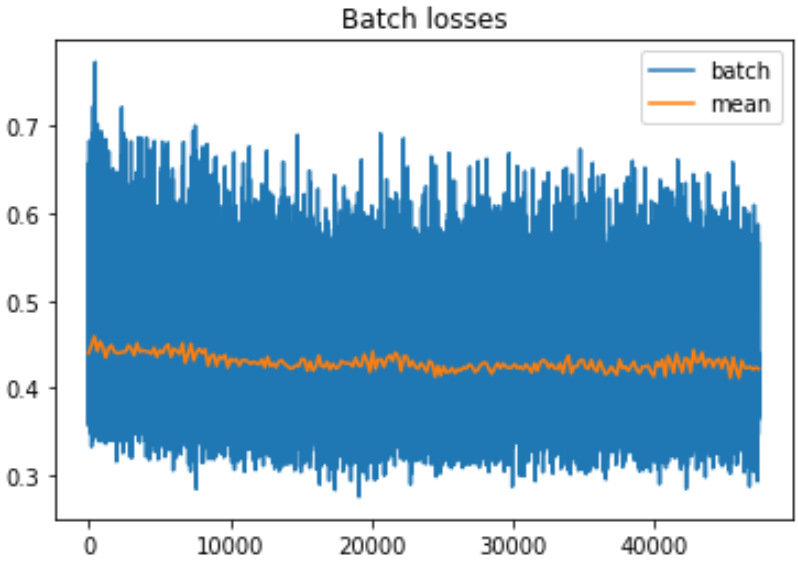


Figura 33 Error L1 (eje vertical) por cada lote (eje horizontal) usando el modelo GRU y lotes de 2

Finalmente se entrenó esta última configuración 200 iteraciones para saber si podría conseguir eliminar la reverberación, sin conseguir ningún avance prometedor. En la figura 34 se puede observar como el error de entrenamiento y el de validación permanecen estables, sin obtener ninguna mejoría muy apreciable.

Debido a un error en el código, solo se pudo registrar el error en validación después de las 75 primeras iteraciones y además este error sufre del problema que se comentó anteriormente con la aumentación de datos.

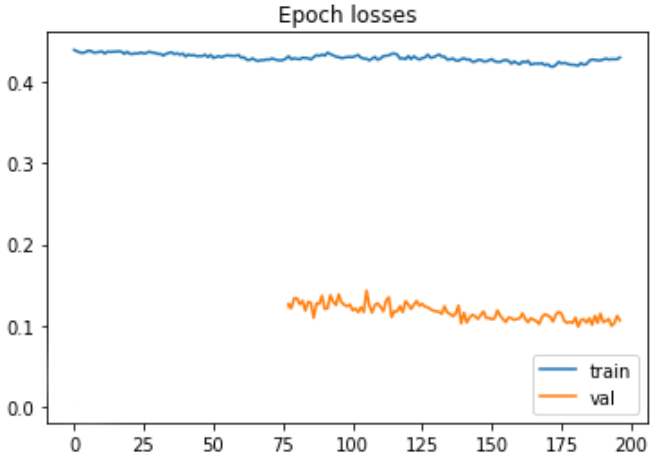


Figura 34 Error durante el entrenamiento de 200 iteraciones

El sonido generado por la red es muy similar al que es introducido, y contiene gran parte de la reverberación, además de una calidad peor. En la figura 36 se pueden ver los espectrogramas de los audios usados para el entrenamiento y el

obtenido por la red. Se puede observar que el obtenido por la red no es exactamente igual que el audio introducido como entrada, pero a pesar de ello, al oído se escuchan muy similares.

Figura 35 Errores por lotes de las 200 iteraciones

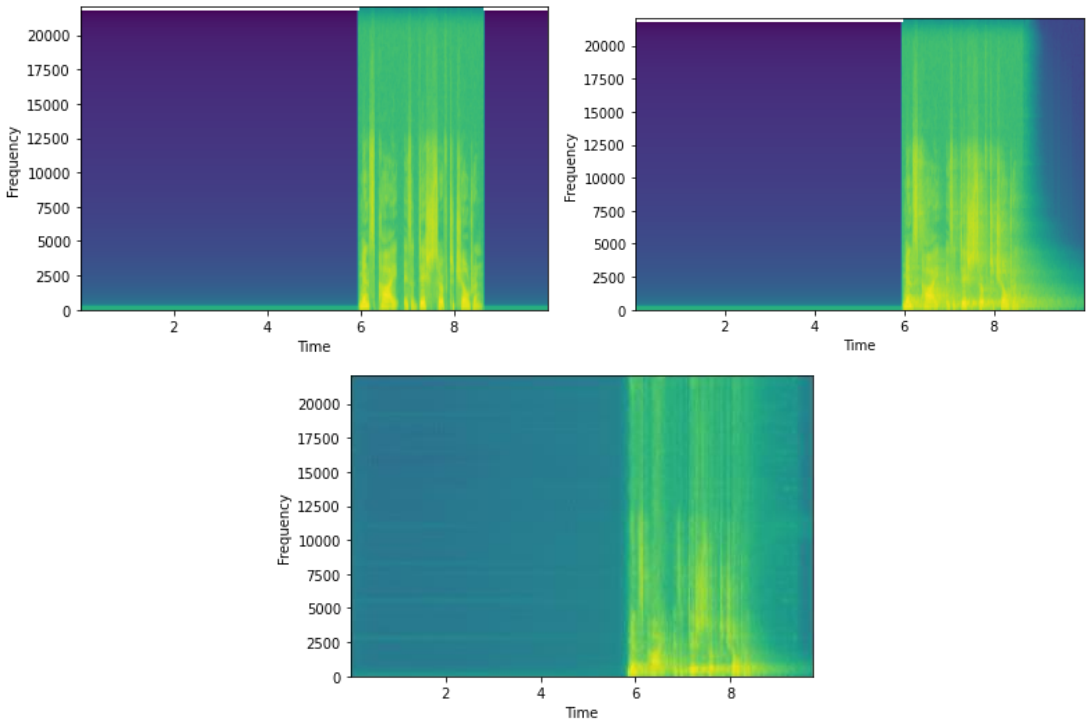


Figura 36 Espectrogramas de un audio aplicado a la red

Tras estos resultados se plantearon nuevos cambios que podrían aplicarse a la red para conseguir que aprendiese.

El primer cambio planteado fue reducir la duración de los audios para conseguir que en los audios de entrenamiento hubiera menos silencio y forzar a la red a aprender más. La mayoría de los audios tienen una longitud menor de 10 segundos, como puede verse en el histograma de la Figura 37, por lo que no pueden llenar los 10 segundos de audio con los que se entrena la red.

La reducción de la duración también debería afectar positivamente a la velocidad de entrenamiento al ser más cortos, pero finalmente se observó que no tenía ningún efecto apreciable. Además 10 segundos parecía una medida propicia enseñar a la red a remover las colas de la reverberación y con longitudes menores de 6 segundos estas colas eran cortadas.

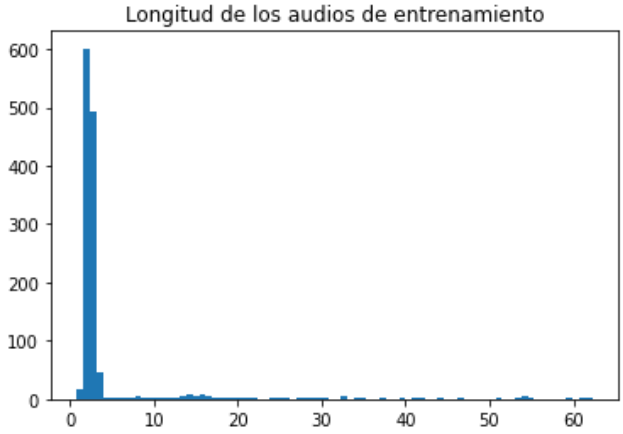


Figura 37 Histograma de la longitud de los audios usados en el entrenamiento.

El segundo cambio planteado fue el de crear una nueva función de error que permitiese mejorar a la red.

3.1.5 Función de error SoundLoss

La función de error usada para calcular el error producido por la red y por tanto, guiar a la red por el descenso del gradiente es la métrica L1.

Esta medida, al igual que el error cuadrático medio o MSE es muy sensible a traslaciones debido a que se basan en restar elementos en la misma posición, como podemos ver en sus definiciones:

$$MSE(x, y) = \frac{1}{N}\sum_{i=1}^{N}(x_i - y_i)^2$$

$$L1(x, y) = \frac{1}{N}\sum_{i=1}^{N}|(x_i - y_i)|$$

Donde N es el número de elementos en los vectores que se desean comparar x e y, que deben tener la misma longitud.

Esto es más problemático cuando los elementos cercanos tienen valores muy distintos, pues pequeños desplazamientos producirán grandes cambios y un claro ejemplo de este tipo de datos que pueden generar este problema son los sonidos agudos con altas frecuencias.

En la figura 38 podemos ver un ejemplo de cómo al comparar el error usando estas dos medidas sobre la misma onda desplazada puede generar que

devuelvan valores desproporcionados ante ondas que para el oído humano son idénticas.

