



Universidad
Internacional
de Valencia

Separación del ruido en una canción

Titulación:

Máster Universitario en
Inteligencia Artificial

Curso académico

2021 – 2022

Alumno:

Vilà Viñuela, Joan

D.N.I: 38876398G

Director/a de TFM:

Escrig Perez, Helio

Convocatoria:

Segunda

5 de septiembre de 2022

De:

 Planeta Formación y Universidades

Resumen

Este proyecto pertenece a la rama de la inteligencia artificial (IA) centrada en la separación de fuentes sonoras, concretamente a la separación de la música respecto a su ruido de fondo. El objetivo de este proyecto se centrará en el desarrollo y comparación de varios modelos para la eliminación del ruido de fondo de las canciones. Con ello se pretenderá ver cuál de estas técnicas es la que mejor se adapta al problema de separación de ruido o extracción de la música. Para lograrlo, primero se investigará los algoritmos y técnicas del orden del día, sobre todo las que usan el aprendizaje profundo.

Los modelos usados serán los que tienen una arquitectura VAE, AE y U-Net, todos ellos siguiendo el esquema *encoder-decoder*. Para cada diseño se elaborarán varios modelos con diferentes configuraciones en los valores de los hiperparámetros: *batch size*, tamaño del *latten space*, *learning rates*, *dropout* y *KL loss weight*. También se probarán con varios datos de entrada diferentes (características de audio), siendo estas los espectrogramas y Mel-espectrogramas.

En lo que respecta a las predicciones, se usará la técnica conocida como *direct mapping*, es decir, se intentará que los modelos predigan directamente los espectrogramas con su versión sin ruido, para después, mediante el uso de *Griffin-Lim* y la recuperación de la fase de los audios originales, poder generar el audio a partir de los espectrogramas predichos.

Se concluirá que el uso de *direct mapping* junto a las arquitecturas de red VAE, AE y U-Net no es la opción ideal, ningún modelo llegará a generar audios reconocibles. Sin embargo, el más prometedor será el U-Net con la obtención de los mejores resultados, probablemente debido al uso de *skip-connections*. Los que peores resultados darán serán los modelos basados en la arquitectura VAE, generando espectrogramas muy simplificados y borrosos, probablemente a su naturaleza y la incapacidad de generar buenos resultados desde un espacio latente gaussiano.

Palabras clave: Eliminación de ruido, separación de fuentes, codificador-decodificador, mapeo directo, redes convolucionales.

Abstract

This project belongs to the branch of artificial intelligence (AI) focused on sound source separation (SSS), specifically the separation of music from its background noise. The objective of this project will focus on the development and comparison of several models for the removal of background noise from songs. The aim will be to see which of these techniques is best suited to the problem of noise separation or music extraction. To achieve this, we will first investigate the algorithms and techniques of the order of the day, especially those that use deep learning.

The models used will be those with VAE, AE and U-Net architecture, all following the encoder-decoder scheme. For each design, several models will be developed with different configurations of the hyperparameter values: batch size, latent space size, learning rates, dropout and KL loss weight. They will also be tested with several different input data (audio features), these being spectrograms and Mel-spectrograms.

As far as predictions are concerned, the technique known as direct mapping will be used, i.e., the models will try to directly predict the spectrograms with their noise-free version, and then, by using Griffin-Lim and phase retrieval of the original audios, the audio will be generated from the predicted spectrograms.

It will be concluded that the use of direct mapping together with VAE, AE and U-Net network architectures is not the ideal option, no model will ever generate recognizable audios. However, the most promising will be the U-Net with the best results, probably due to the use of skip-connections. The worst performers will be the models based on the VAE architecture, generating very simplified and blurred spectrograms, probably due to their nature and inability to generate good results from a Gaussian latent space.

Keywords: Denoising, source separation, encoder-decoder, direct mapping, convolutional networks.

Índice

| | |
|--|-----------|
| Resumen | 1 |
| Abstract..... | 2 |
| 1. Introducción | 9 |
| 1.1. Estructura..... | 10 |
| 1.2. Justificación y elección del tema | 11 |
| 1.3. Justificación del título | 12 |
| 1.4. Descripción del trabajo y objetivos | 13 |
| 1.5. Estado del arte | 15 |
| 2. Metodología y planificación | 23 |
| 3. Marco teórico | 26 |
| 3.1. Conceptos básicos de una señal..... | 27 |
| 3.2. Propiedades básicas de la música | 27 |
| 3.3. Digitalización de una señal continua | 28 |
| 3.4. Características del audio..... | 29 |
| 3.4.1. Explicación de la transformada de Fourier | 30 |
| 3.4.2. Características en el dominio temporal | 33 |
| 3.4.3. Características en el dominio frecuencial..... | 35 |
| 3.4.4. Características en ambos dominios | 36 |
| 3.5. Explicación de los modelos usados..... | 39 |

| | | |
|-----------|--|-----------|
| 3.5.1. | Tipo de capas de los modelos | 40 |
| 3.5.2. | <i>Autoencoder y variational autoencoder</i> | 43 |
| 3.5.3. | U-Net..... | 48 |
| 4. | Desarrollo e implementación | 50 |
| 4.1. | Equipo, tecnologías y librerías usadas | 51 |
| 4.2. | Análisis del dataset | 52 |
| 4.3. | Preparación de los datos para el dataset final..... | 54 |
| 4.3.1. | Transformación de los datos..... | 54 |
| 4.3.2. | Selección y justificación de los parámetros de la STFT | 56 |
| 4.4. | Hiperparámetros de los modelos..... | 58 |
| 4.5. | Arquitecturas de los modelos | 59 |
| 4.6. | Funcionamiento del código y fases de implementación..... | 62 |
| 5. | Resultados y análisis | 65 |
| 6. | Conclusiones | 81 |
| 6.1. | Valoración y discusión de los resultados | 81 |
| 6.2. | Cumplimiento de los objetivos..... | 84 |
| 6.3. | Problemas y contratiempos | 85 |
| 6.4. | Recomendaciones para futuros trabajos | 87 |
| 6.5. | Aplicaciones extraordinarias..... | 91 |
| 7. | Referencias | 92 |
| 8. | Anexos | 99 |

| | | |
|------|--|-----|
| 8.1. | Anexo I: Explicación detallada de la FT y la creación de los espectrogramas .. | 99 |
| 8.2. | Anexo II: Explicación detallada de los Mel-espectrogramas | 107 |
| 8.3. | Anexo III: Explicación de las propiedades básicas de la música | 110 |
| 8.4. | Anexo IV: Requisitos y ejecución del programa..... | 112 |

Índice de figuras

| | |
|---|----|
| Figura 1: Diagrama de Gantt del proyecto | 24 |
| Figura 2: Diagrama de flujos de la planificación del proyecto..... | 25 |
| Figura 3: Tipos de funciones ventana | 32 |
| Figura 4: Operación convolucional..... | 40 |
| Figura 5: Movimiento del kernel durante una convolución..... | 41 |
| Figura 6: Tipos de pooling..... | 41 |
| Figura 7: Arquitectura simplificada de una VAE sin parametrization trick. | 46 |
| Figura 8: Arquitectura simplificada de una VAE con parametrization trick..... | 46 |
| Figura 9: Arquitectura de una U-Net con masking..... | 49 |
| Figura 10: Esquema de la implementación. | 50 |
| Figura 11: Arquitectura U-Net con la configuración original..... | 61 |
| Figura 12: Curva de aprendizaje del modelo AE-5..... | 67 |
| Figura 13: Muestras de espectrogramas predichos del modelo AE-5 | 69 |
| Figura 14: Curvas de aprendizaje de los modelos VAE-3 y VAE-4 | 71 |
| Figura 15: Muestras de espectrogramas predichos del modelo VAE-3 (KL loss alta).. | 75 |
| Figura 16: Muestras de espectrogramas predichos del modelo VAE-6 (KL loss bajo) | 75 |
| Figura 17: Curvas de aprendizaje de los modelos U-Net-1 y U-Net-2 | 77 |
| Figura 18: Muestras de espectrogramas predichos del modelo U-Net-2..... | 78 |
| Figura 19: Muestras de espectrogramas predichos del modelo U-Net-Mel-1 | 79 |

| | |
|---|-----------|
| <i>Figura 20: Muestra de las “waveform” de los audios originales y su predicción del modelo U-Net-2</i> | <i>80</i> |
| Figura 21: Muestra ampliada de la “waveform” de la predicción del modelo U-Net-2 .. | 80 |
| Figura 22: Arquitectura propuesta de input/output | 89 |
| Figura 23: Ilustración del plano complejo. | 100 |
| Figura 24: Ejemplo particular de un integrando gf | 101 |
| Figura 25: Integrando gf de frecuencia 1,1. | 102 |
| Figura 26: Forma de la suma de varias señales puras | 103 |
| Figura 27: Ejemplo de un espectro frecuencial. | 105 |
| Figura 28: Ejemplo de espectrograma | 105 |
| Figura 29: Filtros Mel | 108 |
| Figura 30: Formula detallada del ceptrum | 109 |
| Figura 31: Organización del proyecto de programación | 113 |

Índice de tablas

| | |
|---|----|
| Tabla 1: Distribución y peso de los subconjuntos train, validation y test del dataset ... | 55 |
| Tabla 2: Hiperparámetros probados del modelo AE | 58 |
| Tabla 3: Hiperparámetros probados del modelo VAE. | 58 |
| Tabla 4: Hiperparámetros probados del modelo U-Net | 59 |
| Tabla 5: Mejores resultados de los modelos con la arquitectura AE | 67 |
| Tabla 6: Mejores resultados de los modelos con la arquitectura VAE | 71 |
| Tabla 7: Mejores resultados de los modelos con la arquitectura U-Net..... | 76 |

1. Introducción

Este trabajo es un de investigación orientado a la industria, en él se ha querido aplicar algunas técnicas, algoritmos y modelos del mundo de la IA para la solución de un problema práctico, la separación o eliminación de ruido en una canción. La hipótesis es la siguiente: “Los algoritmos que usen una arquitectura del tipo *encoder-decoder* con capas convolucionales y sin el uso de bloques recurrentes usando la técnica de *direct mapping*, son capaces de eliminar el ruido de una canción desde sus espectrogramas”.

En ocasiones separar el ruido de fondo de las canciones es necesario. Hoy en día se han conseguido ya muy buenos resultados, sobre todo por lo que respecta a la separación de la voz e instrumentos musicales, no obstante, aún estamos lejos de encontrar el modelo perfecto, aquel que sea capaz de separar los sonidos deseados en varias fuentes diferentes de forma ideal, es decir, sin la introducción de artefactos acústicos o la fuga de sonidos provenientes de otras fuentes.

En este trabajo se ha investigado sobre varios modelos basados en redes neuronales convolucionales con la arquitectura *encoder-decoder* con tal de poder afrontar el reto e intentar progresar en camino de encontrar el modelo ideal.

Una vez realizada la investigación, se seleccionarán aquellas arquitecturas y técnicas que se hayan creído más convenientes para la resolución del problema, la separación del ruido. Varias configuraciones de hiperparámetros serán probadas, una por cada modelo. Una vez realizados los experimentos y obtenido los resultados, estos serán comparados y evaluados para observar que modelo ofrece el mejor rendimiento y resultado. Las arquitecturas que seleccionarán son: los *autoencoders*, los *variational autoencoders* y las *U-Nets*.

El ruido de fondo, o ambiente, utilizado para su separación será de dos tipos diferentes. El primero de ellos corresponderá al ruido que nos podemos encontrar en una ciudad, mientras que el segundo, será el ruido ambiente que nos podemos encontrar en una cafetería estándar. En ellos se podrán encontrar sonidos tan peculiares como: los del tráfico generado por los vehículos circulando, gente hablando, golpes de platos, sirenas de emergencia, entre otros.

Por otro parte, la música usada será extraída de un conjunto de datos públicos dividido en varias pistas (*stems*), una por cada tipo de instrumento. Para crear el audio final se combinarán todas las pistas en una sola y estas serán combinadas con los ruidos mediante la suma de sus valores. Las canciones con las que se entrenarán los modelos serán principalmente de los géneros musicales rock y pop.

Para la extracción de características, se empleará la STFT y con ella se obtendrán los diferentes *inputs* probados en los modelos, el espectrograma y el Mel-espectrograma.

1.1. Estructura

Este informe se ha organizado en seis bloques principales. A continuación, se describe brevemente en qué consiste cada uno y se nombra cuál será su contenido.

PRIMER BLOQUE, se describirá superficialmente en qué consiste este trabajo, cuál ha sido la motivación de su elección y su justificación, así como la explicación del cambio en el título inicial. También se hará un breve estudio del estado del arte actual, nombrando y viendo en qué consisten algunos de los trabajos más recientes en este campo para poder ver desde donde partir y como proceder.

Finalmente, en este bloque se definirá el objetivo principal y secundarios del proyecto, estableciendo que se quiere lograr en cada uno de ellos.

SEGUNDO BLOQUE, se explicará que metodología, rigor y organización se ha seguido para realizar exitosamente el proyecto.

TERCER BLOQUE, es el bloque correspondiente al marco teórico. En él se analizará y explicará escuetamente las principales técnicas, modelos y características del audio, así como las empleadas en la implementación del proyecto. Se observará propiamente que es el audio, o sonido, y cómo podemos tratarlo computacionalmente.

CUARTO BLOQUE, corresponde a la explicación general de la implementación. Se explicará superficialmente el código usado, así como que tecnología, componentes y librerías se han usado para crear y entrenar los modelos. También habrá un párrafo

donde se explicará cómo debe de ejecutarse el código en caso de que se quisiera replicar algún experimento o continuar el proyecto.

En este apartado también encontraremos la explicación de como se ha creado el *dataset*, así como se verá de que están compuestos los datos de este, realizando un análisis sobre estos.

Por último, se mencionará los pasos que se han seguido en la extracción de las características para la generación de los *inputs* de los modelos y se verá las diferentes arquitecturas de estos junto con las diferentes configuraciones de hiperparámetros usadas.

QUINTO BLOQUE, se mostrará y analizarán los resultados obtenidos de los diferentes modelos.

SEXTO BLOQUE, se redactarán las conclusiones finales mediante un análisis del cumplimiento de los objetivos iniciales, los resultados obtenidos de los modelos usados, las equivocaciones cometidas en la selección de estos y sus hiperparámetros, una recomendación para futuras ampliaciones e investigaciones relacionadas con la presente, los problemas encontrados durante la elaboración del proyecto, explicando cómo se han resuelto, o en caso contrario, como se recomienda solucionarlos para futuros trabajos y una pequeña autocrítica global personal respecto al proyecto.

1.2. Justificación y elección del tema

La motivación de este proyecto viene dada principalmente por dos razones. La primera es la voluntad de aprender a rasgos generales como se aplica la inteligencia artificial (IA) en el mundo de la música y sonido, viendo sus casos de uso en aplicaciones y técnicas para la obtención de características o la resolución de problemas.

La segunda motivación viene dada por la necesidad de, en casos cotidianos, poder separar las diferentes fuentes de sonido que puede contener un audio. En muchas ocasiones nos encontramos con audios donde solo se quiere escuchar un sonido muy particular, como, por ejemplo, una canción que suena de fondo, una conversación o simplemente saber cuántas personas están hablando o cuantos y qué instrumentos

hay en una canción sonando al mismo tiempo. Sin embargo, hay ocasiones en que esto no es posible o es muy molesto y costoso intentar focalizarse en el sonido deseado para poder aislarlo del resto y escucharlo correctamente.

Adicionalmente, siempre ha habido curiosidad y fascinación sobre cómo se trata el audio para poder crear programas como reconocedores de canciones, modificadores de voz, transcripción, etcétera. Al mismo tiempo, cada vez son más las aplicaciones de este tipo donde parecen ser más y mejores.

Es por ello que, en este proyecto, se ha querido desarrollar una aplicación o modelo que sea capaz de separar alguna fuente de un audio donde como mínimo tenga dos sonidos diferentes produciéndose al mismo tiempo.

Por último, como se verá a lo largo del estado del arte, dado que la literatura está muy centrada en la separación de la voz, a consecuencia del auge del *speech recognition* y *natural language processing*, así como el interés personal para separar la música del ruido ambiente y no el habla (la voz), se decidió centrar el proyecto en la separación de canciones respecto a un ruido de fondo casual, como por ejemplo, el generado en un restaurante por el murmullo de la gente o el generado en la calle por el ruido del tráfico.

1.3. Justificación del título

Si bien el título original del proyecto era: “*Separación de instrumentos en una canción instrumental*”, debido al poco conocimiento previo que se tenía sobre el tema y la libertad que se dejó en la elaboración del proyecto, siempre y cuando este siguiera la línea de investigación referente a la del título original, después de investigar los primeros conceptos básicos y artículos científicos relacionados, así como teniendo en cuenta la motivación principal del proyecto, separar una canción de su ruido ambiente, se decidió cambiar el título por uno más descriptivo del proyecto que finalmente se ha desarrollado: “*Separación del ruido en una canción*”.

El cambio ligero de nombre y proyecto respecto al previsto originalmente, se ha podido realizar sin problema debido a que el área de investigación, técnicas aplicadas y

modelos son los mismos, por ende, las conclusiones y resultados del estudio actual se puede pensar que serán parecidos a los que se hubieran obtenido del proyecto original.

El único cambio realizado es en el dominio y no el problema. Adicionalmente, en cambiar a un dominio más pequeño, separar una canción de su ruido ambiente (dos fuentes sonoras distintas) y no cada instrumento por separado (varias fuentes sonoras), nos brinda la oportunidad de realizar más experimentos debido a la simplificación del problema, siendo más fácil para los modelos alcanzar su objetivo en ser este de menor alcance.

1.4. Descripción del trabajo y objetivos

El presente trabajo consistirá en crear un modelo que sea capaz de eliminar el ruido ambiente que se puede encontrar en la grabación de una canción. Debido a tener solo dos fuentes de audio a separar, el ruido y la canción. La separación de una de estas fuentes es equivalente a su eliminación, dado que, una vez eliminada una de las dos fuentes, la otra puede ser generada simplemente restando al audio original la fuente no eliminada, en este caso, la canción. Es decir, el trabajo también podría haberse llamado: *“Eliminación del ruido (ambiente) en una canción”* en lugar de *“separación del ruido en una canción”*.

Puesto que para alcanzar dicha meta hay varias formas de supuestamente lograrlo, teniendo varios métodos, algoritmos y técnicas actuales que pueden solucionar el problema, se ha decidido centrarse solo en aquellas que para dicha tarea deben funcionar mejor, las basadas en aprendizaje profundo mediante el uso de redes neuronales [1]–[3].

Sin embargo, y debido a que dentro del campo en aprendizaje profundo (redes con varias capas ocultas) también hay muchos tipos de redes neuronales y arquitecturas diferentes, se seleccionarán solo aquellas que no sean muy costosas computacionalmente y que se crea que puedan dar resultados decentes en la eliminación del ruido. No se seleccionarán redes muy potentes, aunque tengan un

potencial mucho mayor debido a la limitación del equipo técnico para el entrenamiento de los modelos.

Para conseguir el objetivo, finalmente se usarán tres arquitecturas de red neuronal profundas con una estructura del tipo *convolutional encoder-decoder* (CED) donde sus datos de entrada serán un espectrograma o un Mel-espectrograma. De esta forma, en ser los *inputs* de las redes una imagen, nos permitirán el uso de bloques convolucionales para que estas reduzcan y vayan extrayendo patrones a un coste computacional mucho más bajo que cualquier otro tipo de red neuronal más compleja como las que usen algún tipo de bloque con recurrencia.

Los tres tipos de red seleccionadas serán las *variacional autoencoders* (VAE), *autoencoders* (AE) y U-Net. Esta selección es a consecuencia de la reputación de dichas arquitecturas (tipo de red), puesto que, después de las recurrentes, son las que parecen tener mejor resultado, tal y como se verá en el estudio del arte del siguiente apartado, el 1.4) *Estado del arte*.

En resumen, el objetivo principal de este trabajo es:

- ***Eliminar el ruido ambiente de una canción mediante modelos de aprendizaje profundo.***

Para lograrlo, teniendo en cuenta la limitación de tiempo y tecnológica (*hardware*), se han marcado varios hitos para concretar el dominio. Estos son los siguientes:

- a) Implementar solo los modelos VAE, AE y U-Net y ver cuál funciona mejor habiendo una optimización de sus hiperparámetros más importantes.
- b) Reducir el tipo de canciones (géneros) y ruidos ambientes a unos pocos mediante la creación de una *dataset* personalizado. De otra forma, debido a que cada género y ruido tienen patrones diferentes, los modelos tardarían más en aprender.
- c) Implementar un sistema para que, de los espectrogramas predichos, se puedan volver a transformar a audio para escuchar y analizar el resultado final, viendo si finalmente, se ha logrado o no cumplir el objetivo principal.

Otros objetivos secundarios de este proyecto son:

- I) Obtener una primera idea o intuición de cómo se aplica la inteligencia artificial (IA) al campo de la música y/o habla.
- II) Elaborar un primer proyecto de IA de principio o fin, aplicando y consolidando algunos de los conceptos vistos durante el máster. Sobre todo, los relacionados con las redes neuronales y la extracción de características útiles para estas.

1.5. Estado del arte

Desde la aparición del audio digital, incluso en algunos casos el analógico, siempre se ha buscado alguna forma para modificarlo, ya sea para crear nuevos sonidos, como el uso de los sintetizadores, o para eliminar ruidos y frecuencias no deseadas, como, por ejemplo, los pitidos agudos generados muchas veces por interferencias eléctricas.

Con el creciente mundo de la inteligencia artificial (IA), sobre todo en la rama del aprendizaje profundo (en inglés *deep learning*, DL), el basado en redes neuronales profundas, el potencial de poder modificar el audio con estas nuevas técnicas ha alimentado y reforzado la modificación del audio, siendo actualmente uno de los momentos con más auge.

Antes de la llegada de los algoritmos basados en redes neuronales (NN), para la modificación del audio y eliminación de ruido se utilizaban técnicas básicas como los ecualizadores [3]. Estos se basaban en la transformada de Fourier para poder aplicar filtros paso bajos y altos al audio para así poder cambiar la amplitud de algunas frecuencias y también el valor de las fases (cancelación de fases) para, por ejemplo, generar nuevos sonidos o eliminar las frecuencias no deseadas [4].

Otras técnicas ampliamente usadas eran y son, trabajar directamente sobre audios con información MIDI, como, por ejemplo, aquellos que están gravados en varias pistas (*stems*) donde cada una corresponde a un instrumento diferente [3]. Sin embargo, el problema en este caso es que muchas veces no se tiene el archivo MIDI o las pistas de audio separadas, considerando el ruido como una pista adicional.

Otros algoritmos tradicionales de ML aplicados a la separación del audio son: el *Gaussian Mixtures*, ICA (con o sin el uso de PCA), ISA, NMF, NTF, CAM y otros [5], [6]. Todos ellos se basan en la aplicación de diversas fórmulas matemáticas para intentar separar las diferentes fuentes sonoras, usando para ello la amplitud de la señal recibida, o en su defecto, su espectro de frecuencias haciendo uso de la STFT para extraerlo [5]. Ninguna de estas técnicas va más allá de aplicar de forma inteligente diferentes fórmulas matemáticas, que, si bien son rápidas de calcular, y, por ende, separar las señales, se necesita independencia entre las diferentes señales, así como que hayan sido capturadas en varios puntos geográficos con la ayuda de varios micrófonos [5], [6]. Estos requisitos hacen que muchas veces sea inviable la aplicación de estas técnicas.

Actualmente, hay aplicaciones como *Audacity* que tienen integrados *software* de cancelación de ruido. Sin embargo, muchos de ellas simplemente intentan restar del audio un ruido gaussiano o aplicar filtros (o atenuadores) de altas y bajas frecuencias para la eliminación de ruido [3]. Aunque estas técnicas den un resultado aceptable, están lejos de ser perfectas.

Si bien estos programas acaban consiguiendo eliminar el ruido original casi en su totalidad, debido a que son técnicas destructivas, no generan audio nuevo (como si hacen las técnicas constructivas como las que aplican redes neuronales profundas), acaban también por eliminar o atenuar información original deseada, haciendo que en el resultado final aparezcan nuevos ruidos artificiales (artefactos) no deseados, aunque menos molesto que el ruido original. Asimismo, cuando se aplican filtros, las frecuencias que caigan dentro de su rango, tanto si son frecuencias pertenecientes al ruido como si no, quedan atenuadas [3]. Esta atenuación sigue permitiendo pasar algo de ruido en la señal resultante, así como modificar ligeramente el audio original deseado.

Al mismo tiempo, programas más sofisticados y específicos como pueden ser: *Moises.ai*, *Lalal.ai*, *Spitter.ai*, *Xtrax Stems*, *RX 7*, entre otros, tienen implementados *softwares* basados en modelos que usan redes neuronales para la eliminación de ruido, sin embargo, la mayoría de estos suelen tener implementados los mismos [3]. Algunos de los más comunes y populares son: *Spleeter*, *Cassiopeia*, *Rocknet*,

Demucs, *OpenUnmix* y otros [7]–[10]. Todos ellos usan las arquitecturas de redes neuronales más usadas en el estado del arte actual.

Para la eliminación de ruido en una señal se han encontrado algunos artículos que usan solamente la estadística e interpolación, no se aplican técnicas de ML. Se usa en casos donde el ruido que se encuentra ocupa una unidad de tiempo muy pequeña, como, por ejemplo, un corte (silencio pequeño), un golpe, o similares [11].

Aunque aparentemente este estudio puede no parecer relevante para la investigación actual de este proyecto, debido a que el sonido de fondo de la música es constante a lo largo del tiempo, podría ser interesante aplicarlo una vez se haya reconstruido la señal sin ruido en caso de quedar algunos artefactos auditivos no deseados y de poca duración para acabar de pulir el resultado final.

Otros artículos vistos y estudiados como él [12], usan varias características del audio como entrada a los modelos o redes neuronales en lugar de una sola. Algunas de estas son los MFCC, Mel-espectrogramas, espectrogramas *vanilla*, *Constant-Q Transform* y *energy normalized statistics* (ENS).

El uso de varias características al mismo tiempo es debido a la gran dificultad que supone para los algoritmos y humanos poder separar el ruido del audio. La idea es que cuanta más información tengan los algoritmos, más fácil pueden reconocer los patrones y separar audios, aunque requiera un tiempo de entrenado mucho mayor.

Del artículo anterior [12] es interesante fijarse en la estrategia seguida, consistiendo en resumir la gran cantidad de características que se le pasa al modelo. Para ello, primero se tiene un bloque convolucional para generar de forma más compacta e informativa nuevas características, extrayendo de forma automática solo la información útil y necesaria. Una vez extraídas las nuevas características se las pasa a la segunda parte del modelo, la parte más densa y costosa de entrenar, la encargada de solucionar el problema a tratar como la separación de fuentes. Con los datos menos pesados, el entrenamiento es menos costoso y se reduce así su tiempo de aprendizaje e información redundante que puede empeorar las predicciones finales.

Por otro lado, del estudio [12] también es interesante fijarse en su objetivo, que el modelo sea capaz de identificar el tipo de ruido (gaussiano, de tráfico, obras, etc.) que le es pasado como *input*. Una vez identificado el tipo de ruido, se pueden aplicar técnicas de supresión de ruidos específicas para cada tipo y así poder aumentar el rendimiento de las predicciones de los modelos.

Sin embargo, cabe mencionar que características como las MFCCs y *Constant-Q Transform* no suelen usarse cuando se usan redes neuronales o modelos complejos. Estas características deben ser evitadas, ya que eliminan información importante como la relación espacio-temporal [6]. Las más idóneas, y, por tanto, las más usadas son las que si tienen en cuenta esta relación como pueden ser las señales de audio en crudo (*waveform*), los espectrogramas o los Mel-espectrogramas.

Estudios recientes han investigado la separación de diferentes fuentes de audio provenientes de una misma señal (sonido), ya sea separando el ruido de fondo del habla [1], [6], [13]–[18], los diferentes instrumentos que están tocando [19]–[21] o simplemente comprimir el audio [22]. Sin embargo, y dado que en los últimos años el lenguaje natural (NLP) está en auge, así como la necesidad de traducir de forma automática las conversaciones de las personas a texto para después poder procesarlo, ya sea para NLP o simplemente para, por ejemplo, tareas de traducción o clasificación y detección de tópicos hablados en una conversación, hace que casi todos los estudios se centran únicamente en tres tipos de separación, tal y como se observará a lo largo de este apartado.

Las diferentes separaciones son: 1) la separación del habla (*speech denoising*), 2) la separación de instrumentos en una canción en diversas pistas, una por instrumento y 3) la clasificación de música por géneros musicales, siendo estos dos últimos tipos de separación probablemente para agilizar la clasificación automática de música y reconocimiento de canciones debido a aplicaciones como *Spotify*, *Shazam* o *Soundhound* y el incremento exponencial de creación de música, géneros y artistas nuevos en los últimos años junto con la necesidad humana de tenerlo todo etiquetado.

Estos estudios son interesantes de analizar debido a que, como veremos, la mayoría de ellos acaban usando las mismas arquitecturas de redes y estrategias con pequeñas variaciones para adaptarse al problema actual, esto es así debido a su gran similitud

en la raíz de problema, tratar con audio para realizar algún tipo de separación. De este modo, el conocimiento extraído de estos estudios resultará útil para obtener una idea de cómo afrontar el problema de este proyecto. Recordamos que el problema actual trata de aislar, o separar, canciones de su ruido ambiente.

El ruido y la música se pueden ver como una pista o instrumento de audio adicional, teniendo así un total de dos fuentes o pistas de audio a separar. Otras de las extracciones que se pueden hacer de estos estudios es la aplicación de sus arquitecturas y la selección de las mejores características del audio para obtener los *inputs* del modelo.

Los algoritmos de ML aplicados a la separación de audio aprenden a identificar patrones de como suena cada fuente para, después, poder separar cada fuente por separado. En la separación del audio encontramos dos grupos de modelos principales, los basados en la eliminación espectral (deconstructivos) y los basados en la generación espectral (constructivos), estos últimos divididos en otros dos grupos según las técnicas usadas, los que usan un mapeo directo (*direct mapping*) y los que usan máscaras para la generación del espectrograma objetivo (*masking*) [21].