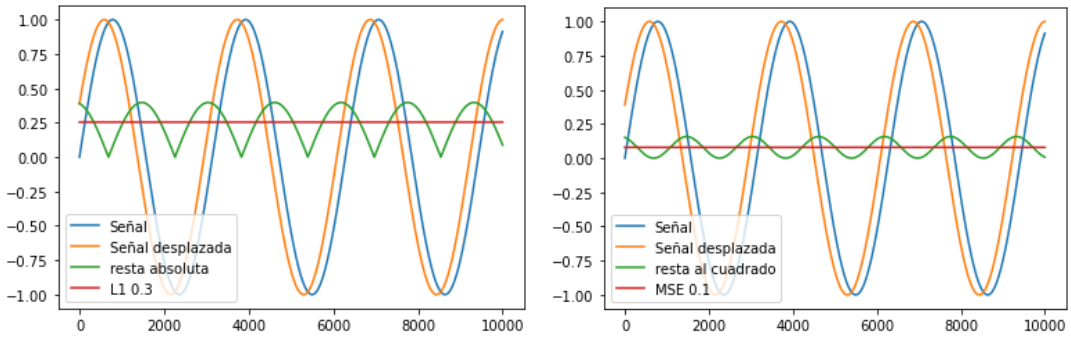


Figura 38 Pequeña demonstración de cómo un desplazamiento de la misma señal puede causar que el error medido por MSE o L1 aumente.

Este problema es mencionado en el artículo de Demucs[1], pero concluye que, para la tarea de separar pistas, L1 y MSE serán una buena métrica siempre que la entrada y la salida estén alineadas.

Sin embargo, como menciona el mismo artículo en la generación de sonido se han usado otras métricas invariantes al desplazamiento o cambio de fase de la señal como en el modelo SING[29]. Es probable que en la eliminación de la reverberación parte del audio deba de ser generada más que enmascarada, por lo que necesite de métricas de error más complejas.

La solución es usar una función de coste que sea invariante ante pequeños desplazamientos. En el artículo Mean Square Error, Love it or leave it[20] se presenta este problema y también una medida para sustituir a MSE en estos casos, la Similaridad Estructural o SSIM.

Esta medida recorre los pares de datos como si fuera una convolución comparando la media de la ventana y su varianza para posteriormente hallar la media de todas estas medidas. Las traslaciones y ruido que sean menores que el tamaño de la ventana afectarán menos a la medición del error.

$$SSIM(x,y) = l(x,y) \cdot c(x,y) \cdot s(x,y) = \left(\frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1}\right) \cdot \left(\frac{2\sigma_x\sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2}\right) \cdot \left(\frac{\sigma_{xy} + C_3}{\sigma_x\sigma_y + C_3}\right)$$

En este caso x e y representan pixeles de la imagen. La función l y el primer término de la última igualdad miden la luminancia entorno al píxel, calculado con medias muestras de alrededor del píxel μ_x y μ_y. La función c se centra en los contrastes de la imagen y realiza el mismo proceso con las desviaciones estándar σ_x y σ_y. Finalmente, la función mide las estructuras locales. Estas tres formulas pueden suelen ser ponderadas con hiperparámetros. Los coeficientes C1, C2 y C3 son también hiperparámetros.

SSIM viene implementada en la librería Scipy, pero no es posible integrarla con el sistema de gradientes de Pytorch, por lo que se tendría que implementar de nuevo usando la API de Pytorch.

Otra solución es convertir la red en una cGAN, dejando Demucs como generador y crear una red discriminadora que intente distinguir entre pares con y sin

reverberación generados por Augmented Reverbs y pares con reverberación y procesados por Demucs. Esta solución podría haber generado buenos resultados, pero debido a la complejidad que supone entrenar este tipo de redes y las modificaciones que deberían hacerse se decidió probar otras métricas.

En el libro Speech Dereverberation[7] aparecen varias métricas para comparar sonidos, en concreto el Logaritmo de la Distorsión Espectral o LSD es la más simple de todas.

Esta métrica compara dos espectrogramas de frecuencia donde se ha aplicado un logaritmo a su intensidad para calcular los decibelios. Finalmente se calcula la media de sus diferencias y se realiza una raíz cuadrada.

$$LSD(x,y) = \frac{1}{M}\sum_{i=0}^{M}\sqrt{\frac{1}{N}\sum_{j=0}^{N}\left(log_{10}\left({x_{ij}}^2\right) - log_{10}({y_{ij}}^2)\right)^2}$$

En la formula anterior M es el número de divisiones del tiempo del espectrograma y N el número de frecuencias. Las variables x e y representan matrices de espectrogramas de audio.

Al tratarse de una métrica antigua existen bastantes variaciones de esta, además que diferentes configuraciones de los espectrogramas pueden conducir a diferentes medidas de error, por lo que se debe de tener especial cuidado al comparar resultados de distintos trabajos.

Al comparar frecuencias estos desplazamientos no son tan importantes, ya que una onda desplazada ligeramente tiene las mismas frecuencias por lo que también es invariante a estos cambios.

El modelo SING[29] usó un función de error similar y esta fue implementada en Demucs[1] en las primeras pruebas sin obtener ninguna mejoría apreciable. Sin embargo, en este proyecto ayudo a mejorar los resultados de la red como veremos posteriormente.

A pesar de que Pytorch no cuenta con esta función de error, la librería Pytorch contiene un módulo preparado para trabajar con audio, Pytorch Audio y esta si implementa espectrogramas de frecuencia.

Usando esta función se implementó la función de error LSD creando una clase llamada SoundLoss. El espectrograma creado por esta función es el que se puede observar en la Figura 39, en esta visualización el eje horizontal es el tiempo y el vertical la frecuencia, pero las frecuencias graves están en la parte superior.

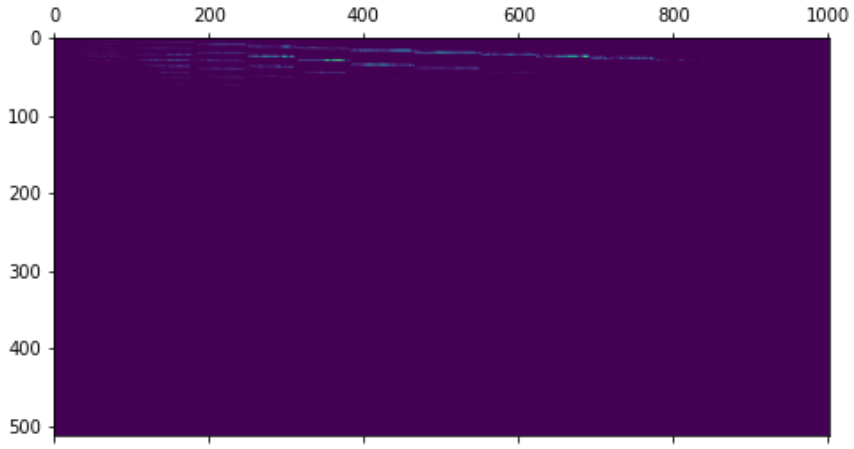


Figura 39 Espectrograma generado por la función spectrogram de la librería Pytorch Audio del módulo functional

Como la información de la frecuencia se encuentra en unos pocos pixeles la red apenas podía guiarse con ellos para progresar en el aprendizaje.

En un primer intento se intentó arreglar el problema ponderando esta medida con la medida L1, pero resulto en sonidos con artefactos extraños y espectrogramas donde se aprecia problemas con las frecuencias.

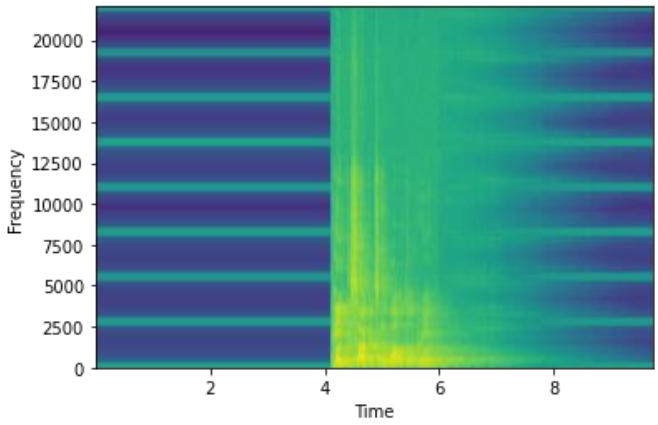


Figura 40 Espectrograma de audio creado por la red al usar una ponderación de las métricas de LSD y L1

Este problema se pudo resolver reescalando el espectrograma usando la escala de Mel, aunque esto generó otro problema.

Pytorch audio cuenta con una parte funcional, que puede ser usada con el sistema de gradientes de Pytorch autograd y otra de transformaciones que opera con tensores, pero no puede funcionar con autograd. La función de escala Mel únicamente se encontraba en el módulo de transformaciones.

Tras revisar el código de la función de la escala Mel en GitHub, el código de la función se modificó para que fuese compatible con autograd. La función precalculaba una variable y la almacenaba en un buffer de memoria, el cual no era compatible con autograd. Finalmente se optó por crear esta variable en la llamada a la función, aunque puede optimizarse guardándola en memoria de otra manera compatible con autograd.

Llamaremos a esta nueva métrica, combinación de MSE y la modificación de LSD, por el nombre de la clase que la implementa, SoundLoss, para no confundirla con la métrica LSD.

$$SoundLoss(x, y) = \omega \cdot LSD\big(MelSpect(x), MelSpect(y)\big) + (1 - \omega) \cdot MSE(x, y)$$

Las siguientes figuras representan los pasos del algoritmo LSD modificado en la métrica SoundLoss para generar la medida de error. Primero en la Figura 41 se obtienen los espectrogramas de Mel.

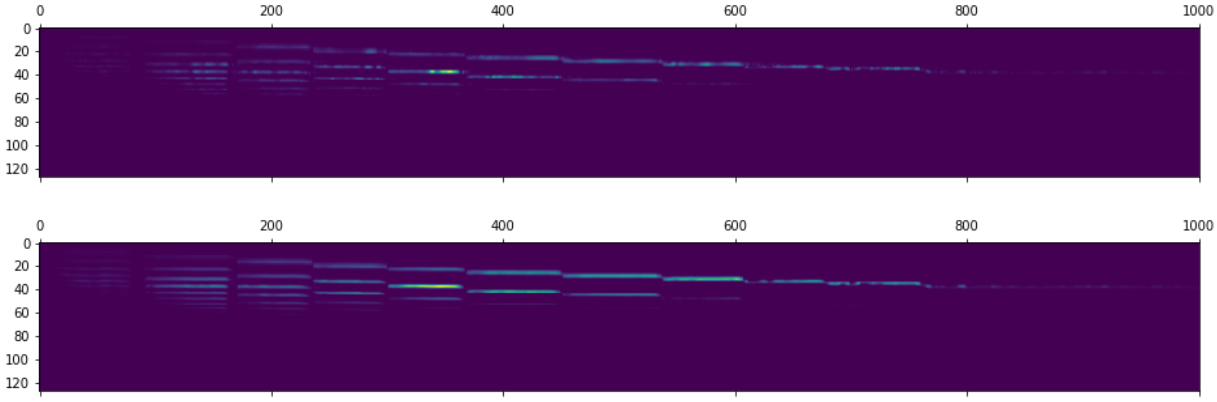


Figura 41 Espectrogramas de audio en escala Mel, arriba el audio de entrada con reverberación y abajo el audio objetivo sin reverberación

En la Figura 42 se amplifica la señal para que las diferencias den valores mayores usando el logaritmo en base 10. Se le suma una constante de valor 0.1 para evitar errores con el logaritmo de 0.

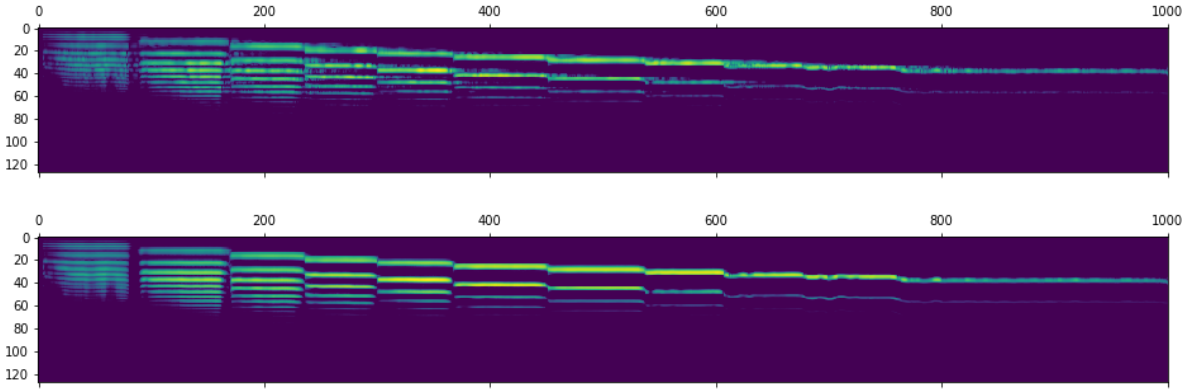


Figura 42 Espectrogramas con la intensidad amplificada al aplicarse un logaritmo en base 10

Posteriormente se restan ambas señales y se elevan al cuadrado para obtener un resultado positivo, como se puede observar en la Figura 43. Como en este ejemplo estamos comparando una señal sin reverberación y su pareja con reverberación, la diferencia es todos aquellos sonidos creados por el efecto de la reverberación y que deseamos eliminar.

Figura 43 Diferencia entre los dos espectrogramas al cuadrado

Finalmente aplicamos la media en cada instante de tiempo para obtener una medida de cuanto se diferencian las frecuencias en ese momento y se le aplica una raíz cuadrada, obteniendo un vector como el de la Figura 44.

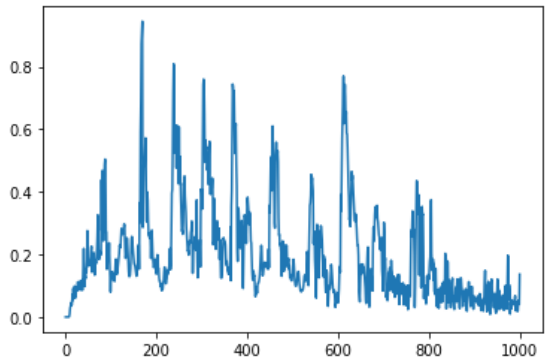


Figura 44 Media de la diferencia en donde el eje horizontal representa el tiempo

Por último, hallamos la media del vector, obteniendo la métrica del error LSD. Este resultado se pondera con la métrica MSE dando como resultado el error de la métrica SoundLoss.

Cuando esta métrica es usada sobre un batch, los audios pertenecientes al batch son concatenados.

Para comprobar si era necesario realizar la ponderación con la métrica L1, se probó a usar una ponderación que anulase el efecto de la métrica L1. Se obtuvieron buenos resultados rápidamente pero el sonido resultante era en ocasiones confuso. Cuando una reverberación se acopla con el siguiente sonido y tiene la misma frecuencia la red no puede distinguirlo y por tanto eliminarlo.

La métrica LSD permite al método entrenar rápidamente en las primeras iteraciones, pero luego es la métrica L1 la que termina ajustando la señal creada perfectamente a la señal objetivo, eliminando incluso las reverberaciones que se acoplan en la misma frecuencia.

Con este método por fin se logró obtener resultados aceptables y que permiten suponer que con modelos más grandes y mejores datos se pueden obtener incluso mejores resultados.

3.1.6 Resolución de errores

Tras realizar estos cambios se intentó probar el modelo con audios grabados por un micrófono la red, pero estos produjeron resultados con ruido. En un principio se pensó que podría ser efecto de un sobreajuste a los audios, por lo que se añadió un nuevo dataset de audio.

El conjunto de datos de artículos de Wikipedia[30] recopila grabaciones de personas leyendo artículos de la Wikipedia para que puedan ser accedidos por otros usuarios de esta forma. Se recogieron audios de artículos en inglés y se realizó una criba donde se eliminaban aquellos audios de mala calidad, con reverberación o que fuesen sintetizados por un programa, obteniendo 383 nuevos audios.

Finalmente se descubrió que el ruido generado era producto de que el audio de entrada de la red no había sido normalizado. A pesar de ello, los audios añadidos ayudaron a la red a generalizar más y obtener mejores resultados.

Durante el entrenamiento en ocasiones los pesos de la red se convertían en Nan. Este error se producía durante el proceso de retropropagación del gradiente, por lo que se usó la función detect_anomaly de autograd para obtener más información sobre el error. Finalmente, el error se encontraba en un operador exponencial de la función de perdida LSD, que fue sustituido por una llamada a la función de pytorch pow, resolviendo el error.

3.2 Código final

El código final está basado en la arquitectura desarrollada por Facebook Demucs, al que se le han realizado diversas modificaciones por los motivos que se cuentan en el capítulo de desarrollo cronológico. Estos cambios se relatan en los siguientes apartados.

Puede encontrarse el código completo en el siguiente repositorio de Github

https://github.com/Xirzag/TFM-Audio-Dereverberation

3.2.1 Red Neuronal

La red neuronal finalmente usada para eliminar la reverberación es una modificación de la versión light de Demucs, la cual tiene menos parámetros entrenables. La versión estándar también puede utilizarse para eliminar la reverberación, pero se carecía de los recursos necesarios para entrenarla.

Como se describe en el estado del arte, la red Demucs es una red del tipo U-Net diseñada para separar pistas usando directamente su señal. La red se modificó para que aceptase de entrada y salida una única pista mono ya que la mayoría de las grabaciones de audio se realizan en este formato.

Entre el codificador y el decodificador de Demucs se encontraba una capa biLSTM que ha sido reemplazada por una capa GRU bidireccional, reduciendo el número de parámetros y obteniendo un rendimiento similar. Estas capas permiten que la red recuerde los sonidos que ha escuchado anteriormente, como se muestra en el apartado de redes recurrentes del estado del arte.

En la Figura 45 se puede observar una representación de la arquitectura final. Todos los parámetros, incluyendo la elección de usar GRU en vez de LSTM, han sido parametrizados.

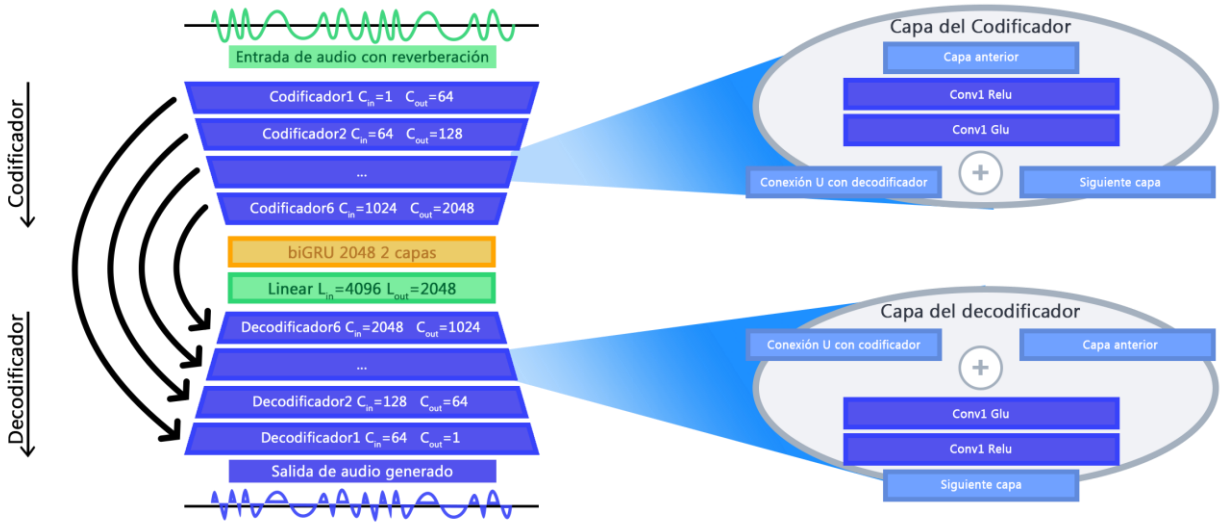


Figura 45 Estructura de Demucs modificada para la eliminación de la reverberación

3.2.2 Entrenamiento

El entrenamiento es la parte que ha sufrido más modificaciones. A partir del código de entrenamiento de Demucs se creó un cuaderno Jupyter de entrenamiento, el cual esta creado para ser compatible con Google Colab y Google Drive.