Los del primer tipo, suelen ser los algoritmos más tradicionales que no usan redes neuronales, como, por ejemplo, la aplicación de filtros. Su forma de proceder normalmente suele ser identificar las frecuencias que provocan el ruido más intenso para después, poder aplicar la transformada de Fourier (STFT) y restar estas frecuencias directamente de la señal para eliminarlas. Para ello, el uso de filtros pasa bajos y altos suele ser habitual, sobre todo cuando el sonido que se quiere aislar es la voz, donde sus frecuencias se centran mayoritariamente en las intermedias [2], [6].

Si bien los algoritmos basados en este tipo de separación son buenos aislando ruidos estáticos, ruidos que no cambian mucho, tienen una varianza muy baja a lo largo del tiempo [2], como puede ser los pitidos o el ruido gaussiano generado por interferencias o en entornos como el ruido de un río o el motor de un avión, no son algoritmos que suelen funcionar bien con sonidos dinámicos o cambiantes, donde, por infortunio, suelen ser los más comunes cuando tratamos con pistas de audio.

Aun así, hay técnicas más avanzadas que permiten ser más eficientes en estos casos. Una de las más conocidas y empleadas es la llamada *Wiener filtering* (un tipo de filtro

dinámico), siendo el actual estándar para muchas aplicaciones y dispositivos como móviles, audífonos, etc. [2].

Por lo que se ha observado, la *Wiener filtering* también se puede usar como un modelo a comparar con otros, sirviendo de *baseline* para evaluar el rendimiento y resultados (posibles mejoras) de, normalmente, modelos más complejas como las redes neuronales, tal y como se hace en el estudio [19].

La idea básica que hay detrás de este filtro es asumir que el ruido es dinámico, y, por ello, va cambiando de forma automática en función del ruido y sonido que va detectando. Para ello hace uso de dos micrófonos, uno para captar la señal en su conjunto y el otro para captar con más potencia el sonido a aislar [1]. No obstante, en caso de no tener dos captadores de audio o no poder captar uno de los dos audios a separar con más potencia que el otro, este filtro no se podrá aplicar [1], [21]. Por ese motivo, tanto este filtro como las técnicas tradicionales de ML quedan descartadas para el proyecto actual.

Volviendo a los tipos de modelos, ahora los de naturaleza constructiva, una de las técnicas más comunes en la separación de fuentes (*source separation*) son los llamados *direct mapping* y *time-frequency masking* [3], [23]. Estos normalmente tendrán como *input* el espectrograma o incluso, en algunos casos, la señal original sin tratar (*waveform*) y suelen ser los usados por los modelos más recientes con redes neuronales, siendo mejores en la separación de ruido no estático debido a su no linealidad [21].

Direct mapping y *masking* se diferencian entre ellos con el *output* que deben generar. Mientras que los basados en *masking* tienen como objetivo, encontrar o aprender una máscara o más (en función del número de fuentes a separar que se deseen) para que cuando esta se multiplique con la señal original quede separada en diferentes fuentes de forma automática, los basados en *direct mapping* tienen como objetivo estimar directamente las nuevas fuentes, sin el paso intermedio de la creación de máscaras [21].

De toda la literatura estudiada, solo [6], [13], [15], [22] han usado *direct mapping*, los demás han usado *time-frequency masking* [1], [14], [16]–[21]. Adicionalmente, todos

los que han usado como arquitecturas algún tipo de *encoder-decoder* han usado la técnica de *masking*.

Por lo que respecta a las arquitecturas más usadas, los artículos acabados de mencionar han empleado el tipo de arquitectura U-Net [24], ya sea en su versión más estándar, empleada en los estudios [20], o con ligeras modificaciones como el uso de *multi-heads* [19] o con la implementación de bloques recurrentes tipo LSTM, GRU y CRN [1], [13], [18], [21], [22], incluso algunos de estos usando los llamados BLSTM (bidireccionales) [16], [17], [21]. Estos últimos casi todos basados en el modelo Unmix [8]. Otro tipo de arquitecturas usadas han sido las basadas en redes convolucionales como las CED, CDAE y VAE [6], [13]–[17], [22].

Muchos de los estudios han intentado combinar capas convolucionales y bloques recurrentes. Algunos de estos simplemente han intentado comparar estos dos tipos de arquitectura [22], demostrando que, en algunos casos, tanto las que tienen alguna recurrencia como las que solo usan convoluciones, tienen rendimientos muy similares, siendo, sin embargo, las convolucionales las que, como es lógico, debido a su menor complejidad, han tenido un tiempo de entrenamiento mucho más veloz.

Respecto al rendimiento entre las redes que usan solo convolución (VAE y AE), las que usan solo una arquitectura tipo U-Net y las que usan U-Net con algunos bloques recurrentes, se ha observado comparando los diferentes estudios encontrados que con el mismo orden que se han nombrado, cada una de ellas va mejorando respecto a la anterior. No obstante, debido a su aumento en complejidad, cada vez se tienen que tener más datos para entrenar, y, incluso, con la misma cantidad de datos se requiere mucho más tiempo de aprendizaje.

En resumen, los artículos más recientes centrados en el problema de la separación del audio, ya sea del habla, música u otros, casi todos aplican redes neuronales para su resolución, teniendo todas ellas arquitecturas e *inputs* muy comunes. Las arquitecturas más comunes encontradas son: *Variational autoencoders* (VAE), *Autoencoders* (AE), *Generative adversarial networks* (GAN), redes recurrentes como la LSTM, BLSTM, GRU y, sobre todo, la U-Net, así como los *inputs* son principalmente los espectrogramas, Mel-espectrograma o el uso de ambos de forma simultánea.

Después de haber analizado los artículos del estado del arte, se han extraído los modelos y configuraciones (estas para una referencia inicial) para probar de solucionar el problema de separación del ruido ambiente y la música. Así pues, el proyecto actual se ha centrado solo en los que tienen una arquitectura del tipo *encoder-decoder*, quedando descartadas las GAN y las redes recurrentes debido a la falta de tiempo para explorar nuevas estructuras como las GAN o equipamiento tecnológico como para poder entrenar redes recurrentes en un tiempo medianamente asumible.

Los modelos usados han sido las VAE, AE y U-Net simples, sin bloques recurrentes. Sus configuraciones y justificación se mencionarán en secciones posteriores, la 1.4) *Descripción del trabajo y objetivos* y la 4.5) *Arquitecturas y modelos usados*, siendo la justificación principal su sencillez y rapidez de entrenamiento debido a la falta de potencia en equipo técnico disponible para el entrenamiento de los modelos.

2. Metodología y planificación

Para que cualquier proyecto no esté destinado al fracaso es necesario tener una buena organización y planificación. Para ayudar a tener las dos características anteriores existen varias metodologías que nos ayudan a conseguirlo. Una de las más famosas, debida a su flexibilidad, es la metodología AGIL, donde uno de sus marcos de trabajo (*frameworks*) más comunes son SCRUM y KANBAN [25], [26].

En este proyecto, sin embargo, dado que no se está trabajando en un equipo de personas, es individual, no tiene demasiado sentido aplicar ninguna de estas metodologías. No obstante, se ha usado algunas de sus características para ayudar en la planificación y organización de este proyecto.

Una de las principales diferencias es la eliminación de reuniones periódicas, ya que, en este caso, como mucho se podrían haber hecho con el tutor del TFM, cuya única finalidad es la de intentar guiar al alumno a realizar el trabajo, pero no contribuir en este. De este modo, aunque esporádicamente puede haber habido alguna reunión con el tutor, sobre todo al principio para la elección del tema y como enfocarlo, así como al final para la redacción de la memoria, la comunicación principal ha sido a través de correos electrónicos para la resolución de dudas en la estructura del trabajo y sobre todo en la memoria.

Para mantener un orden y poder cumplir con las fechas de entrega, al principio se realizó un diagrama de Gantt en el que se indicaron los grandes apartados del trabajo y cuanto debían durar y ejecutarse estos mismos. Marcando para cada uno un inicio y un fin. Para cada bloque, aunque esta vez sin fechas estrictas, dado que a priori no se sabía cuánto iba a durar cada sección del proyecto, se definieron varias metas o hitos a desarrollar, ilustradas en la figura 1, la correspondiente al diagrama de Gantt.

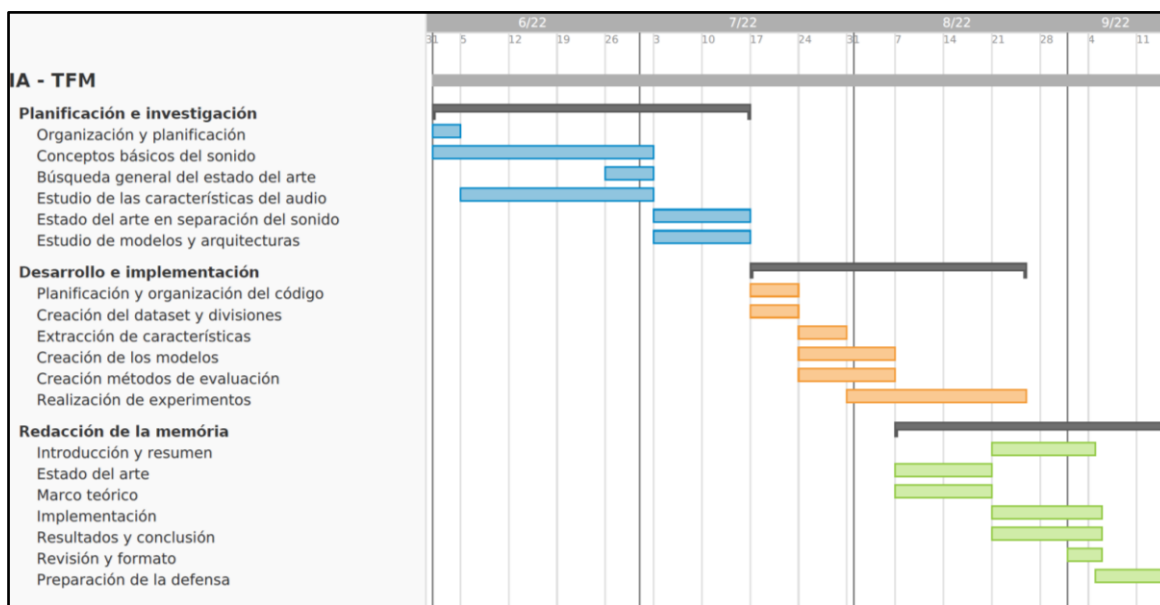


Figura 1: Diagrama de Gantt del proyecto. Elaboración propia mediante teamgantt [27].

Como se puede observar, cada columna (espacio entre las líneas verticales) representa una semana y las de color resaltado (en negrita) un mes. Del mismo diagrama, vemos que el proyecto tuvo una duración total de 3 meses (de junio a septiembre), donde la primera fase, la marcada en color azul, duro aproximadamente la mitad del proyecto. Esto fue debido principalmente a dos motivos: 1) Se tenía menos tiempo que en las siguientes fases debido a que aún había clases y exámenes; 2) Es la parte más importante y difícil, ya que en ella había que planificar y organizar bien todo el proyecto, así como documentarse y entender el dominio de trabajo, para después, poder desarrollar con cierto orden de conocimiento (sobre todo para la configuración de hiperparámetros) y evaluar correctamente los resultados.

Adicionalmente, para tener un mejor control del tiempo, se definió un horario de trabajo donde se estableció trabajar unas 3 horas diarias entre semana y una media de 8 horas diarias los fines de semana durante los meses de julio y septiembre. El mes de junio solo se trabajó 6 horas diarias los fines de semana y 1 hora diaria entre semana. El horario se tuvo que limitar de esta forma debido al trabajo laboral y académico (clases y exámenes), sobre todo durante el mes de junio y principios de julio

Haciendo los cálculos nos da una aproximación de 272 horas, más o menos a las 300 horas que corresponden a los créditos que tiene el trabajo de fin de máster, tomando como referencia las 25 horas por crédito establecidas por el marco europeo.

Para ayudar a la comprensión del desarrollo total del proyecto, a continuación, se muestra un diagrama donde se observa el flujo trabajo seguido durante la elaboración de este.

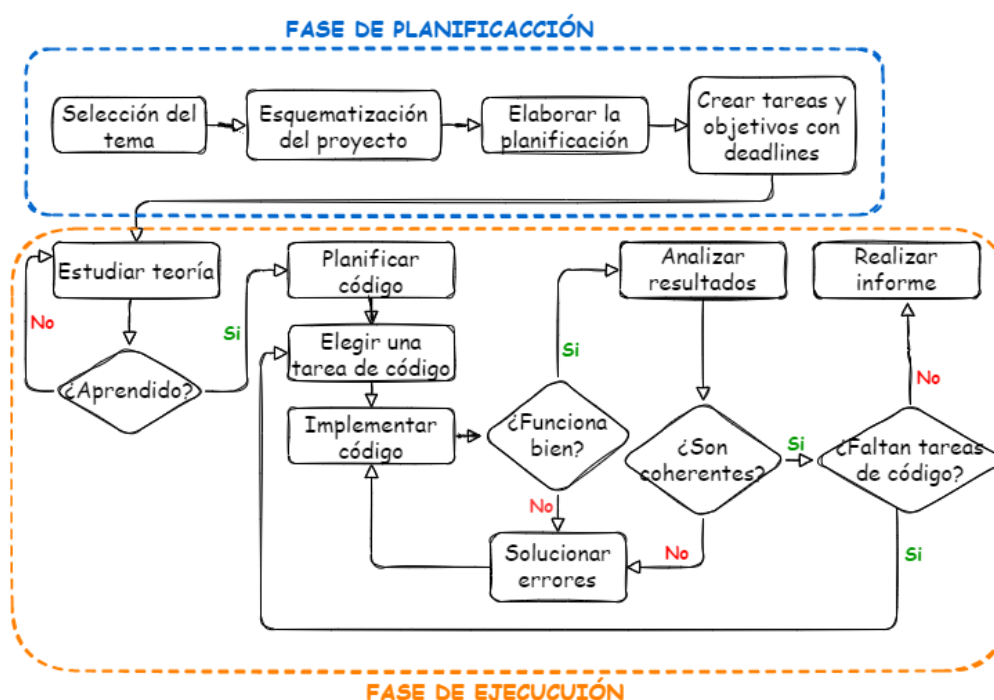


Figura 2: Diagrama de flujos de la planificación del proyecto. Elaboración propia mediante diagrams [28].

Finalmente, para tener un control de versiones del trabajo, se ha utilizado GitHub tanto para el código como para la memoria escrita. En lo que respecta a la parte de investigación, el *software* llamado *Mendeley* ha sido empleado para ir archivando todos los artículos científicos, blogs y videos usados durante la realización del proyecto para su estudio del estado del arte, parte de programación y conceptos teóricos. Para la implementación del código se ha utilizado el IDE PyCharm y el editor de código Sublime Text.

3. Marco teórico

En el aprendizaje automático es de vital importancia tener buenos datos. Idealmente, deben estar correctamente balanceados, regularizados, estandarizados, ser lo máximo explicativo posible, no tener correlación entre ellos para evitar redundancias, etcétera.

Todas estas características son extremadamente importantes principalmente para dos razones:

- 1) Los algoritmos empleados solo aceptan un formato de entrada de longitud invariante y numérico.
- 2) Facilitar el aprendizaje y convergencia de los algoritmos, así como poder generalizar a datos no vistos durante la fase de entrenamiento o aprendizaje.