Conjunto de datos

Para entrenar a la red se usan pares de audio sin reverberación y con reverberación. El audio con reverberación se usa como entrada y la salida generada se compara con el audio sin reverberación grabado en una cámara anecoica.

Se usan dos conjuntos de audios, el Speech Database de Telecommunications & Signal Processing Laboratory Multimedia Signal Processing[27] con diálogos en inglés y OpenAIR[28] con grabaciones de instrumentos. Adicionalmente se añadió el conjunto de audios de grabaciones de artículos de Wikipedia[30] del que solo se recogieron 383 audios que tenían buena calidad.

Estos audios se dividen en 80% un conjunto de datos de entrenamiento, 5% validación y 15% para test y son almacenados en una clase llamada Augmented Reverbs.

Aumentación de datos

La clase Augmented Reverbs almacena los audios sin reverberación y permite generar los pares de audio con y sin reverberación, accediendo al objeto de la clase con un iterador.

Para generar los pares con reverberación se hace uso de la librería pysndfx, a la cual se le puede pasar parámetros para configurar esta reverberación. La clase Augmented Reverbs permite especificar rangos para estos parámetros para los cuales se generará aleatoriamente una configuración cada vez que se devuelva un par de sonidos.

Los audios devueltos por Augmented Reverbs siempre tienen la misma longitud para poder ser agrupados en lotes de entrenamiento. Para ello los audios devueltos son recortados o colocados aleatoriamente en una pista de una longitud determinada, en este caso 10 segundos.

La última aumentación de datos de Augmented Reverbs permite superponer varios audios aumentando aún más la variedad de audios que ve la red y evitando que, debido a que la mayoría de los audios duran menos de 5 segundos, haya menos silencios en los audios de entrenamiento.

En pseudocódigo la aumentación de datos es la siguiente.

```
función aumentacion_datos(índice_audio)
    audio_sin_reverberación = silencio(duración)

    para cada superposición + 1:
        audio = carga_audio(índice_audio)
        si audio.duracion > duracion_seleccionada
            audio = recortar_audio_aleatoriamente(audio, duración)
        sino
            audio = agrandar_audio_aleatoriamente(audio, duración)

        audio_sin_reverberación = mezclar(audio_sin_reverberación, audio)
        índice_audio = nuevo_índice_aleatorio()

    parametros_reverberación = crear_parametros_aleatorios()
    audio_con_reverberación = aplicar_reverberación(
                                        audio_sin_reverberacion,
                                        parametros_reverberacion)

    media = media(audio_sin_reverberación)
    std = desviación_estándar(audio_sin_reverberación)

    audio_sin_reverberación = (audio_sin_reverberación - media) / std
    audio_con_reverberación = (audio_con_reverberación - media) / std

    retornar audio_sin_reverberación, audio_con_reverberación
```

Debido a problemas con Pytorch, los audios son cargados en memoria RAM, acelerando el entrenamiento ya que no hace falta leerlos del disco.

En el cuaderno Jupyter los objetos creados de esta clase son guardados en archivos pickle, ya que la lectura de todos los audios para crear el objeto puede conllevar más de 25 minutos, mientras que cargar estos objetos, a pesar de su gran tamaño, no tarda en demorarse más que unos pocos segundos.

Función de error

Demucs y por tanto la red creada calcula el error comparando el audio objetivo sin reverberación con el audio generado por la red. Debido a que el método que usaba Demucs no daba buenos resultados se cambió a una nueva métrica que fue implementada en la clase SoundLoss.

Esta métrica es a su vez la suma ponderada de dos métricas, la métrica L1 usada por Demucs sobre los pares de audio y una modificación de la métrica Logaritmo de la Distorsión Espectral o LSD. Esta nueva métrica compara los espectrogramas en escala Mel de ambos audios y devuelve la media de las diferencias encontradas.

Tanto la métrica SoundLoss como las métricas de la componen se definen como:

$$LSD(x,y) = \omega \frac{1}{M} \sum_{i=0}^{M} \sqrt{\frac{1}{N} \sum_{j=0}^{N} \left(log_{10}({x_{ij}}^2) - log_{10}({y_{ij}}^2)\right)^2}$$

$$L1 = \frac{1}{N} \sum_{i=1}^{N} |(x_i - y_i)|$$

$$SoundLoss(x,y) = \ \omega \cdot LSD\big(MelSpect(x), MelSpect(y)\big) \ + \ (1 - \ \omega) \cdot L1(x,y)$$

Donde x e y son las señales de audio y MelSpect la función que genera los espectrogramas Mel de audio. El hiperparámetro ω pondera ambos términos.

Sound Loss cuenta con parámetros para configurar los espectrogramas generados, utilizando la API de Pytorch Audio, que a su vez están basados en la librería librosa. Estas implementaciones funcionan o han sido modificadas para funcionar con autograd y permitir su uso en una función de coste.

Sound Loss también cuenta con un parámetro de Debug para visualizar los espectrogramas, pero el gradiente devuelto no puede ser usado para entrenar la red.

Parámetros del entrenamiento

La red es entrenada en lotes de 4 de una duración de aproximadamente 10 segundos para ajustarse al tamaño de entrada de la red. No se ha podido aumentar la cantidad del lote debido al intenso uso de GPU que provoca.

Se usa Adam como optimizador con ratios de aprendizaje bajos, de 1e-4 a 1e-5 La ponderación de la métrica SoundLoss es de un 40% L1 y 60% la métrica LSD modificada. Los parámetros del espectrograma de Mel que no son aquellos por defecto son un nfft de 1024, una definición de 128 mels, un tamaño de ventana de 0.015 segundos o 661 frames y un desplazamiento de 0.01 segundos o 441 frames.

Estos parámetros son los recomendados por Haytham Fayek en su blog[5] y experimentalmente han dado buenos resultados, pero para aumentar su definición se aumentó los valores nfft y de mels.

Para el entrenamiento se usa aumentación de datos en el conjunto de entrenamiento y el de validación para que tengas errores similares. A ambos se le superpone otro sonido encima del ya presente que se va a procesar. Estos sonidos no se comparten entre conjuntos de entrenamiento y validación y pertenecen a su mismo conjunto de datos.

Los rangos de parámetros de reverberación con los que se entrena la red son reverberancia de 10 a 100, tamaño de la sala 10 a 100, retardo de 5 a 25ms y la amortiguación de frecuencias altas de 0 a 100. Estos parámetros que van de 0 a 100 son los parámetros de entrada de la librería Sound Exchange o Sox, de la cual pysndfx no es más que una interfaz de uso desde Python.

Tras 50 a 100 iteraciones se pueden observar buenos resultados como se puede observar en la gráfica del error.

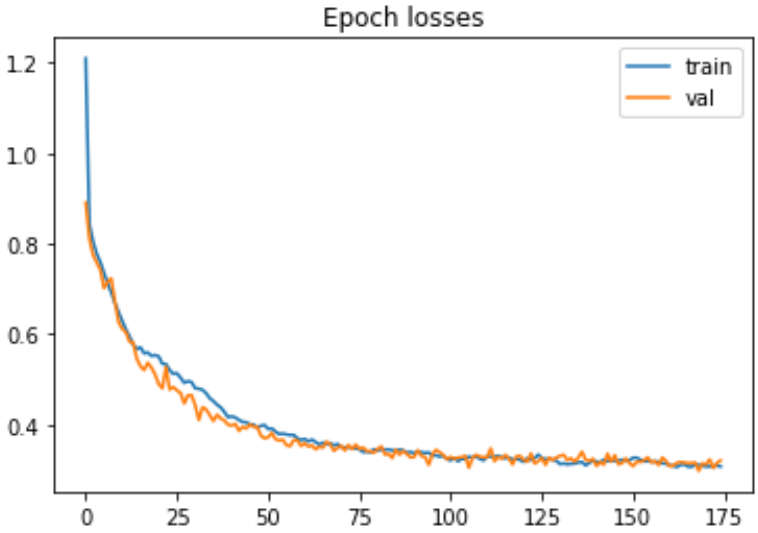


Figura 46 Error SoundLoss usando el entrenamiento descrito

4 Resultados

Tras entrenar el modelo podemos evaluarlo sobre el conjunto de datos de prueba, cuyos audios nunca han sido vistos por la red. Este conjunto se creó, como se describe en el desarrollo cronológico, repartiendo los datasets de audio en tres conjuntos de datos, uno de entrenamiento con el que entrenar la red, otro de validación para comprobar el error durante el entrenamiento y un conjunto de prueba para probar el error de la red una vez entrenada.

No se ha podido comparar con otros métodos debido a que no ha sido posible acceder a ellos para evaluarlos con los mismos datos. Tampoco se ha podido usar las métricas de The Reverb Challenge debido a errores en el Script de evaluación. Por este motivo sólo se proveerán métricas de este trabajo.