Dependiendo del problema en el que nos encontremos, los datos se podrán obtener de forma directa o deberán ser generados a partir de otros más pobres y simples.

Por lo que respecta a las ondas acústicas o audio, todas sus características y datos son generados a partir de la forma de la señal a lo largo del tiempo (*waveform*), es decir, la variabilidad de su amplitud o intensidad.

A lo largo de este apartado, en diferentes secciones y de forma superficial, se explicará los conceptos básicos de las señales o sonidos, sus atributos o componentes de las que están formadas y las principales características que se pueden obtener. De las principales características también se justifican las usadas para este trabajo.

Todos los conceptos y teoría han sido extraídas principalmente de una serie de videos instructivos de un minicurso [29] y dos videos independientes [30], [31], así como corroborado del conocimiento previo que se tenía de la carrera, varios blogs y páginas web [32]–[42] y algunos artículos científicos [43]–[46]. El valor de los hiperparámetros más comunes y utilizados en este proyecto se han corroborado con los usados en cada uno de los estudios vistos en el estado del arte y en las propias referencias acabadas de mencionar.

Para la ayuda y referencia en la redacción de los apartados correspondientes a las características del audio en todos sus dominios, propiedades, digitalización y algoritmos que se usan, se han utilizado principalmente los artículos [38], [40] y el minicurso [29].

3.1. Conceptos básicos de una señal

Una señal de audio pura o simple es aquella que tiene una única frecuencia. Matemáticamente, se formula con la ecuación $s(t) = A \cdot \sin(2\pi \cdot (ft + \phi))$.

- A: La amplitud, volumen, intensidad o energía. Su unidad estándar de medida estándar es el metro y sus derivados (ej.: mm, cm).
- f: La frecuencia o vibraciones (periodos completos) por segundo. Su unidad de medida estándar es el Hz o 1/segundo.
- t: El tiempo. Su unidad de medida estándar es el segundo y sus derivados (ej.: fs, ps, us, ms).
- ϕ : La fase, desplazamiento o posición inicial. Su unidad de medida estándar son los radianes por segundo (rad/seg).

Una señal compleja se definirá como la suma de dos o más señales puras. Un sonido cualquiera normalmente estará compuesto por una infinidad de señales simples. A partir de ahora, siempre que se haga referencia al nombre señal o sonido será sinónimo de una señal compleja.

3.2. Propiedades básicas de la música

La música está formada por uno o más instrumentos, incluida la voz, sonando al mismo tiempo o en *solo* (por separado). Cada instrumento generará la melodía de la música con la combinación de varias notas (tonos) sonando o no al mismo tiempo. Físicamente y hablando simplistamente, cada nota es la composición de una frecuencia o señal pura más otras que son múltiplos de esta, los llamados armónicos.

Dependiendo de los armónicos que suenen estaremos hablando de un instrumento u otro.

Independientemente de las notas, la música tiene otras propiedades que hacen distinguir una canción de otra o simplemente, una interpretación u otra. Estas propiedades son: la frecuencia, el timbre y la intensidad. Quedan explicadas en el anexo III: *Explicación de las propiedades básicas de la música*, en él también se definen las métricas que usan y el porqué de ellas, como, por ejemplo, el uso de los decibelios (dB).

3.3. Digitalización de una señal continua

Dado que digitalmente no se puede trabajar con valores continuos, ya que esto requiere de un espacio de memoria y procesamiento infinito, en la práctica se muestreará la señal para tener una representación suya en formato digital. Este muestreo consiste en tomar el valor o amplitud que toma la señal cada cierto tiempo determinado, técnicamente es llamado tasa de muestreo o, en inglés, *sampling rate* (SR). Observamos entonces que una señal digital es básicamente un vector numérico de longitud $S_r \cdot t$, siendo t la duración en segundos del sonido o señal.

Para una correcta SR recurriremos a la llamada frecuencia de Nyquist (f_N). Esta nos permite obtener un valor de SR que nos permita reconstruir la señal a un espacio continuo sin la introducción o generación de artefactos, ruidos o alteraciones sonoras no deseadas e inexistentes en la señal original. La f_N define la frecuencia máxima a la que se puede reconstruir una señal dada una SR determinada sin que se generen artefactos y es calculada con la fórmula: $f_N = S_r/2$. Despejando la SR de la fórmula vemos que la tasa de muestreo deberá ser justo el doble de la frecuencia máxima deseada.

Los artefactos, también conocidos con el nombre de *aliasing*, físicamente se manifiesta en cuanto la señal reconstruida tiene una frecuencia menor a la original debido a la falta de muestras para poder realizar una correcta interpolación. Todas las frecuencias originales mayores a la de Nyquist se verán afectadas por este fenómeno. Para ello, normalmente antes de muestrear se aplica un filtro pasa bajos donde solo se dejarán las frecuencias más bajas que la f_N , evitando así el *aliasing*.

Finalmente, para pasar de continuo a discreto, se tendrá que tener en cuenta otro tipo de muestreo, el llamado muestreo en amplitud o cuantización. Este muestreo consiste en guardar la información de la amplitud de la señal en el tiempo. Para ello, se calculará el rango entre la amplitud mínima y máxima de la señal en un periodo (rango dinámico), o en el caso de una canción, lo que dure esta, dado que en una canción la señal no se repite en ningún momento, es decir, el periodo será toda ella.

Una vez calculado el rango, este se dividirá en n fragmentos equidistantes y para cada valor continuo en un intervalo de tiempo Δt , marcado por la SR, se establecerá su valor al que le corresponda del fragmento más cercano. La unidad de medida en este caso será el bit y no el Hz, donde en función del número de bits se tendrán más o menos fragmentos. Técnicamente, al número de bits se le llama *bit depth*, donde, normalmente, suele ser de una resolución de 16 bits, dando como resultado 2^{16} fragmentos o valores distintos a los que poder discretizar.

3.4. Características del audio

Las características referentes al audio o señales que se pueden usar en los algoritmos se pueden dividir en tres grupos principales, las que trabajan directamente sobre el dominio temporal (la señal de audio en crudo o forma de la señal), las que trabajan en el dominio frecuencial y las que trabajan en ambos dominios.

Sin embargo, antes de poder empezar a definir por encima cada una de las características principales, sobre todo las que se basan en el dominio frecuencial, para poder entenderlas se necesita describir en que consiste la transformada de Fourier (FT) y la transformada corta de Fourier (STFT).

3.4.1. Explicación de la transformada de Fourier

Para poder pasar al dominio frecuencial, usaremos la transformada de Fourier (FT), o más bien la transformada de Fourier discreta (DFT), la FT aplicada a las señales discretizadas. La DFT nos permite pasar de un dominio temporal a uno frecuencial, mientras que la DFT inversa o IDFT nos permite hacer el proceso inverso para poder recuperar la señal original.

Un espectro de frecuencias nos indicará para cada frecuencia por debajo de la de Nyquist, marcada por la tasa de muestreo (SR), la cantidad de energía que tiene la señal a la que se le ha aplicado la transformada, siendo esta una de las características más importantes que se pueden extraer del audio. Por cada frecuencia, se aplica una FT para calcular la energía de la señal en dicha frecuencia.

La STFT, no es más que la aplicación de una FT múltiples veces para calcular el espectro de frecuencias por cada periodo o unidad de tiempo continuo. La STFT devuelve un valor complejo, donde la magnitud es el valor absoluto de este y la fase el ángulo. El conjunto de magnitudes nos devuelve probablemente, junto con la *waveform*, la característica más importante que se puede extraer del audio, el espectrograma, explicado más adelante.

Parámetros de la transformada de Fourier

Frame size (n_fft)

Es el valor que indica la longitud de cada unidad de tiempo, en otras palabras, cuantas muestras del muestreo de la señal (SR) serán tenidas en cuenta para la aplicación de una FT, y, por ende, por la frecuencia de Nyquist, limitará el número de transformadas a calcular, de aquí a que también muchas veces se represente (referencie) a ella por el nombre de *n_fft*.

El valor del *frame size* habitual suele ser de 512 a 2048. Estos valores son debidos principalmente a tres causas: 1) la SR habitual suele ser de 22 KHz o 44 KHz, 2) la resolución temporal del oído humano es de 10 ms [47], cualquier evento sonoro interior

a 10ms no será posible distinguirlo, y 3) el cálculo de la duración de un *frame* viene dado por la formula $d_f = 1/S_r * K$; $K = \text{num. frames}$.

Spectral leakage y windowing

Es el fenómeno causado por el hecho de que dada una selección aleatoria de un cierto número de *frames* consecutivos, un *bin* temporal, nunca, o casi nunca acabe en la misma amplitud o posición en el eje de abscisas. Esta casuística crea discontinuidades que se traducen como la aparición de componentes de alta frecuencia, artefactos en el espectro frecuencial, que no estaban presentes en la señal original, frecuencias inexistentes.

Para paliar este efecto aplicaremos la técnica llamada *windowing* (en español ventana) en cada *frame*. Este procedimiento consiste en la multiplicación de una función matemática, la llamada ventana, por la señal original, los valores del *frame*. Con dicha operación se elimina, o atenúa, los valores de los extremos, haciendo de forma artificial que los valores de la señal para cada *frame* empiecen y terminen en cero.

Las funciones ventana más típicas son: la rectangular, la triangular, la de Hann, la de hamming, la gaussiana, la blackman, entre otras. Siendo las dos primeras las que suelen dar el rendimiento más pobre de todas, y la de Hann la que suele dar resultados más suavizados y no tan abruptamente truncados como pueden dar otro tipo de funciones. Es por ello que, normalmente, la función de Hann es la empleada por defecto. Dependiendo del problema se elegirá una u otra función. La característica principal de dichas funciones es que en su centro tiene valor 1 y en los extremos su valor tiende a 0.

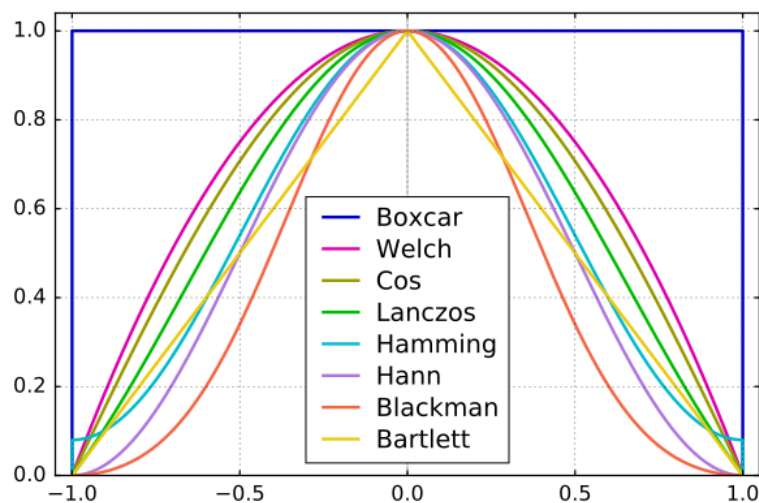


Figura 3: Tipos de funciones ventana. Extraída de [48]

El tamaño de la ventana (*window size*), normalmente suele ser el mismo que el tamaño del *frame size*, en caso contrario, se hará uso de *zero padding* para rellenar con ceros (hacia la derecha o izquierda) hasta igualar ambos tamaños.

Sin embargo, un problema surgido de aplicar esas funciones es la pérdida de información de los extremos del *frame*, información de la señal que puede ser útil. Para remediar este inconveniente se tiene otro parámetro que permite recuperar la información. Este parámetro es el conocido como *overlapping*, solapamiento o *hop size*.

Overlapping (hop size)

Consiste en seleccionar solo unas cuantas muestras nuevas para la creación de un *frame* consecutivo a otro. Se seleccionan solo un porcentaje de muestras nuevas respecto al *frame size* establecido, la resta de muestras, las primeras para el nuevo *frame*, son las últimas del *frame* anterior. Con este truco, se evita la pérdida de causada por el *windowing*.

El porcentaje de nuevas muestras es el valor del parámetro llamado *overlapping*, o, en ocasiones, se indica directamente el número de nuevas muestras a seleccionar con el parámetro llamado *hop size*. Los valores habituales suelen ser del 50% o 75% aunque, como siempre, dependerá del problema y tipo de ventana que se haya seleccionado.

Funcionamiento de la transformada de Fourier

Para calcular cada componente o valor de cada frecuencia, la FT se sirve de la propiedad matemática, donde si dos señales se multiplican entre ellas y posteriormente se calcula su área resultante, se aplica la integral, esta será más grande cuando más similares sean las dos señales originales que se han usado para multiplicar. El área será positiva o negativa en función de la diferencia de fases que se tengan en ambas señales a comparar. La magnitud o amplitud del espectrograma para cada frecuencia en cada *bin* temporal será precisamente el valor del área en valor absoluto.

Para saber detalles más técnicos, mirar el anexo I, en él se explica de forma detallada el funcionamiento de la transformada de Fourier y algunas modificaciones que se le pueden hacer para obtener la información más compactada como la creación de los Mel-espectrogramas, también usados en este proyecto.

3.4.2. Características en el dominio temporal

Cuando se usa la forma del audio (*waveform*) para extraer características es cuando diremos que se encuentran en el dominio temporal. Son las que se utilizan normalmente en los algoritmos aprendizaje máquina (ML) más tradicionales como los SVM, *random forest*, *K-neighbours*, k-means, entre otros. Los algoritmos con redes neuronales, sobre todo los que tienen recursión, como las LTSM, GRU, etcétera, usan directamente la *waveform* como *input* de los modelos, entrada de las redes. No obstante, debido al tiempo de entrenamiento requerido para las NN en ser la *waveform* un dato mucho más pesado (contiene mucha más información) y complejo de extraer patrones temporales, las NN normalmente usarán los espectrogramas o sus variaciones (dominio frecuencial), ya que estos son menos pesados y permiten captar los patrones mucho más rápido, aunque con resultados ligeramente peores.

Algunas de estas características son los siguientes. Se explican brevemente solo por satisfacer el objetivo secundario de conocer las principales características y técnicas del audio, ya que, como se menciona en las mismas características, ninguna de ellas será válida para el problema actual de eliminación de ruido.

Amplitud envelope

Consiste en dividir la señal en varios segmentos (*frames*). De cada uno se extrae y guarda solo la muestra con el valor más grande, creando así un vector numérico de longitud igual al número de *frames*. Como se puede deducir, el vector de características finales será mucho más pequeño que el de la señal original.

Esta técnica es útil cuando tenemos problemas de *onset detection*, como, por ejemplo, saber cuándo empieza una nota musical o una palabra. Sin embargo, es sensible a los valores atípicos (*outliers*) y no será válido para el problema del presente trabajo a causa de la imposibilidad de recuperar la señal original.

Root mean square eneregy

Similar al método anterior, con la salvedad de usar la función RMS (media con valor absoluto de todos los valores) en lugar de aplicar el máximo, por lo tanto, es una técnica más robusta contra los *outliers*.

Es útil cuando se quiere eliminar o detectar silencios, o, incluso, para hacer una clasificación por géneros musicales, ya que, por ejemplo, hay algunos como el rock que darán valores mucho más elevados que otros géneros como la música clásica.

Al igual que la anterior característica, y por la misma razón, no es apta para el problema actual.

Zero crossing rate

Cuenta y devuelve el número de veces que una señal, o audio, pasa por amplitud cero.

Es útil para identificar sonidos de percusión versus de timbre o melódicos, identificar cuando el audio se trata de voz o no, ya que la voz suele tener un ratio bajo, etcétera.

Nuevamente, no es una técnica apta para nuestro problema con la misma justificación que las anteriores.

3.4.3. Características en el dominio frecuencial

Son las características que se extraen de la transformada de Fourier (FT) y sus derivadas. Usan las frecuencias para extraer las características o propiedades que más tarde se les pasarán a los algoritmos.

Teniendo en cuenta de que el ruido que nos podemos encontrar es normalmente ruido gaussiano, donde cabe esperar una distribución de energía equipotencialmente distribuida a lo largo de todas las frecuencias, y que, en la música (incluida la voz) normalmente la energía suele encontrarse centrada más bien en las frecuencias más bajas, entre los 20 Hz y los 10 KHz, donde, a más, cada instrumento diferente tiene unas determinadas frecuencias para cada nota musical (los llamados armónicos, una frecuencia fundamental o nota más algunas que son múltiples de esta), hace que el espectro frecuencial tenga una vital importancia en la detección de patrones en el audio, ideal para los algoritmos.

Mientras que para los algoritmos que usan redes neuronales se suele usar los espectrogramas y Mel-espectrogramas, otras características de este dominio son: *band energy ratio*, *spectral centroid*, *spectral flux*, *spectral roll-off*, etc. A continuación, se explican las más importantes, que, al igual que las del dominio temporal, se explicarán solo brevemente para tener un conocimiento básico, pero sin que se vayan a usar en el proyecto actual debido a su incompatibilidad.

Band energy ratio (BER)

Para cada *frame* se elige una frecuencia de corte, normalmente de 2 KHz, después, todas las energías de las frecuencias inferiores a esta, así como las superiores, por separado, se les aplica el cuadrado (para eliminar valores negativos) y se suman. De los dos valores obtenidos se dividen ambos creando una ratio con la información sobre la relación entre las bandas frecuenciales más pequeñas y las más altas.

Esta es una característica útil para, por ejemplo, clasificar por género musical, reconocimiento del habla (*speech recognition*), y otras. No obstante, al igual que pasaba con las características centradas únicamente en dominio temporal, en no poder recuperar la señal original desde el valor de la ratio, esta no puede ser usada para el problema actual.

Spectral centroid

Devuelve la frecuencia media ponderada (centro de masas) para cada *frame*. Para ello, se multiplica cada frecuencia por su proporción de energía respecto a todas las demás y, finalmente, se suman todas ellas.

Nuevamente, puede ser útil para la clasificación musical y auditiva, pero no para el problema al que nos atañe.

3.4.4. Características en ambos dominios

Son las características que se extraen del espectrograma (STFT) o modificaciones de este como los Mel-espectrogramas, siendo estos una característica en sí. Las características más usadas por los algoritmos tradicionales en este dominio son: las MFCC (coeficientes del Mel-espectrograma, extraídos de este mismo) o la *constant-Q transform*.

Dado que con las características de dominio frecuencial y temporal por separado no podemos tener la información sobre el tiempo y la frecuencia al mismo tiempo, para ello tenemos representaciones visuales que juntan ambas informaciones, permitiendo ver las variaciones frecuenciales a lo largo del tiempo.

La principal diferencia de los modelos DL (redes neuronales) versus a los tradicionales (SVM, *random forest*, etcétera) es que de todas las características no tenemos que seleccionar aquellas que creíamos conveniente y que podrían funcionar mejor para resolver el problema actual, sino que directamente le pasamos a las NN la información en crudo del espectrograma, Mel- espectrograma o la *waveform* para que sean estas quienes aprendan a extraer las características de forma automáticamente al mismo tiempo que minimizan su función de coste durante su fase de aprendizaje o entrenamiento.

A continuación, se explica por encima las características que se utilizarán en este proyecto, el espectrograma y el Mel-espectrograma. Se usarán estas y no otras ya que son idóneas debido a su información que nos aportan (dominio frecuencial y temporal).

Espectrograma

Consiste en una matriz donde las filas son todas las frecuencias (*bins* o *frames* frecuenciales) y las columnas el tiempo (*bins* o *frames* temporales). El número de estas está definido por el parámetro *frame size*. La STFT es simplemente la aplicación de forma sucesiva de una FT a un intervalo de tiempo concreto dividido en *n bins* temporales, donde en cada uno se le aplica una FT por cada frecuencia con tal de crear los *bins* frecuenciales.

Para obtener el espectrograma, simplemente hay que aplicar el valor absoluto a los valores obtenidos de aplicar la STFT a la señal o audio. La característica que nos devuelve el espectrograma es la cantidad de energía de cada frecuencia por cada unidad de tiempo, es decir, la variación de energía frecuencial temporal. Adicionalmente, debido a que los humanos percibimos la intensidad, o energía de las frecuencias, de forma logarítmica, después de aplicar el valor absoluto, se transforman los valores obtenidos a decibelios, para pasar así de una escala lineal a una logarítmica y más representativa.

Un ejemplo de este se muestra en la figura 28, encontrada en el anexo I) *Explicación detallada de la FT y la creación de espectrogramas*, donde recordamos que, en él, como indica su nombre, se explica de forma más detallada y técnica en que consiste un espectrograma.

Mel-espectrograma

Uno de los problemas del espectrograma es que no representa correctamente la percepción que tenemos los humanos sobre el sonido. Esto es debido a que la forma en que percibimos el *pitch* no es lineal sino logarítmico, al igual que pasa con la intensidad (uso de dB).

Las frecuencias bajas son en las que tenemos mejor resolución frecuencial, es decir, distinguimos mejor dos frecuencias con la misma distancia en los tonos más graves (frecuencias bajas) que en los más agudos (frecuencias altas). Percibimos que los tonos más agudos están más cerca entre ellos que los graves.

En definitiva, si bien el espectrograma nos suministra una buena información o características sobre el audio, estas aún se pueden mejorar más. Esta mejora, no solo

consiste en dar una información más cercana a la que percibimos los humanos, sino también en comprimirla. Con ella, teóricamente se facilita y acelera el aprendizaje de los modelos en ser datos más descriptivos y compactos, facilitando el aprendizaje de patrones. Esta mejora es la creación de un nuevo tipo de espectrograma llamado Mel-espectrogramas, el cual, para generarse parte del básico o *vanilla*, el creado por la STFT.

El Mel-espectrograma contiene menos datos que el espectrograma normal, ya que, en basarse en una escala logarítmica, hace que muchas de las frecuencias más agudas (percibidas casi todas igual), queden solapadas en una única frecuencia (elimina redundancia), o, mejor dicho, Mel-frecuencia, las “frecuencias” del Mel-espectrograma.

Los Mel-espectrogramas, a rasgos generales, serían los dB de la frecuencia. Consiste en transformar la escala de las frecuencias a otra escala, donde, al igual que pasa con los dB, también sea logarítmica. Una vez aplicada dicha transformación, la distancia entre dos Mel-frecuencias equidistantes, perceptualmente serán igual de diferentes, aportarán la misma información.

Con esta última mejora podemos extraer una buena característica del audio, la cual tiene, como rasgos, tener una representación:

- 1) En tiempo-frecuencia (espectrograma).
- 2) Perceptualmente relevante en la amplitud (dB).
- 3) Perceptualmente relevante en la frecuencia (Mel).

Todos los rasgos que debería de tener una buena característica de audio.

Para crear un Mel-espectrograma se deberán de seleccionar cuantas Mel-frecuencias totales se quieren obtener. Los valores más comunes suelen ser de 40, 60, 90 o 128 como máximo, valores cercanos a los que nos podemos encontrar en las notas totales que forman la música occidental (88) o las que se suelen usarse comúnmente para la creación de la música pop (44, las notas centrales).

Información más detalla, científica y técnica sobre los Mel-espectrogramas se puede encontrar en el anexo II) *Explicación detallada de los Mel-espectrogramas*, y en las

referencias principales usadas para la redacción de este y los anteriores apartados [38], [49].

3.5. Explicación de los modelos usados

Esta sección está dividida en cuatro apartados diferentes. En el primero se explica brevemente en qué consisten las diferentes capas usadas en los modelos del proyecto, también se ve su funcionalidad. En los otros tres apartados, en cada uno de ellos, se explica muy superficialmente en qué consisten los modelos empleados (AE, VAE y U-Net).

Como se dijo en el estado del arte y en el apartado de la justificación del tema, todos los modelos seleccionados pertenecen a la familia de arquitecturas de red neuronales (NN) del tipo *encoder-decoder*, más concretamente las que usan redes convolucionales durante todo el proceso, pasándose a llamarse *convolutional encoder-decoder* (CED).

Este tipo de NN normalmente consisten en una red neuronal profunda, redes que tienen varias capas ocultas, donde la primera parte es la llamada *encoder* y la segunda *decoder*. Ambas partes están conectadas por una capa densa intermedia, siendo esta la que menos neuronas tenga, la llamada *bottleneck* o *latten space*, erradicando en ella la principal funcionalidad de este tipo de modelos.

La funcionalidad de este tipo de arquitecturas consiste en reducir el tamaño inicial a una dimensionalidad más pequeña (trabajo del *encoder*), extrayendo así solo las características importantes (representadas en el *latten space*) y, a partir de estas, recuperar la dimensionalidad original para realizar las predicciones (trabajo del *decoder*).

A continuación, empieza el primer apartado de esta sección, el de la explicación de los diferentes tipos de capas que nos podemos encontrar en los modelos usados e implementados en este proyecto.

3.5.1. Tipo de capas de los modelos

Dicho apartado ha sido elaborado gracias a las explicaciones y definición de conceptos encontrados en los artículos [37]–[39].

Capa convolucional

Una capa convolucional consiste básicamente en aplicar un filtro definido mediante algunos hiperparámetros, los pesos a aprender de dicha capa son los valores del filtro. Este filtro se irá moviendo por todos los datos de entrada de dicha capa aplicando una operación matemática, la multiplicación elemento a elemento de los pesos del filtro por los datos de entrada correspondientes. Los hiperparámetros más importantes son: El *kernel*, el *stride* y *padding*, donde su funcionalidad es establecer el tamaño del filtro, cuantos datos se deben avanzar en cada iteración durante la convolución y si rellenar o no la matriz resultante de la operación con ceros para igualar, o no, el tamaño de los datos de entrada con los de salida. Una convolución siempre tendrá como datos de entrada una matriz de como mínimo dimensión 2, como, por ejemplo, una imagen.

Un parámetro adicional y quizás el más importante junto al *kernel*, es el del número de filtros que se quieran crear y aplicar en cada capa de convolución, cada filtro resultará en una dimensión extra a los datos de salida. Normalmente, a medida que la capa de convolución sea más profunda en la red, más filtros se van a querer tener. Las primeras capas de convolución suelen ser para extraer patrones simples y básicos, mientras que las últimas capas de convolución suelen extraer patrones más complejos y específicos de cada problema a tratar.

Gráficamente, se puede resumir con las imágenes de las figuras 4 y 5.

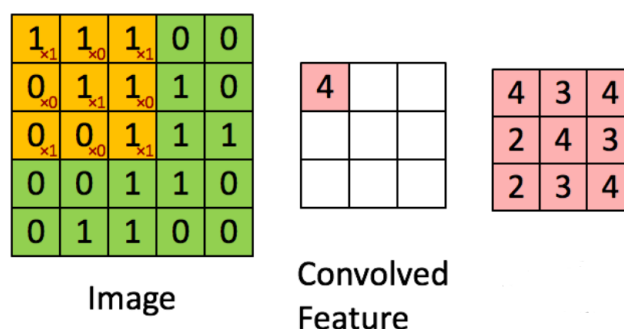


Figura 4: Operación convolucional. Imagen extraída de [37].

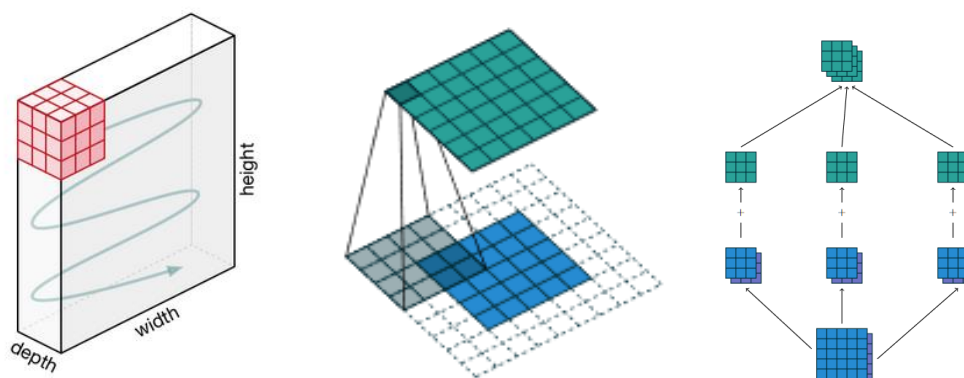


Figura 5: Movimiento del kernel durante una convolución. Imágenes extraídas de [37] y [39] sucesivamente.

Capa de pooling

La capa de *pooling* se usa normalmente cuando en la de convolución se aplica *padding*, y, por esto, no decrece el tamaño de los datos de entrada. Con *pooling*, similar a la capa de convolución, también se establece un tamaño de *kernel* para su filtro, así como los otros parámetros, a excepción del número de filtros que no existe, es un único filtro. No obstante, su principal diferencia es que esta vez no se aprenden pesos. La operación matemática que se hace es una operación simple, normalmente una media o suma de los valores que caen dentro del filtro en cada iteración (*stride*) del *pooling*.

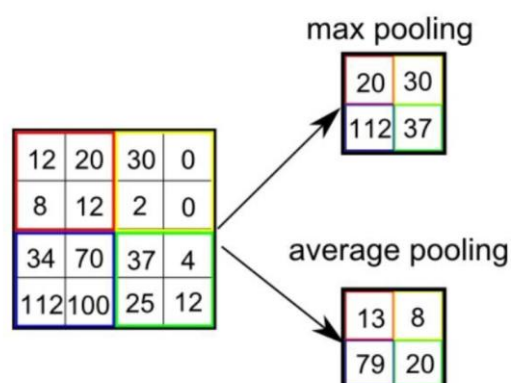


Figura 6: Tipos de pooling. Imagen extraída de [37].

Tanto la capa de *pooling* como la de convolución hacen que la red aprenda a identificar patrones de los datos que se le pasan. Con estos patrones, después, en la última capa convolucional de la red, el modelo deberá de usarlos y realizar sus predicciones con ellos. Por tanto, la funcionalidad de estos dos tipos de capa, es la de reducir la información o datos de entrada a un espacio cada vez más pequeño, pero con mayor riqueza en cuanto a su información útil.

Capa de convolución transpuesta o *upsampling*

Las capas de *upsampling* o convolución transpuesta hacen lo contrario a sus versiones estándar. Su finalidad es la de a partir de un espacio de datos más pequeño e ir aumentándolo hasta llegar al tamaño deseado para poder realizar las predicciones del modelo. Para conseguirlo hay varias técnicas, las más famosas son la interpolación, la copia de los datos cercanos o la utilización de *padding* para poder aplicar una convolución normal, pero permitiendo aumentar el tamaño de los datos de entrada sin *padding*.