Los resultados de la red se pueden medir con métodos objetivos o subjetivos. Los métodos subjetivos de realce del habla requieren realizar experimentos con oyentes que midan la inteligibilidad de las frases. En este trabajo solo se han realizado métricas objetivas sobre el conjunto de datos de prueba.

Las métricas usadas han sido L1, MSE, y las métricas específicas de audio LSD y la métrica usada para el entrenamiento SoundLoss que suma de forma ponderada L1 y LSD en una proporción 40-60.

La métrica LSD aquí usada se realiza con el espectrograma de Mel, por lo que es igual a utilizar el SoundLoss con el hiperparámetro ω a 1. La medida L1 sería la resultante de usar SoundLoss con el hiperparámetro ω valiendo 0.

Tabla 1 Medidas de error del modelo evaluado sobre el conjunto de prueba

	L1	MSE	LSD	SoundLoss
count	294	294	294	294
mean	0.223976	0.122252	0.988783	0.642171
std	0.038481	0.043753	0.148358	0.095656
min	0.133845	0.049974	0.403183	0.269501
25%	0.197845	0.094688	0.901846	0.585648
50%	0.219325	0.113088	0.992255	0.640838
75%	0.240564	0.139883	1.084041	0.702272
max	0.457635	0.351255	1.430595	0.969999

El error SoundLoss medio del conjunto de entrenamiento es de 0.42, por lo que se observa un ligero sobreajuste con respecto al conjunto de datos de prueba.

En las siguientes gráficas podemos observar el error generado por el modelo con respecto a la duración del audio y el tipo de audio de cada conjunto de datos.

El primer dato importante que resaltar es que mientras que los conjuntos de audio de grabaciones de artículos de audio y diálogos de voz están situados en un cierto rango, las grabaciones de música están más dispersas en el eje X.

Esto puede ser debido a que estas grabaciones son realizadas por diferentes instrumentos, además de que algunos instrumentos aparecen en más audios que otros, por lo que la red tiene menos ejemplos con los que aprender. Los dos instrumentos que generan más error son una guitarra, la cual es difícil retirar la reverberación debido a que su sonido proviene de la reverberación de la caja de resonancia. El segundo es una batería, de la cual solo hay un ejemplo en el dataset. Entre las grabaciones con menor error hay una gran cantidad de instrumentos de viento, la mayor parte trompetas.

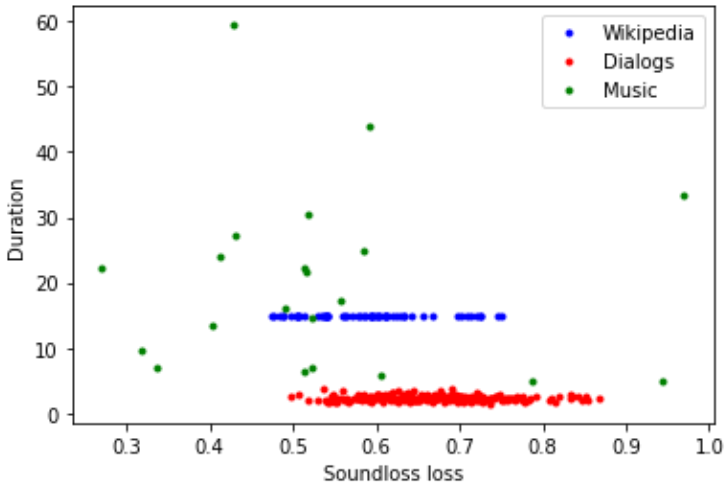


Figura 47 Gráfica de errores SoundLoss del conjunto de prueba. El error se muestra en el eje X, en el eje Y la duración del audio. Cada conjunto de audio está representado en un color.

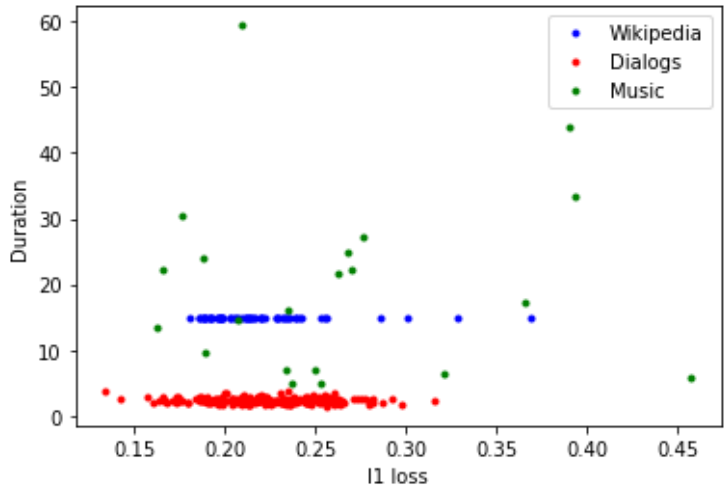


Figura 48 Gráfica de errores L1 del conjunto de prueba.

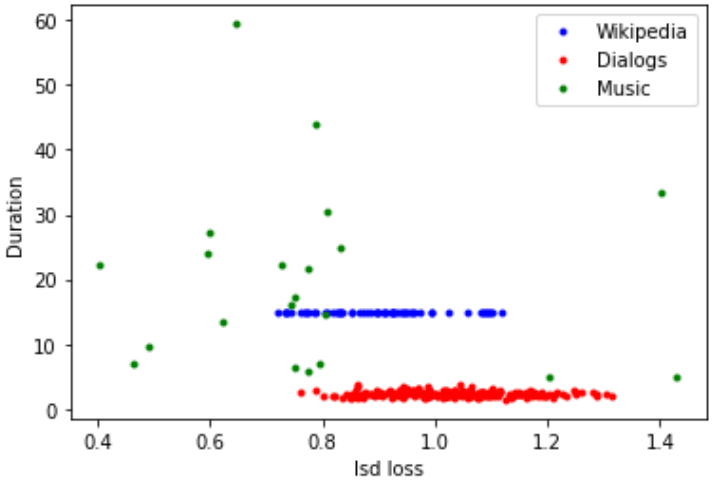


Figura 49 Gráfica de errores LSD del conjunto de prueba.

Mientras que la desviación estándar del error SoundLoss en el conjunto de diálogos y artículos de Wikipedia son de 0.075 y 0.073, para el conjunto de audios de instrumentos es de 0.175.

La duración del audio no parece afectar al rendimiento del modelo a pesar de que el entrenamiento siempre se realizó con la misma longitud. Como gran parte de los audios tienen las mismas duraciones, se recortaron aleatoriamente. Se observa una mayor dispersión que puede estar relacionada a que cuanto mayor sea el audio, es más probable que este contenga partes más sencillas de eliminar la reverberación y otras más difíciles. Al calcular la media de todo el audio unas partes compensan a las otras, pero esto en audios más cortos es menos probable que pase.

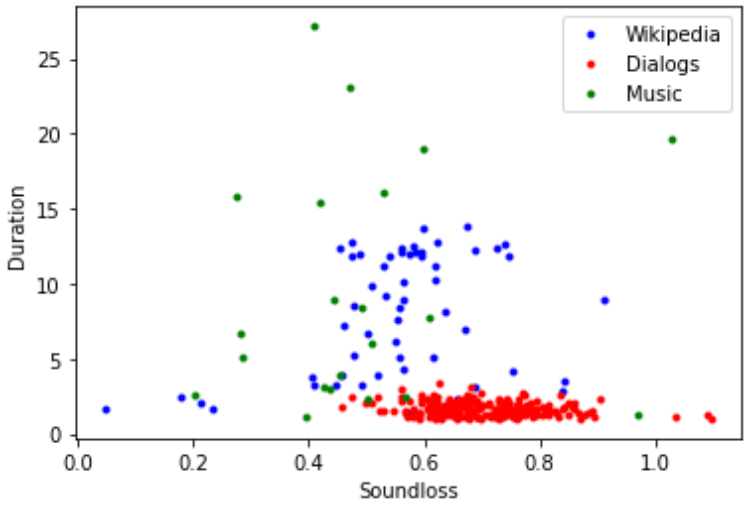


Figura 50 Errores SoundLoss con audios recortados para aumentar la variabilidad de sus longitudes

Las grabaciones de artículos de Wikipedia han sido las más afectadas por esta dispersión, haciendo que su desviación estándar sea de 0.151. Los otros conjuntos también se han dispersado, siendo su desviación estándar de 0.096 para el conjunto de diálogos y 0.194 para el conjunto de instrumentos.

En cuanto a cómo afecta la reverberación aplicada al audio podemos observar cómo se desempeña el modelo con diferentes configuraciones del generador de reverberación, el cual tiene cuatro parámetros, la cantidad de reverberación aplicada, el tamaño de la sala, el retardo del eco y la atenuación de frecuencias altas.

La cantidad de reverberación es el parámetro que lógicamente más varía el error generado por la red. Cuanta más reverberación más difícil es para la red eliminarla, como se puede observar en la gráfica 51. En esta gráfica el error se dispone en el eje vertical. La zona en gris representa la región que comprende los cuantiles 20 a 80 de todos los audios del conjunto de prueba.