Capa de activación

Las capas de activación no son más que simples funciones matemáticas que se aplican al valor obtenido por cada neurona o, en caso de una convolución, cada elemento de la matriz obtenida. Estas funciones permiten obtener valores de salida que se encuentren en un dominio concreto y específico definido por la función aplicada.

Las funciones de activación más comunes son: la *ReLU*, donde todos los valores negativos pasan a valer cero, la *LeakyReLU*, similar a la *ReLU* pero dejando pasar los valores negativos aunque muy atenuados, la sigmoide y la *tanh*, siendo estas dos últimas las que normalmente se usan en la última capa para dar las predicciones, ya que, ambas funciones, acotan los valores entre 0 y 1 o -1 y 1 sucesivamente, un rango acotado y de infinitos valores, ideal cuando se tienen datos normalizados o se quiere realizar una clasificación.

Capa de *batch normalization*

El *batch normalization* tiene como finalidad normalizar todos los datos de cada *batch*, es decir, calcular la media y la desviación estándar de estos para transformar los datos y escalándolos a una distribución normal de media cero y desviación estándar uno. Con esta transformación se aumenta la estabilidad del modelo durante la fase de entrenamiento.

La estabilidad es aumentada debido a que todos los valores quedan cercanos a cero, implicando que todos los datos de todas las características pasados a los modelos o, redes neuronales, tengan la misma importancia a priori debido a encontrarse en la misma escala. Es el modelo con el aprendizaje de los pesos quien decidirá qué datos son más relevantes y cuáles no.

Una vez mencionadas y explicadas todas las capas que nos podemos encontrar en los modelos seleccionados, finalmente pasamos a definirlos brevemente.

3.5.2. *Autoencoder y variational autoencoder*

Para la redacción de este apartado, diversos blogs, páginas web y artículos científicos [30], [33]–[36], [41]–[45] han servido como referencia.

La funcionalidad de los *autoencoders* (AE) y *variational autoencoders* (VAE), al igual que todas las otras arquitecturas de red basadas en la estructura *encoder-decoder*, consiste en reducir la dimensionalidad de las características o matriz inicial pasada como *input* a una de dimensionalidad menor (normalmente un vector, matriz de una dimensionalidad), de tal forma que esta misma, a partir del nuevo vector, sea capaz de reconstruir la matriz original (volver a su dimensionalidad inicial) perdiendo la menor cantidad de información posible.

Estas arquitecturas pueden tener como objetivo, utilizando la información extraída y guardada en el *latent space*, uno de las siguientes metas:

- 1) Intentar replicar la matriz original la más fidedigna posible a partir del vector reducido. Teniendo como objetivo aprender representaciones de menor dimensionalidad de una imagen o matriz para poder guardarla ocupando

menos espacio en memoria o usar dicha información en lugar de la imagen para pasársela a otros modelos como *input*, es decir, comprimir la información.

- 2) De una imagen o matriz que le falte un segmento pequeño o este corrupto, usando el *latten space* poder estimar la información que falta y, por ende, reconstruir la imagen al completo.
- 3) Detectar anomalías (*outliers*) en los datos de entrada. Para ello se usa una VAE o AE ya entrenada en el dominio correspondiente y, si con el *latten space* generado a partir del *input* no es capaz de reconstruirlo correctamente, se asumirá que este es un *outlier*.

Para este proyecto, la segunda meta ha sido su objetivo, tratando al ruido como una parte corrupta de la imagen a eliminar (predecirla sin ruido).

De las tres secciones que se componen los *autoencoders*, la primera, el *encoder*, será la que se encargue de ir reduciendo poco a poco la dimensionalidad original, donde en cada capa de profundidad irá siendo cada vez más pequeña. La segunda sección o conector entre una parte y otra del *autoencoder*, la *bottleneck*, representa el vector de dimensionalidad reducida, llamado *latten*. Por último, la tercera sección, el *decoder*, consiste en realizar el proceso inverso del *encoder*, es decir, en cada capa el número de neuronas irá aumentando, tanto de longitud como de dimensionalidad hasta alcanzar las dimensionalidades originales del *input* de la NN.

Normalmente, los *autoencoders* y sus variaciones, como las VAE, se suelen usarse con capas convolucionales (a veces llamados CAE y CVAE en lugar de AE y VAE), facilitando y aligerando los cálculos para la extracción de características de imágenes o matrices que se le pasen a la red como datos iniciales en su primera capa del *encoder*.

La principal diferencia entre una VAE y AE es en la construcción del *latten space* (L o z). Mientras que en las VAE está basado en el aprendizaje de una distribución de probabilidad, en las AE simplemente se aprende un vector numérico de características reducidas. La ventaja de aprender una distribución de probabilidad radica en la distribución (posición) de los puntos o valores en el nuevo espacio que se generan en

la *latten*, siendo esta la correspondiente a la distribución objetiva, normalmente una gaussiana normalizada.

Las distribuciones hacen que en el nuevo espacio se generen agrupaciones (puntos cercanos entre ellos) donde en cada una se generará, o reconstruirá, un tipo de imágenes (matrices) similares, es decir, cada agrupación representará una categoría. Del mismo modo, el espacio entre agrupaciones, lugar donde sus puntos no generan ninguna imagen realista (fiel a la de entrada), será prácticamente nulo.

Esta propiedad idónea para la generación de imágenes nuevas (no vistas en la fase de entrenamiento) no se da en las AE estándar, dificultando así en ellas la generalización del modelo y no haciéndolo tan apto como una VAE para la reconstrucción de imágenes nuevas.

Para generar la distribución de probabilidad en las VAE simplemente se añaden dos capas paralelas extra justo antes a la correspondiente al *latten* o *bottleneck*. Estas consistirán en la representación de las medias (μ_i) y varianzas (σ_i) de la distribución a generar sucesivamente. Su tamaño tendrá que ser el mismo que el del *latten* dado que cada neurona o posición de estas dos capas corresponde a la media y varianza que generarán una coordenada de la matriz (vector) *latten* mediante una selección aleatoria de la distribución formada por la μ_i y σ_i .

Dado que una distribución es probabilística (muestras aleatorias) y no determinista, esto impide que se pueda hacer *backpropagation* durante el entrenamiento del modelo, es por ello que, para poder entrenar una VAE se emplea un truco matemático llamado *parametrization trick*, el cual, tiene como finalidad separar la aleatoriedad de las capas μ y σ , habilitando en el proceso la opción de *backpropagation*.

El tipo de distribución de probabilidad deberá ser especificado y dependerá del problema y sus datos, sin embargo, dado que la mayoría de veces en la natura y datos se sigue una distribución normal, sobre todo cuando los datos que se tienen son muchos, normalmente se seleccionará que la distribución de probabilidad sea normal (Gaussiana), en caso de no obtener buenos resultados se podrían probar otro tipo de distribuciones como la esférica.

El impedimento del *backpropagation* sin el uso del *parametrization trick*, es debido a que, en seleccionar una muestra aleatoria, esta no va a corresponder al valor obtenido de los pesos de las neuronas de las capas μ y σ , en otras palabras, aun teniendo las mismas medias y varianzas, en cada iteración el resultado será diferente. Así, para separar la aleatoriedad, el truco de parametrización consiste en, a rasgos generales, generar una tercera capa adicional (épsilon, ϵ) previa a la *latten*.

Esta nueva capa tendrá la finalidad de generar los valores aleatorios desde una distribución normal (media cero y varianza uno). Es decir, la aleatoriedad ahora será un *input* interno de la red, pudiéndolo así separarlo de esta en el momento del *backpropagation* permitiendo realizarlo. Los valores del *latten* se calculan ahora con la fórmula $z_i = \mu_i + \sigma_i \cdot \epsilon_i$ en lugar de calcularse directamente, sin separar la aleatoriedad, o, lo que es lo mismo, sin usar la ϵ .

A continuación, en las figuras 7 y 8 se muestra gráficamente las diferencias entre una VAE con el *parametrization trick* y otra sin.

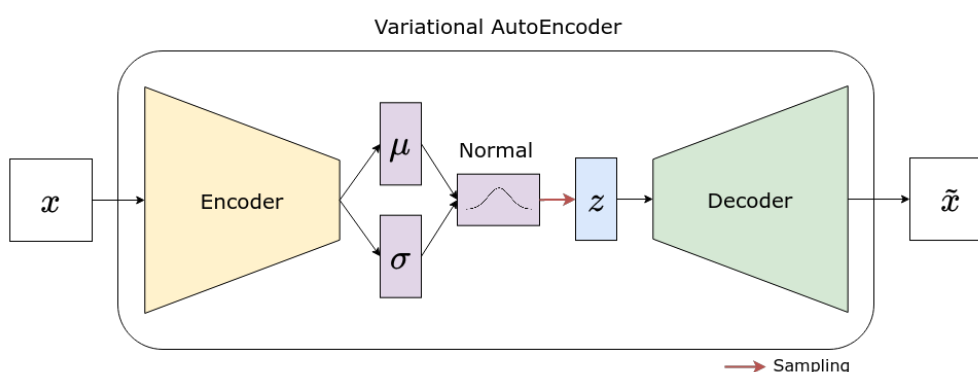


Figura 7: Arquitectura simplificada de una VAE sin *parametrization trick*. Imagen extraída de [33].

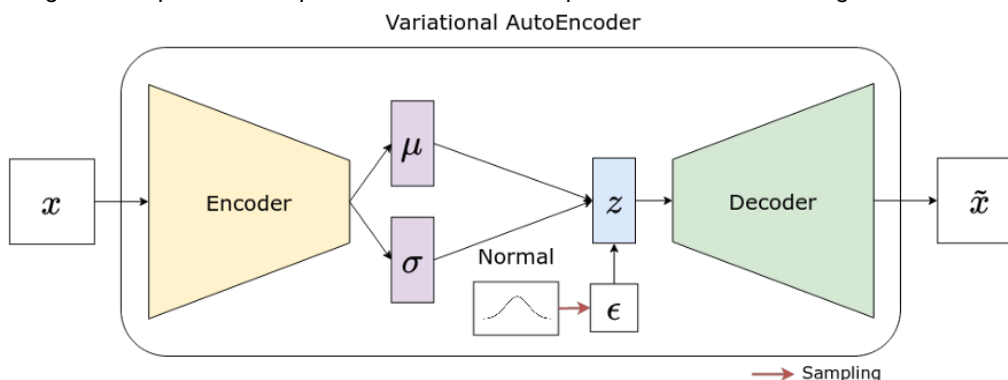


Figura 8: Arquitectura simplificada de una VAE con *parametrization trick*. Imagen extraída de [33].

Por otro lado, para controlar que la *latten* se genere en un espacio gaussiano, aparte de usar la función de pérdidas normal, la llamada función de pérdidas de reconstrucción (*reconstrucción loss*), encargada de verificar la similitud entre los datos iniciales y los predichos, se tendrá otra función llamada *Kullback–Leibler* (*KL loss*). Ambas funciones sumándose formarán la función de pérdidas usadas en la actualización de pesos (*backpropagation*) para una VAE.

La *KL loss* se basa en la divergencia de *Kullback–Leibler*, usada para comprobar que dos distribuciones sean iguales. Se mira la diferencia entre dos distribuciones diferentes a partir de los datos de entrada (*latten*), los pertenecientes a la distribución real, y los que se esperan tener por las mismas μ y σ dada la distribución objetiva, la comparada, asumiéndose una normal en la mayoría de casos. El objetivo de la pérdida KL es devolver un valor cercano a cero, ya que, en tal caso, querrá decir que las *latten* aprendidas por el modelo seguirán la distribución objetiva.

No obstante, el hecho de tener dos pérdidas, una que mira las distribuciones generadas y la otra como de buenas son las reconstrucciones, hace que se tenga el problema del balanceo entre pérdidas, es decir, tener que establecer a qué función de pérdida se le da más importancia mediante un factor multiplicativo en una de las dos pérdidas. Encontrar un balanceo óptimo muchas veces no es fácil y este puede variar dependiendo de los datos de entrada, problema, objetivo que tenga la VAE, distribución deseada, etcétera.

Una incorrecta selección de pesos en las dos funciones de pérdidas puede suponer un tener un modelo ineficaz. Por ejemplo, dándole todo el peso (importancia) únicamente a la pérdida de reconstrucción será como si tuviéramos un *autoencoder* (AE) estándar, es decir, las *latten* generadas no seguirán una distribución normal. Por otro lado, si le damos toda la importancia a la *KL loss*, los resultados obtenidos en la reconstrucción serán pésimos, dado que solo serán puntos (muestras aleatorias) de la distribución objetiva (normalmente normal), es decir, se generará únicamente ruido.

Así pues, debido al problema del balanceo, hace que uno de los hiperparámetros más importantes a gestionar en las VAE sea la distribución de pesos cada una de las funciones de pérdida.

Por otro lado, hay artículos como [42] que, si bien intentan automatizar este balanceo de forma automática y dinámica durante el entrenamiento, en el proyecto actual no se ha implementado ya que, uno, se quiso observar diferentes configuraciones de pesos para ver si se encontraba una buena configuración, dos, la técnica mencionada en el artículo no siempre da buenos resultados, dependerá como siempre, de la complejidad del problema y los datos que se tengan (su distribución puede cambiar).

3.5.3. U-Net

Todo el contenido de este apartado ha sido elaborado usando principalmente la información encontrada en [31], [38], [50]–[52].

Las U-Net son otros tipos de *encoder-decoder*, aunque esta vez, a diferencia de las VAE y AE, las capas que estén en la misma profundidad o altura en la parte del *encoder* y *decoder* deben tener exactamente las mismas dimensiones (número de neuronas), ya que estas estarán conectadas mediante concatenación (*skip-connections*).

La concatenación de las capas del *encoder* con las del *decoder* tienen la finalidad de recordar a la red (recuperar) la información original que podría haberse perdido durante el proceso de la creación de la *latten*. La concatenación podría ser vista como un filtro adicional de una capa convolucional, siendo este una copia idéntica de la matriz o capa concatenada. En la U-Net las capas concatenadas son las que corresponden al *encoder*, las que tienen guardada más información de los datos iniciales.

Adicionalmente, las U-Net también se diferencian de los AE estándar en la dimensionalidad del *bottleneck* o *latten*, siendo esta una capa convolucional más cuya dimensión de salida suele ser superior a 2, es decir, una matriz multidimensional y no un vector.

Asimismo, el objetivo de las U-Net no será el de aprender una representación de la información original a un espacio menor y después ser capaz de reconstruirlo, sino más bien transformar el *input* a otro tipo de *output* mediante el uso de la *latten*, como, por ejemplo, crear una máscara de segmentación. No tiene sentido usarla para

reconstruir una imagen a partir de un vector aleatorio (*latent*), ya que debido al uso de concatenaciones se necesita tener la imagen original.

Dado que el problema a tratar en este proyecto es muy similar a extraer una máscara, ya que se quiere eliminar parte de la información de los datos originales, la perteneciente al ruido, hace que esta técnica sea apropiada para aplicarla.

Gráficamente, la U-Net puede ser representada tal y como se muestra en la figura 9.