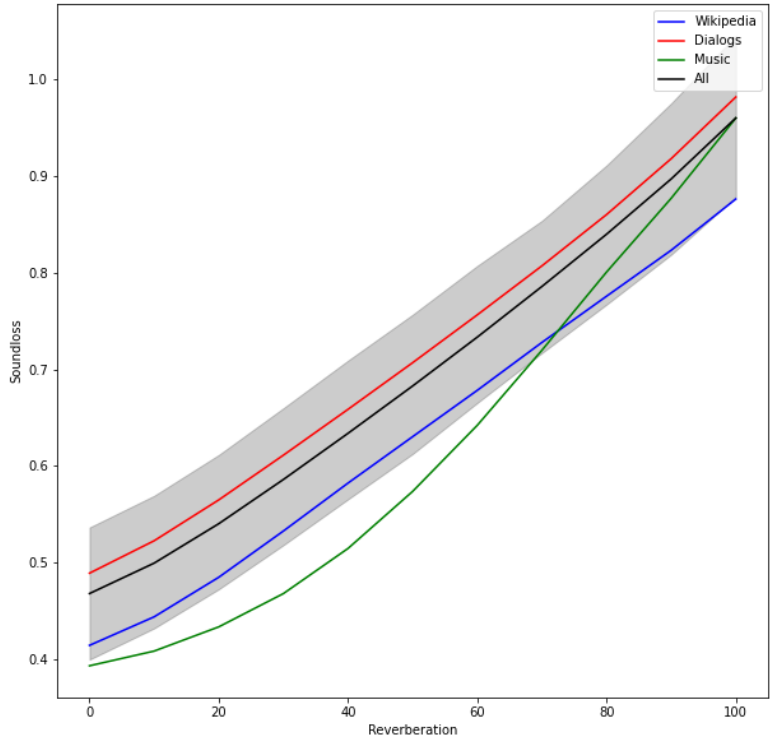


Figura 51 Errores SoundLoss en el eje vertical según se aumenta la reverberación, en el eje horizontal

El tamaño de sala simula que reverberaciones se producirían en estas estancias. Cuanto mayor es la habitación más reverberación puede producir, por lo que vuelve a ser más difícil para la red eliminar la reverberación. Sin embargo, este incremento no supone un gran aumento del error, cuya media solo se eleva en 0.08 unidades.

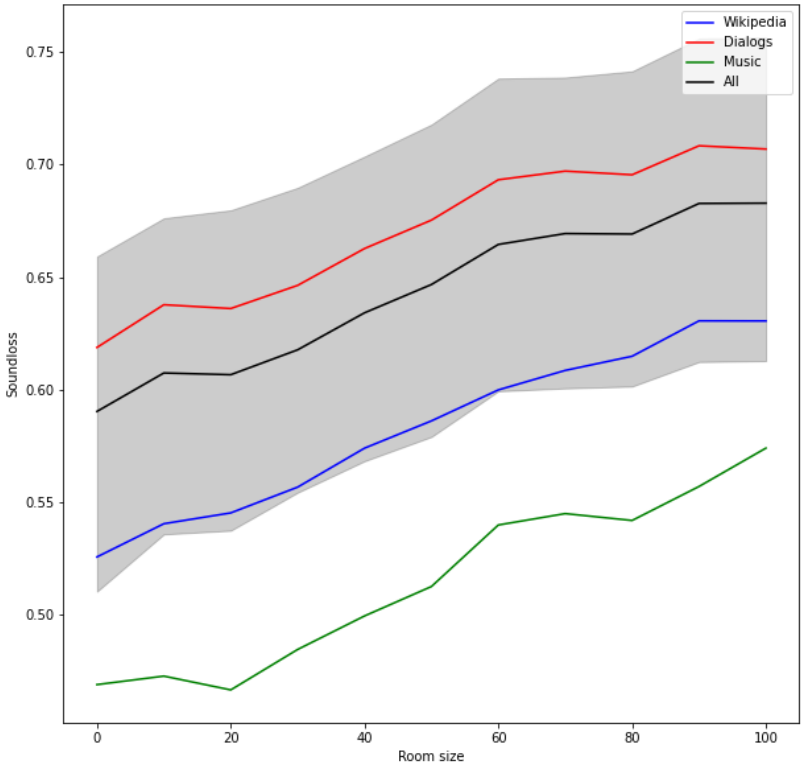


Figura 52 Errores SoundLoss comparados con el tamaño de la sala

La atenuación de las frecuencias altas produce que en las colas de las reverberaciones los sonidos agudos se atenúen. Este efecto también puede ser simulado, pero tampoco tiene apenas efecto en el error. Curiosamente cuanto más se produce este efecto menos error produce la red, lo cual es contraintuitivo.

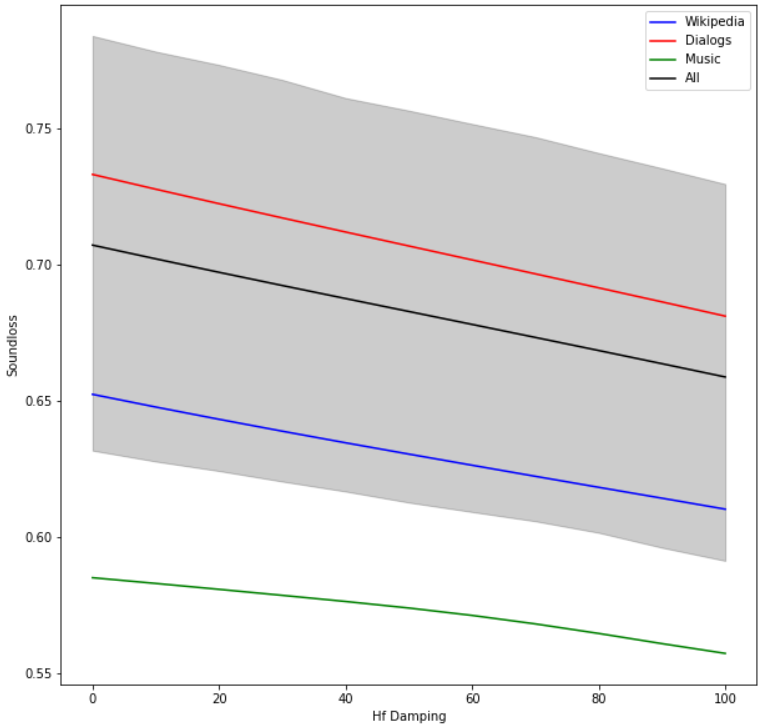
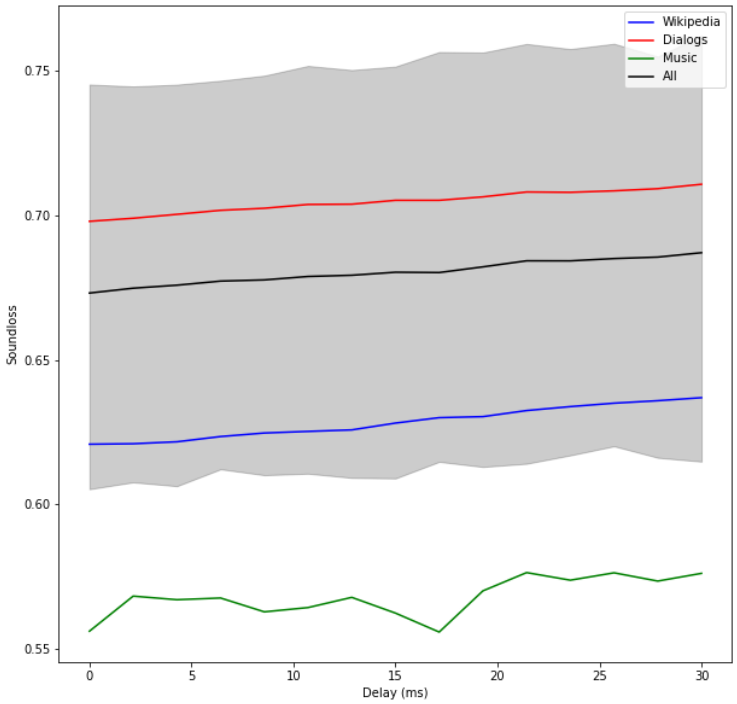


Figura 53 Errores SoundLoss comparado con la atenuación de la frecuencia

Finalmente, el retardo de la primera señal de la reverberación que llega al receptor tampoco varia en gran medida el error producido por la red, aumentando como podía ser esperable, ya que cuanta más separación hay entre un pico de sonido y el pico de la reverberación que produjo el pico anterior, más difícil es identificar este como uno que no sea una reverberación.



En todas las gráficas se puede observar como el conjunto de datos de instrumentos genera una media de errores por debajo del cuantil 20 del error de todos los audios. Esto puede ser debido a lo diferente que son comparados con los audios de diálogos.

En posteriores pruebas con grabaciones propias, las respuestas creadas por la red suelen ser bastante buenas si estas no tienen una gran cantidad de reverberación. En ocasiones en los silencios genera ruido y distorsiona el sonido original, pero generalmente consigue identificar las reverberaciones cuando estas siguen oyéndose a pesar de que el sonido original ha parado y las elimina.

La red entrenada en este trabajo puede probarse en este Notebook de Google Colab, realizado para mostrar la capacidad del modelo para eliminar la reverberación. El notebook permite grabar un audio, aplicarle reverberación y posteriormente retirársela con el modelo entrenado en este trabajo.

Para acceder a él se debe ir a la siguiente dirección y seguir los pasos que aparecen en el Notebook.

https://colab.research.google.com/drive/1DA4hHL1KahFxTa3vs6k8I1ycBk5ZMCsy?usp=sharing

También se pueden escuchar 8 ejemplos del conjunto de prueba en la siguiente web.

https://xirzag.github.io/TFM-Audio-Dereverberation/test_dereverberation.html

5 Conclusiones

La reverberación es un fenómeno que es difícil de eliminar de una grabación en gran parte debido a su variabilidad. Tanto la posición de las fuentes de sonido, de los receptores y la forma y materiales de la estancia donde se realice la grabación afectará al audio recogido por el receptor. Es por esta razón por la cual no se ha conseguido resolver en su totalidad este problema.