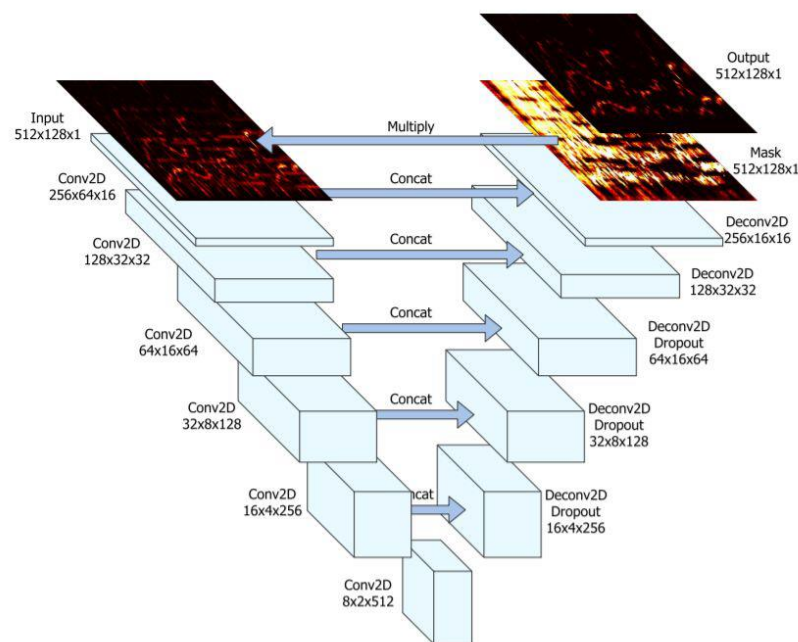


Figura 9: Arquitectura de una U-Net con masking. Imagen extraída de [16].

4. Desarrollo e implementación

Para la implementación del proyecto, se siguieron una serie de pasos generales.

- 1) Se seleccionaron los datos, audios, y posteriormente se construyeron los *datasets*, aplicando la STFT para construir los espectrogramas y Mel-espectrogramas de estos.
- 2) Se programaron los modelos VAE, AE y U-Net.
- 3) Se programó un bloque de código que permitiera recuperar a partir de los espectrogramas predichos el audio de estos. Para ello se utilizó la técnica de Griffin-Lim.
- 4) Se entrenaron y evaluaron los modelos mediante sus predicciones con el *dataset* de test, entrenamiento y validación.

La siguiente ilustración (figura 10), muestra todos los pasos resumidos de forma esquemática.

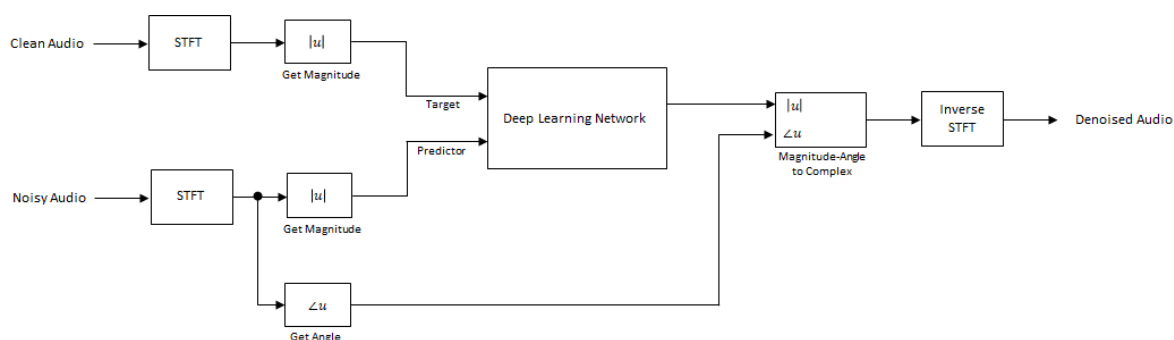


Figura 10: Esquema de la implementación. Imagen extraída de [14].

En las siguientes secciones se explica de manera más detallada como se realizaron cada uno de estos pasos.

4.1. Equipo, tecnologías y librerías usadas

El equipo utilizado para el entrenamiento de los modelos se compone de los siguientes componentes:

- CPU: i7-4790K
- GPU: GTX 970
- RAM: 16 GB

Las tecnologías y librerías fundamentales para el desarrollo del proyecto fueron:

- CUDA v10.1
- cuDNN v7.6.1
- Python 3.7.2
- tensorflow v2.9.1
- keras v2.9.0
- numpy v1.22.3
- librosa 0.9.1

Adicionalmente, otras librerías necesarias, pero menos importantes fueron: pandas, matplotlib, SoundFile, pickle, tqdm entre otras. Para saber el listado completo, y su versión correspondiente, mirar el archivo *requirements.txt* en el repositorio GitHub¹ del proyecto.

Como se puede observar de las tecnologías usadas, se usó CUDA para poder usar la GPU durante el entrenamiento de los modelos, y de esta forma, usando los diferentes núcleos de la GPU se pudo paralelizar algunos cálculos, permitiendo tener entrenamientos mucho más rápidos que si solo se hubiera empleado la CPU.

¹ <https://github.com/joan95vila/VIU-IA-TFM>

4.2. Análisis del dataset

Debido a que para este proyecto se necesitaba tener, por un lado, los audios con las canciones mezcladas con ruido, y por el otro, las canciones sin ruido, fue necesario tener dos *datasets*, uno para el ruido y el otro para las canciones. Finalmente, para generar el audio con ruido, simplemente se creó un tercer dataset combinando los dos primeros. Más adelante, en el siguiente punto, se explicará y justificará todo el proceso y pasos en que ambos *datasets* fueron fusionados.

El *dataset* de ruido está compuesto por dos videos. El primero de estos consiste en el ruido ambiente generado en una cafetería tradicional [53], principalmente se encuentra el murmullo de la gente hablando y muy ocasionalmente el ruido generado por la cubertería, tazas y platos que chocan entre sí o con el mobiliario. La fuente original de donde se extrajo este audio fue en YouTube, teniendo una duración total de 10 horas, sin embargo, debido al programa utilizado para descargar su audio en formato mp3, una aplicación web genérica en su versión gratuita, solo se pudo descargar las primeras 6 horas y 45 minutos.

El segundo video [54] también fue extraído de la misma red social anterior teniendo la misma duración original, aunque otra vez, debido a las restricciones impuestas por el programa de descarga, esta vez solo se pudo bajar 1 hora y 24 minutos. El audio del video consiste en el sonido ambiente que se puede extraer de una ciudad común, encontrándonos mayoritariamente el ruido generado por el tráfico, es decir, coches y otros vehículos circulando con mayor o menor velocidad, motos con su ruido particular de acelerones, cláxones y muy de vez en cuando alguna sirena de policía, ambulancia o bombero.

Las restricciones de duración en la extracción del audio de los dos videos no representan ningún problema debido a que, en ambos videos, el ruido y sus variaciones esporádicas como cláxones o choques entre platos se encuentran uniformemente distribuidos en toda la línea temporal, es por eso que no ha sido necesario usar otras herramientas o sonidos para tener los sonidos en su duración completa.

Por lo que respecta al *dataset* que contiene las canciones, el más importante de todos, ya que, aparte de combinarlos con los ruidos del *dataset* anterior, también será el objetivo de los modelos, su estimación, es el llamado *musdb18* [55], usado en muchos estudios del estado del arte debido a su estructura.

El *musdb18*, este compuesto por 150 canciones con una duración total aproximada de 10 horas de diferentes géneros. Cada canción está grabada en cuatro pistas separadas, correspondiendo cada una a: batería, bajo, voz y otros. Las canciones de este *dataset* vienen separadas en dos directorios diferentes, uno para *test* y el otro para el *train*, es decir, para el testeo y entrenamiento de los modelos. Sin embargo, para facilitar el proceso de transformación de los datos y creación del *dataset* final, esta separación inicial fue eliminada. Físicamente, los audios han sido grabados en estéreo y discretizados a 44.1 kHz con una profundidad de 16 *bits*.

Los géneros de las canciones que se encuentran dentro del *dataset*, así como su distribución son: pop/rock (72), rock (17), cantautor (14), Heavy Metal (12), pop (11), rap (8), electrónica (8), country (3), jazz (3) y reggae (2). Para más información de las canciones como su fuente original, licencia y nombre, se puede encontrar en [56].

Observamos que no hay una distribución uniforme, siendo el rock y el pop los géneros mayoritarios, representando un 70% del *dataset*. Así, podemos decir que los modelos han sido entrenados con canciones de pop y rock con algunas excepciones esporádicas.

Para una primera fase de desarrollo y evaluación de los modelos, ver cuál funciona mejor, no se ha creído conveniente aplicar ningún balanceo, como, por ejemplo, añadir más canciones de los géneros musicales menos representados en el *dataset* actual. Así mismo, dado que el rock y pop hoy en día son los géneros más escuchados, probablemente más del 70% de la música que se escucha, es también de proceder que los géneros a probar en los modelos sean de estos dos géneros musicales.

Los géneros minoritarios no se han eliminado, ya que, de esta forma, pueden servir como un ruido extra para hacer los modelos más robustos. Este ruido podría ser visto como canciones de pop o rock que, aun siendo catalogadas como tal, fueran bastante distintas del resto de las de su género. No obstante, más adelante, en posibles

ampliaciones y mejoras de modelos, para futuros proyectos, más géneros deberían ser agregados y probados.

4.3. Preparación de los datos para el dataset final

4.3.1. Transformación de los datos

Para la creación del *dataset* final, el que contiene las canciones mezcladas con el ruido ambiente, se combinaron los dos *datasets* del anterior apartado siguiendo los siguientes pasos:

- 1) De las canciones se eliminó la *intro* y la *outro*, de 20 segundos de duración cada parte. La razón de no tener en cuenta ni el inicio ni el final de las canciones es el hecho de que, en muchas canciones, en estas partes la voz no interviene y los instrumentos que lo hacen son pocos y con poca intensidad, siendo la mayor parte silencio.
- 2) Cada fragmento de audio, tanto el del ruido como el de las canciones, se fragmentó en segmentos de una duración de 0.74 segundos cada uno. La selección de dicha duración es justificada en el siguiente apartado 4.3.2) *Selección y justificación de los parámetros de la STFT*.
- 3) Se combinó los dos *datasets* (ruido y canciones). Para ello, de cada segmento de canción se crearon otros dos, uno combinado con el ruido de cafetería y el otro con el ruido de ciudad. Los segmentos de ambos ruidos fueron escogidos de forma aleatoria para cada canción.

Mientras se iban creando los nuevos segmentos, en un archivo csv se iba almacenando toda la información necesaria para poder saber cómo fue la combinación, así como tener identificadas las partes, nombres de las canciones y ruidos que fueron combinados en cada segmento.

Para la combinación de los audios, el volumen de las canciones se redujo a la mitad y el del ruido ambiente se mantuvo igual que el del audio original.

Después de varias pruebas se vio que esta era la mejor forma de mezclar ambos audios, de otra forma, o bien el ruido ganaba demasiada importancia, teniendo mucho más volumen que la música o viceversa.

- 4) Se aplicó la STFT para la extracción del espectrograma y Mel-espectrograma por cada segmento. Para ello, de los valores devueltos por la STFT se les aplicó el valor absoluto y se transformaron a la escala logarítmica (decibelios), es decir, solo se obtuvo la magnitud. No obstante, en otro directorio también fue guardada la fase por si más adelante se necesitaba recuperar la información referente a ellas, como, por ejemplo, poder recuperar el audio a partir de los espectrogramas.
- 5) Se normalizó los valores entre 0 y 1 para dar estabilidad y formato al modelo, facilitando así su aprendizaje. El rango de valores para la normalización fue este y no otro debido a la función de activación usada en las arquitecturas en su última capa, la sigmoide.
- 6) Una vez realizadas todas las transformaciones, los datos se separaron en tres partes, una para el entrenamiento y las otras para la validación y test. La selección de los datos (espectrogramas y Mel-espectrogramas de los segmentos de audio) se realizó de forma aleatoria para cada uno de los subconjuntos. La distribución que se asignó en cada grupo, así como su tamaño en memoria, es la mostrará en la tabla 1.

| Subconjunto | Segmentos de audios | GB (STFT) | GB (Mel-STFT) | GB (audios) | Duración (Horas) | % |
|-------------------|---------------------|-----------|---------------|-------------|------------------|------------|
| Train | 36428 | 7.08 | 0.83 | 4.63 | 7.82 | 67% |
| Validation | 9107 | 1.77 | 0.21 | 1.17 | 1.98 | 17% |
| Test | 8600 | 1.1 | 0.19 | 0.85 | 1.86 | 16% |

Tabla 1: Distribución y peso de los subconjuntos train, validation y test del dataset. Elaboración propia.

Debido a la gran cantidad de datos que se tiene, así como debido al uso de modelos con redes neuronales profundas, no se ha realizado ninguna validación cruzada (VC) durante el entrenamiento de estos. En caso de que se hubiera realizado, debido al tiempo de entrenamiento requerido para cada modelo, hubiera sido completamente

inviabile poder realizar este proyecto. No obstante, en tener una gran cantidad de datos en el *dataset* de validación y test, no haber aplicado la VC no es tan relevante como si este subconjunto hubiera sido de pocos datos, dado que, en tener una cantidad suficiente de datos, probabilísticamente podemos asegurar con bastante certeza de que estos contendrán una población representativa de la realidad, y, por ello, los resultados de las métricas obtenidas de este deberían de ser realistas y fiables.

4.3.2. Selección y justificación de los parámetros de la STFT

La duración de 0.74 segundos para la división, segmentación de los audios, es debido a dos motivos:

- a) Según la serie de videos educativos instructivos (minicurso) [29], de donde se escogió la arquitectura VAE y posteriormente se modificó para construir también los *autoencoders* VAE y AE del proyecto, de este se concluyó que la duración de 0.74 segundos para el espectrograma, y, por tanto, lo que debe durar cada segmento de canción era la duración que mejor resultado daba para el problema y datos que se trataba en dicho curso, la generación de audio (habla) artificial.
- b) Aprovechando el motivo de a) y, que todos los estudios vistos en la investigación de este proyecto (los nombrados en el estado del arte) tomaban segmentos, o espectrogramas, de duración entre 10 y 15 segundos, se quiso probar qué resultados se obtenían con espectrogramas más pequeños, teóricamente permitiendo a los modelos ser más precisos en captar mayor información temporal.

Por otro lado, recordando que la fórmula de Nyquist ($f_N = SR/2$) es la que nos muestra que la tasa de mínima muestreo a tener para llegar a reconstruir el audio hasta una frecuencia máxima, y, teniendo en cuenta de que los humanos tenemos un rango auditivo que normalmente abarca el espació frecuencial entre los 20 Hz y 20 KHz, pero que como norma todo el ruido ambiente, así como la voz humana e instrumental se suele encontrar entre el rango de 20 Hz y 10 KHz, finalmente, de los segmentos de

audio del *dataset* se volvieron a muestrear a 20 KHz (22050 Hz) para darnos una f_N de 10 KHz (11025 Hz), un muestreo más bajo que el de los audios originales pero suficiente para el problema a tratar.