En este trabajo se propone la utilización de una modificación de la red Demucs[1] para eliminar la reverberación de grabaciones de audio, consiguiendo resultados prometedores. Estos avances, junto con los nuevos trabajos listados en el estado del arte, aportan nuevas maneras de acercarse a resolver el problema de la eliminación de la reverberación. El uso de Deep Learning permite simplificar el problema en comparación con los enfoques basados en teoría de señales, permitiendo encontrar nuevas soluciones.

La red Demucs, la cual fue diseñada para separar pistas de música, parece tener la capacidad de separar la reverberación del audio original, aunque precisa de ciertas modificaciones para poder realizarlo. Esta capacidad no es perfecta, sino una buena aproximación de la que se pueden restaurar audios grabados en habitaciones muy reverberantes.

Las mejoras realizadas a la red al igual que todo el estudio y pruebas realizadas pueden servir de base para avanzar en este ámbito. La métrica creada para este modelo puede ser usada en otras aplicaciones relacionadas con audio, mientras que el uso de GRU reduce el número de parámetros de la red Demucs obteniendo resultados similares.

Las aplicaciones de la eliminación de la reverberación son bastante amplias. La eliminación de la reverberación en grabaciones puede servir para mejorar la inteligibilidad de llamadas o mensajes de voz, mejorar la calidad de grabaciones aplicado como filtros inteligentes en softwares de edición de audio o en asistentes de voz para mejorar la comprensión de los comandos de voz.

6 Trabajo futuro

Este trabajo puede ser expandido de múltiples maneras ya sea para mejorar la calidad del audio generado o cambiar las posibles aplicaciones de la red.

La calidad del audio generado probablemente puede mejorarse aumentando la calidad y la cantidad de audios con los que se entrena la red. Cuanto más variados sean estos audios, más robusta será la red ante nuevos audios. En caso de que se necesite especializar la red en un cierto ámbito, por ejemplo, la grabación de instrumentos sería lógico suministrarle más muestras de este tipo.

Si se desea quitar las reverberaciones de un lugar en específico se pueden añadir grabaciones de pares con y sin reverberación realizadas allí para entrenar a la red. Otra opción posible es obtener la función AIR de la habitación, simular la posible respuesta a los audios sin reverberación y añadir los pares de audio al conjunto de datos de entrenamiento.

Para ello sería necesario cambiar el sistema actual que simula reverberación por otro más complejo que pueda simular respuestas AIR. Este cambio también podría mejorar la calidad de las respuestas de la red, ya que las reverberaciones simuladas serían más parecidas a las reales.

En caso de añadir estas mejoras y no apreciar ninguna mejoría, una de las posibles soluciones es aumentar el tamaño de la red. El modelo Demucs es escalable y en este trabajo se ha usado la red light debido a la imposibilidad de entrenar la red de tamaño normal.

El modelo Demucs cuenta con un sistema de entrenamiento distribuido para permitir entrenar este enorme modelo. Este sistema debería ser reintegrado en el código modificado de entrenamiento ya que fue retirado debido a que no se iba a utilizar. Al igual que aumentar el tamaño puede tener ciertas ventajas, reducirlo puede servir para crear pequeños modelos con un menor número de parámetros que puedan ejecutarse en dispositivos menos potentes.

Por último, la propia red puede ser modificada y ciertos componentes reemplazados por otros que puedan reducir el error de la red. Una sugerencia puede ser cambiar las capas centrales biGRU, anteriormente biLSTM en Demucs, por Transformers, los cuales han sido utilizados en tareas de clasificación de audio[31].

Otra de las posibles modificaciones es la de convertir la red en una red generativa adversaria, cambiando la función de error por una red discriminadora.

El ámbito del problema de esta red puede ampliarse fácilmente a resolver el problema de realce del habla incluyendo ruido en la generación de los pares con reverberación. De esta forma se obtendría una red que resuelve ambos problemas al mismo tiempo. Probablemente al aumentar la complejidad del problema la red también deba escalarse como se mencionó anteriormente.

Otro ámbito donde podría aplicarse es en la transformación entre pares de audios, como mejorar la calidad de grabaciones de audio. Sin embargo, como

las pruebas realizadas se dirigen a eliminar parte del audio en vez de generar nuevo puede que esta aplicación no de tan buenos resultados.

Finalmente, las modificaciones de este trabajo pueden usarse en las aplicaciones que tenga Demucs, incluyendo su aplicación principal de separar pistas de audio. Reentrenando la red con estas modificaciones, se podrían comparar los resultados obtenidos con los de la red original, permitiendo medir la mejoría de los cambios, en caso de que haya.

Como vimos en el apartado de resultados, también es necesario añadir más métricas, como valoraciones subjetivas y comparaciones con otros métodos.

7 Fuentes de figuras

Las figuras que no aparezcan en esta lista han sido creadas y no han sido descargadas de internet.

Figura 1: Propagación aire

https://es.m.wikipedia.org/wiki/Archivo:CPT-sound-physical-manifestation.svg

Figura 2: Forma onda

https://es.wikipedia.org/wiki/Archivo:Posici%C3%B3n_Mov_periodico.svg

Figura 10: Partes de la reverberación

Recent Advances in Speech Dereverberation[32]

Figura 12: Cámara anecoica
https://commons.wikimedia.org/wiki/File:Consumer_Reports_-_product_testing_-_headphones_in_anechoic_chamber.tif

Figura 13: Modelo de reverberación y eliminación de la reverberación

Speech Dereverberation[7]

Figura 14: Red recurrente

https://en.wikipedia.org/wiki/Recurrent_neural_network#/media/File:Recurrent_neural_network_unfold.svg

Figura 15: Red LSTM

https://en.wikipedia.org/wiki/Long_short-term_memory#/media/File:The_LSTM_cell.png

Figura 16: Red GRU

https://en.wikipedia.org/wiki/Gated_recurrent_unit#/media/File:Gated_Recurrent_Unit,_base_type.svg

Figura 17: Red autoencoder

https://commons.wikimedia.org/wiki/File:Autoencoder_structure.png

Figura 18: Red U-Net

https://en.wikipedia.org/wiki/U-Net#/media/File:Example_architecture_of_U-Net_for_producing_k_256-by-256_image_masks_for_a_256-by-256_RGB_image.png

Figura 19: Red Gan

https://medium.com/sigmoid/a-brief-introduction-to-gans-and-how-to-code-them-2620ee465c30

Figura 20: Red densa

LEARNING SPECTRAL MAPPING FOR SPEECH DEREVERBERATION [15]

Figuras 21-22: Logo y datasets Reverb Challenge

https://reverb2014.dereverberation.com/introduction.html

Figura 23: U-Net para eliminación de reverberación

Speech Dereverberation using Fully Convolutional Networks[3]

Figura 24: Modelo GAN para eliminación de reverberación

Single-channel Speech Dereverberation via Generative Adversarial Training[19]

Figura 25: Modelo BLSTM para eliminación de reverberación

Simultaneous Denoising and Dereverberation Using Deep Embedding Features[21]

Figura 26-27: Modelo Demucs

Music Source Separation in the Waveform Domain[1]

8 Bibliografía

[1] A. Défossez, N. Usunier, L. Bottou, y F. Bach, «Music Source Separation in the Waveform Domain», *arXiv:1911.13254 [cs, eess, stat]*, nov. 2019, Accedido: abr. 16, 2020. [En línea]. Disponible en: http://arxiv.org/abs/1911.13254.

[2] K. Wang, J. Zhang, S. Sun, Y. Wang, F. Xiang, y L. Xie, «Investigating Generative Adversarial Networks based Speech Dereverberation for Robust Speech Recognition», *Interspeech 2018*, pp. 1581-1585, sep. 2018, doi: 10.21437/Interspeech.2018-1780.

[3] O. Ernst, S. E. Chazan, S. Gannot, y J. Goldberger, «Speech Dereverberation Using Fully Convolutional Networks», *arXiv:1803.08243 [cs, eess]*, abr. 2019, Accedido: feb. 28, 2020. [En línea]. Disponible en: http://arxiv.org/abs/1803.08243.

[4] D. Brüel og Kjaer A/S Naerum y Arbetarskyddsfonden, *Noise Control: Principles and Practice*. Brüel & Kjaer, 1986.

[5] Haytham M. Fayek, «Speech Processing for Machine Learning: Filter banks, Mel-Frequency Cepstral Coefficients (MFCCs) and What's In-Between». https://haythamfayek.com/2016/04/21/speech-processing-for-machine-learning.html.

[6] MicroPyramid, «Understanding Audio Quality: Bit Rate, Sample Rate», *Medium*, oct. 04, 2017. https://medium.com/@MicroPyramid/understanding-audio-quality-bit-rate-sample-rate-14286953d71f (accedido jun. 15, 2020).

[7] Patrick A. Naylor y Nikolay D. Gaubitch, *Speech Dereverberation*. .

[8] G. W. Elko, «Microphone array systems for hands-free telecommunication», *Speech Communication*, vol. 20, n.º 3, pp. 229-240, 1996, doi: https://doi.org/10.1016/S0167-6393(96)00057-X.

[9] «Fifty Years of Reverberation Reduction». [En línea]. Disponible en: https://www.audiolabs-erlangen.de/content/05-fau/professor/00-habets/06-activities/Keynote_Habets_2016_AES_60.pdf.

[10] H. Purwins, B. Li, T. Virtanen, J. Schlüter, S. Chang, y T. Sainath, «Deep Learning for Audio Signal Processing», *IEEE J. Sel. Top. Signal Process.*, vol. 13, n.º 2, pp. 206-219, may 2019, doi: 10.1109/JSTSP.2019.2908700.

[11] H.-W. Dong, W.-Y. Hsiao, L.-C. Yang, y Y.-H. Yang, «MuseGAN: Multi-track Sequential Generative Adversarial Networks for Symbolic Music Generation and Accompaniment», *arXiv:1709.06298 [cs, eess]*, nov. 2017, Accedido: abr. 16, 2020. [En línea]. Disponible en: http://arxiv.org/abs/1709.06298.

[12] P. Isola, J.-Y. Zhu, T. Zhou, y A. A. Efros, «Image-to-Image Translation with Conditional Adversarial Networks», *arXiv:1611.07004 [cs]*, nov. 2018, Accedido: jul. 13, 2020. [En línea]. Disponible en: http://arxiv.org/abs/1611.07004.

[13] *Handbook of Neural Networks for Speech Processing*. 2000.

[14] Y. Xu, J. Du, L.-R. Dai, y C.-H. Lee, «An Experimental Study on Speech Enhancement Based on Deep Neural Networks», *IEEE Signal Process. Lett.*, vol. 21, n.º 1, pp. 65-68, ene. 2014, doi: 10.1109/LSP.2013.2291240.

[15] «LEARNING SPECTRAL MAPPING FOR SPEECH DEREVERBERATION», *IEEE International Conference on Acoustic, Speech and Signal Processing (ICASSP)*, 2014, [En línea]. Disponible en: https://web.cse.ohio-state.edu/~wang.77/papers/HWW.icassp14.pdf.

[16] K. Kinoshita *et al.*, «A summary of the REVERB challenge: state-of-the-art and remaining challenges in reverberant speech processing research», *EURASIP J. Adv. Signal Process.*, vol. 2016, n.º 1, p. 7, dic. 2016, doi: 10.1186/s13634-016-0306-6.

[17] J. Eaton, N. Gaubitch, A. Moore, y P. Naylor, *Acoustic Characterization of Environments (ACE) Challenge Results Technical Report*. 2016.
[18] M. Harper, «The Automatic Speech recogition In Reverberant Environments (ASpIRE) challenge», dic. 2015, doi: 10.1109/ASRU.2015.7404843.
[19] C. Li, T. Wang, S. Xu, y B. Xu, «Single-channel Speech Dereverberation via Generative Adversarial Training», *arXiv:1806.09325 [cs, eess]*, jun. 2018, Accedido: mar. 04, 2020. [En línea]. Disponible en: http://arxiv.org/abs/1806.09325.
[20] Zhou Wang y A. C. Bovik, «Mean squared error: Love it or leave it? A new look at Signal Fidelity Measures», *IEEE Signal Process. Mag.*, vol. 26, n.º 1, pp. 98-117, ene. 2009, doi: 10.1109/MSP.2008.930649.
[21] C. Fan, J. Tao, B. Liu, J. Yi, y Z. Wen, «Simultaneous Denoising and Dereverberation Using Deep Embedding Features», *arXiv:2004.02420 [cs, eess]*, abr. 2020, Accedido: abr. 13, 2020. [En línea]. Disponible en: http://arxiv.org/abs/2004.02420.
[22] W.-J. Lee, S.-S. Wang, F. Chen, X. Lu, S.-Y. Chien, y Y. Tsao, «Speech Dereverberation Based on Integrated Deep and Ensemble Learning Algorithm», *arXiv:1801.04052 [cs, eess]*, feb. 2018, Accedido: feb. 28, 2020. [En línea]. Disponible en: http://arxiv.org/abs/1801.04052.
[23] O. Novotny, O. Plchot, O. Glembek, J. «Honza» Cernocky, y L. Burget, «Analysis of DNN Speech Signal Enhancement for Robust Speaker Recognition», *arXiv:1811.07629 [cs, eess]*, nov. 2018, Accedido: abr. 13, 2020. [En línea]. Disponible en: http://arxiv.org/abs/1811.07629.
[24] M. Yasuda, Y. Koizumi, L. Mazzon, S. Saito, y H. Uematsu, «DOA Estimation by DNN-based Denoising and Dereverberation from Sound Intensity Vector», *arXiv:1910.04415 [cs, eess, stat]*, oct. 2019, Accedido: mar. 04, 2020. [En línea]. Disponible en: http://arxiv.org/abs/1910.04415.
[25] D. Stoller, S. Ewert, y S. Dixon, «Wave-U-Net: A Multi-Scale Neural Network for End-to-End Audio Source Separation», *arXiv:1806.03185 [cs, eess, stat]*, jun. 2018, Accedido: may 04, 2020. [En línea]. Disponible en: http://arxiv.org/abs/1806.03185.
[26] Facebook Research, «Repositorio de Demucs en Github». https://github.com/facebookresearch/demucs.
[27] «Telecommunications & Signal Processing Laboratory Multimedia Signal Processing Speech Database». http://www-mmsp.ece.mcgill.ca/Documents/Data/.
[28] «OpenAIR». https://openairlib.net/?page_id=310.
[29] A. Défossez, N. Zeghidour, N. Usunier, L. Bottou, y F. Bach, «SING: Symbol-to-Instrument Neural Generator», *arXiv:1810.09785 [cs, eess, stat]*, oct. 2018, Accedido: jul. 05, 2020. [En línea]. Disponible en: http://arxiv.org/abs/1810.09785.
[30] «Wikipedia:Spoken articles», *Wikipedia*. jul. 12, 2020, Accedido: jul. 12, 2020. [En línea]. Disponible en: https://en.wikipedia.org/w/index.php?title=Wikipedia:Spoken_articles&oldid=967338965.
[31] N.-Q. Pham, T.-S. Nguyen, J. Niehues, M. Müller, S. Stüker, y A. Waibel, «Very Deep Self-Attention Networks for End-to-End Speech Recognition», *arXiv:1904.13377 [cs, eess]*, may 2019, Accedido: jul. 05, 2020. [En línea]. Disponible en: http://arxiv.org/abs/1904.13377.
[32] «Recent Advances in Speech Dereverberation». [En línea]. Disponible en: https://www.research.ibm.com/haifa/Workshops/speech2008/present/Recent_Advances_in_Speech_Dereverberation.pdf.
[33] *Tectonica 14 Acústica*. .

9 Anexos

9.1 Estructura de Demucs

```
Demucs(
  (encoder): ModuleList(
    (0): Sequential(
      (0): Conv1d(2, 100, kernel_size=(8,), stride=(4,))
      (1): ReLU()
      (2): Conv1d(100, 200, kernel_size=(1,), stride=(1,))
      (3): GLU(dim=1)
    )
    (1): Sequential(
      (0): Conv1d(100, 200, kernel_size=(8,), stride=(4,))
      (1): ReLU()
      (2): Conv1d(200, 400, kernel_size=(1,), stride=(1,))
      (3): GLU(dim=1)
    )
    (2): Sequential(
      (0): Conv1d(200, 400, kernel_size=(8,), stride=(4,))
      (1): ReLU()
      (2): Conv1d(400, 800, kernel_size=(1,), stride=(1,))
      (3): GLU(dim=1)
    )
    (3): Sequential(
      (0): Conv1d(400, 800, kernel_size=(8,), stride=(4,))
      (1): ReLU()
      (2): Conv1d(800, 1600, kernel_size=(1,), stride=(1,))
      (3): GLU(dim=1)
    )
    (4): Sequential(
      (0): Conv1d(800, 1600, kernel_size=(8,), stride=(4,))
      (1): ReLU()
      (2): Conv1d(1600, 3200, kernel_size=(1,), stride=(1,))
      (3): GLU(dim=1)
    )
    (5): Sequential(
      (0): Conv1d(1600, 3200, kernel_size=(8,), stride=(4,))
      (1): ReLU()
      (2): Conv1d(3200, 6400, kernel_size=(1,), stride=(1,))
      (3): GLU(dim=1)
    )
  )
  (decoder): ModuleList(
    (0): Sequential(
      (0): Conv1d(3200, 6400, kernel_size=(3,), stride=(1,))
      (1): GLU(dim=1)
      (2): ConvTranspose1d(3200, 1600, kernel_size=(8,),
stride=(4,))
      (3): ReLU()
    )
    (1): Sequential(
      (0): Conv1d(1600, 3200, kernel_size=(3,), stride=(1,))
      (1): GLU(dim=1)
      (2): ConvTranspose1d(1600, 800, kernel_size=(8,),
stride=(4,))
```

```
      (3): ReLU()
    )
    (2): Sequential(
      (0): Conv1d(800, 1600, kernel_size=(3,), stride=(1,))
      (1): GLU(dim=1)
      (2): ConvTranspose1d(800, 400, kernel_size=(8,),
stride=(4,))
      (3): ReLU()
    )
    (3): Sequential(
      (0): Conv1d(400, 800, kernel_size=(3,), stride=(1,))
      (1): GLU(dim=1)
      (2): ConvTranspose1d(400, 200, kernel_size=(8,),
stride=(4,))
      (3): ReLU()
    )
    (4): Sequential(
      (0): Conv1d(200, 400, kernel_size=(3,), stride=(1,))
      (1): GLU(dim=1)
      (2): ConvTranspose1d(200, 100, kernel_size=(8,),
stride=(4,))
      (3): ReLU()
    )
    (5): Sequential(
      (0): Conv1d(100, 200, kernel_size=(3,), stride=(1,))
      (1): GLU(dim=1)
      (2): ConvTranspose1d(100, 8, kernel_size=(8,),
stride=(4,))
    )
  )
  (lstm): BLSTM(
    (lstm): LSTM(3200, 3200, num_layers=2, bidirectional=True)
    (linear): Linear(in_features=6400, out_features=3200,
bias=True)
  )
)
```