Una tasa de muestreo más alto no debería de aportar información adicional, y, por ende, no debería de mejorar los modelos, sino más bien todo lo contrario. En aumentar el número de datos (muestras) con información vacía (sin información) también se hubiera incrementado el tiempo de entrenamiento y la dificultad de los modelos en la búsqueda de patrones. Al mismo tiempo, una SR más baja permitió tener un *dataset* relativamente pequeño en cuanto a su tamaño se refiere, siendo este del orden de unos 25 GB en total, tal y como se puede calcular de la tabla 1 mostrada en la sección anterior.

Por lo que respecta a los demás parámetros requeridos para la transformada de Fourier (STFT), el *frame size* elegido fue de 2048 con un *overlapping* del 50% utilizando la función Hann para el *windowing*. Todos estos parámetros son comunes cuando se trata con música, ya que otra vez, por Nyquist, en aplicar la STFT y tener un *frame size* de 2048, solo la mitad, 1024, serán las frecuencias discretizadas (*bins* frecuenciales o resolución frecuencial) que se tendrán en el espectrograma.

En tratar con voz, normalmente será suficiente usar la mitad o un cuarto de este tamaño debido al menor rango frecuencial que ocupa la voz respecto a los diferentes instrumentos que podemos encontrarnos en las canciones. Dicha información se puede verificar si se miran los valores para los mismos parámetros en la literatura investigada y nombrada durante el apartado del estado del arte, ya que, en casi todas ellas usan los mismos valores para los hiperparámetros, variando solo el valor del *overlapping* y *frame size* entre los diferentes estudios, utilizando *overlappings* del 50% o 75%, así como un *frame size* de 2048, 1024, 512 o 256, puesto que, como se explicó en el apartado teórico (concretamente en el anexo I), para usar la STFT (la versión rápida de la transformada de Fourier) dichos valores tienen que ser en potencia de dos.

Dado que el tamaño del *frame size* repercute de forma inversamente proporcional en la resolución temporal (explicado también en el anexo), establecer un buen valor es de vital importancia, ya que uno malo supondría la incapacidad de aprender por parte

de los modelos debida a la pobre información que se les suministraría. En definitiva, encontrar un balance entre la resolución temporal y frecuencial es crucial y muchas veces no trivial.

4.4. Hiperparámetros de los modelos

De las arquitecturas usadas, se han probado los siguientes valores por cada uno de los siguientes hiperparámetros:

| Arquitectura | AE |
|----------------------|-------------------------------------|
| <i>Batch size</i> | 32, 128 |
| <i>Latent space</i> | 16, 128, 532 |
| <i>Learning rate</i> | 1e-3, 1e-4 |
| <i>Input data</i> | Espectrogramas, Mel- espectrogramas |

Tabla 2: Hiperparámetros probados del modelo AE. Elaboración propia.

| Arquitectura | VAE |
|-----------------------|--|
| <i>KL loss weight</i> | 0 ² , 0.0001, 0.001, 0.005, 1 |
| <i>Batch size</i> | 32, 128 |
| <i>Latent space</i> | 16, 128, 532 |
| <i>Learning rate</i> | 1e-3, 1e-4, 2e-5 |
| <i>Input data</i> | Espectrogramas, Mel- espectrogramas |

Tabla 3: Hiperparámetros probados del modelo VAE. Elaboración propia.

² En establecer el valor del KL *loss* en 0, se simula una AE usando la arquitectura VAE, es decir, con las dos capas adicionales de μ y σ . Con ello se ha querido probar que efecto tienen estas capas en una AE, usadas como una forma de regularización adicional.

| Arquitectura | U-Net |
|----------------------|-------------------------------------|
| <i>Batch size</i> | 16 |
| <i>Dropout rate</i> | Ausente (0), 0.5, 0.3 |
| <i>Bottleneck</i> | 1024 |
| <i>Learning rate</i> | 1e-3 |
| <i>Input data</i> | Espectrogramas, Mel- espectrogramas |

Tabla 4: Hiperparámetros probados del modelo U-Net. Elaboración propia.

En todas las configuraciones se usó la técnica de “*reduce on plateau*” para reducir el *learning rate* en un factor de 5 cuando, durante 3 épocas seguidas, la función de pérdidas (coste) no fuera capaz de reducir su valor en más de un 0.0001. Con ella se pudo optimizar más algunos de los modelos, permitiéndoles hacer las últimas mejoras para acabar de obtener los mejores resultados posibles dentro del mínimo local.

El *input shape* de todos los modelos (datos de entrada), establecido por el espectrograma o Mel-espectrograma, fueron de 1024x16x1 y 128x16x1 respectivamente, determinados por la duración del espectrograma (0.74 segundos) y el tamaño del *frame size* junto a la SR establecida.

En lo que respecta a la U-Net, solo se pudo hacer uso de un *batch size* de tamaño 16. Este hecho fue gracias a la relación directa y proporcional que existe entre el tamaño de este y la cantidad de datos que se tienen que cargar en memoria, así como la arquitectura y complejidad de la arquitectura U-Net usada, con concatenaciones (copias de las capas del *encoder* al *decoder*). Dicha relación y arquitectura, junto al equipo informático disponible para entrenar, hizo que en establecer un valor del *batch size* superior a 16, el modelo terminara de forma abrupta su ejecución (impidiendo el entrenamiento) por falta de memoria RAM.

4.5. Arquitecturas de los modelos

VAE y AE

Las arquitecturas usadas para la VAE y AE se compone de varios bloques convolucionales o convolucionales transpuestos (dependiendo si es la parte del *decoder* o *encoder*), donde en cada uno se tiene un *kernel size* de 3 y *stride* de 2 para

reducir el tamaño de entrada a la mitad en la salida de cada bloque convolucional, o, en su defecto, duplicar el tamaño en los bloques convolucionales transpuestos. Una vez el *stride* y *kernel* (filtro convolucional) es aplicado, una capa adicional de *batch normalization* y otra de activación con la función *LeakyRelu* son también aplicadas.

El *batch normalization* permite estabilizar el modelo ayudándolo en la convergencia del descenso del gradiente, puesto que, *batch normalization*, normaliza los valores de salida de la capa convolucional para que estos no tiendan a crecer indefinidamente. Al mismo tiempo, esta técnica contribuye a la regularización de los datos y como consecuencia combatir el *overfitting* levemente.

El uso de *LeakyReLU* como función de activación es debido a ser la evolución de la *ReLU* estándar, donde, por sus propiedades matemáticas, atenúa los valores negativos, ayudando también en la convergencia del modelo, así como la liberación de carga computacional en algunos cálculos, y, por ende, acelerando el proceso de aprendizaje.

Esta liberación computacional es gracias a la transformación de los números negativos en valores casi nulos (cerca de cero). Es por eso y la observación experimental positiva (buenos resultados) de los últimos años en la literatura, que, la *ReLU* y *LeakyReLU* suelen ser las más empleadas. En la última capa del *decoder* se ha usado una sigmoide como función de activación. Esto es debido a que las predicciones a realizar (valores del espectrograma) tienen que estar comprimidos entre el rango de valores que va desde 0 hasta 1 gracias a la normalización que se le aplicó en la creación de los *datasets*.

En la parte del *encoder*, los filtros de cada capa, de menor a mayor profundidad son: 32, 64, 128, 256 y 512. Los filtros de las capas convolucionales transpuestos que se encuentran en la parte del *decoder* son del mismo tamaño. La capa intermedia de la VAE y AE, la llamada *bottleneck*, su vector *latent* tiene un tamaño variante en función del experimento o configuración a realizar. Estas se especifican en el apartado 5) *Resultados y análisis*, concretamente en las tablas que se muestran conteniendo las métricas obtenidas por cada modelo, así como los valores empleados de sus hiperparámetros.

Sin embargo, para adaptarla a los datos del proyecto actual, ha sido modificada en su tamaño de entrada (tamaño del *input*) y, por lo tanto, los tamaños de entrada de las capas intermedias también han sido modificados. Complementariamente, para alguno de los modelos también se han añadido capas de *droupout* antes de la concatenación y de la aplicación de *maxpooling*. Los valores de *droupout* para cada configuración (modelo), otra vez, se pueden encontrar en las tablas del apartado 5) *Resultados y análisis*. Dichas capas han sido implementadas para añadir un elemento más de regularización, evitando el *overfitting* que se generaba sin la aplicación de *droupout*, tal y como se indica y muestra en el apartado 5) *Resultados y análisis*.

Finalmente, tanto en la AE, VAE y U-Net, se ha usado la *seed* con el valor 123 con la intención de que todos los procesos que contengan alguna operación de aleatoriedad, como la selección de datos para los *batch* en los generadores, sean los mismos para todos los experimentos, permitiendo así compararlos justamente y volver tener la opción de reproducibilidad de los experimentos en cualquier momento en caso de que fuera necesario.

4.6. Funcionamiento del código y fases de implementación

Para obtener un mejor control sobre las librerías empleadas, así como sus versiones y dependencias, se ha trabajado sobre un entorno virtual de Python.

El código se ha estructurado modularmente para ser más comprensible, flexible y ampliable. Para ello se han creado varios directorios, uno para los *datasets*, otro para las funcionalidades o paquetes de programación a crear, y, finalmente, el de las ejecuciones, conteniendo el archivo principal del programa (*main*) con tal de poder ejecutarlo.

En caso de querer una explicación más detallada de cada parte, así como la relación entre los diferentes archivos creados y como ejecutarlo, mirar el anexo IV) *Requisitos y ejecución del programa*.

A continuación, para cada archivo *Python* que se ha creado, se usará para explicar las diferentes funcionalidades del programa (fases de implementación), así como fueron logradas.

- *datagen.py*: Se encarga de generar los generadores a partir del *dataset* para crear los conjuntos de test, entrenamiento y validación de los modelos. El hecho de usar generadores y no los datos de los *dataset* directamente es debido al tamaño de este, ya que, otra vez más, con el *hardware* actual disponible no es posible cargar todos los datos a memoria. Con los generadores, estos se irán cargando y descargando poco a poco para cada *batch*.
- *datasets* (directorio): cada subdirectorio corresponde a los datos que corresponde su nombre, por ejemplo, en *STFTs* se guardan los espectrogramas estándar, en *STFTs – complex* el resultado de aplicar STFT a los audios de las canciones, ruidos y su combinación, etcétera.
- *autoencoder.py* y *unet.py*: En cada uno se encuentran los modelos correspondientes con todos sus métodos, clases y atributos necesarios, para después, poder ser instanciados en otros archivos como el *main.py*. El archivo *autoencoder.py*, contiene los modelos VAE y AE.
- *model* (directorio): En el subdirectorio *tmp*, se almacenan los modelos que se estén entrenando, así como su histórico de métricas durante su entrenamiento para después poder evaluarlo. El subdirectorio *model_x* (no existe), es la representación de otros subdirectorios con el nombre del modelo y configuración correspondiente, es la copia del directorio *tmp* de modelos ya entrenados y evaluados.
- *main.py*: Es donde se instancia los modelos, se establecen sus configuraciones llamando al archivo *params.py* y *paths.py*.
- *main_generate_audio.py*: Se carga un modelo y audio con ruido para generar una predicción de audio limpia, sin ruido, solo música. Se aplica la IDFT con la técnica de *Griffin-Lim* de las predicciones y las fases de la señal/audio con ruido para reconstruir el espectrograma predicho a audio.

- *main_plot_samples.py*: Se carga un modelo y audio/s con ruido para generar una predicción/es de audio limpio. Una vez se tiene el espectrograma/s, se visualizan para observar el resultado gráficamente.
- *preprocess.py*: Se encarga de segmentar los audios de los dataset originales y el combinado (música mezclada con el ruido).
- *stft.py*: De los audios segmentados, se crean sus espectrogramas aplicando la STFT. El resultado es también normalizado y transformado a escala decibélica. Los Mel-espectrogramas también son generados en este mismo archivo.
- *Overlay.py*: Es donde se mezclan las canciones con el sonido ambiente para crear el *dataset* final o combinado.

Para la reconstrucción de la señal original, a partir del espectrograma predicho, existen varias técnicas para recuperar la fase perdida, la más común y la empleada para este proyecto es la llamada *Griffin-Lim*, no obstante, para ayudar a que los resultados sean mejores, en lugar de pasarle un vector de fases aleatorio, se le pasan las fases de la señal que se usó para predecir su espectrograma sin ruido, de esta forma, la técnica *Griffin-Lim* debería de dar aún mejores resultados.

Para la implementación de la VAE y AE, se ha tomado como punto de referencia, plantilla, el código del proyecto [29] debido a su claridad y modalidad. Mientras que para la U-Net se ha tomado como referencia el código del proyecto [31].

Para la segmentación del audio, aplicación de las STFT y la creación de los dos tipos de espectrogramas usados, se han combinado y usado varias funciones, métodos y clases provenientes de la librería de *Python* llamada *librosa*. Esta librería ya contiene implementada toda la programación necesaria con las fórmulas y métodos para la generación de audio y extracción de características del mismo.

5. Resultados y análisis

En este apartado se analizará y comparará el tiempo de entrenamiento que ha requerido cada uno de los diferentes modelos creados versus a las pérdidas obtenidas de los conjuntos de validación (*validation loss*), entrenamiento (*training loss*) y test (*test loss*). Este análisis permitirá ver qué configuración, o modelo, es el mejor. Un modelo que obtenga las mismas pérdidas de test con un menor tiempo de entrenamiento respecto a otro será mejor.

Para elegir el mejor modelo, primero solo se tendrá en cuenta el valor de sus pérdidas, así como el *overfitting* que se haya tenido, en caso de empate, es cuando el tiempo de entrenamiento intervendrá para decidir cuál de ellos es el mejor. Las pérdidas corresponden a la métrica MSE (error cuadrático medio) entre el espectrograma predicho y el espectrograma objetivo (canción sin ruido).

El *overfitting* será evaluado mediante la comparación entre las pérdidas de validación con las de entrenamiento. En las tablas de resultados (de la 5 a la 7) las pérdidas de validación y entrenamiento que se muestran hacen referencia a la mejor época obtenida durante el entrenamiento del modelo. Esta época es la correspondiente al número indicado entre paréntesis que hay en algunos de los modelos en la columna “*trained epochs*” de las tablas acabadas de mencionar. Dicha columna es la encargada de indicar el total de épocas entrenadas para cada modelo. En caso de no haber paréntesis, significa la última época fue la que mejor resultado dio, en otras palabras, que el modelo seguía aprendiendo, aunque ya de forma mucho más lenta, empezando así su fase de estancamiento.

Para ver las pérdidas de todas las épocas se pueden encontrar en GitHub³, concretamente en los archivos encontrados en el directorio llamado “*Evaluations*”. En esta memoria no se ilustrarán todas las pérdidas de cada época (curva de aprendizaje) por cada modelo, sino solo la de los mejores modelos de cada tipo de

³ <https://github.com/joan95vila/VIU-IA-TFM>

arquitectura. De otra forma, debido a la gran cantidad de información que se mostraría, mucha de ella siendo redundante, haría imposible su comprensión.

A continuación, procedemos a analizar las configuraciones o modelos que tienen como arquitectura un *autoencoder* (AE). Primero se mostrará la tabla donde hay representados los mejores resultados obtenidos por cada uno de estos modelos, así como sus configuraciones o valores de hiperparámetros usados. Después, a continuación de la tabla, se mostrará la curva de aprendizaje para el mejor modelo obtenido de los del tipo AE (AE-5). Finalmente, basándonos en la tabla y la gráfica de la curva de aprendizaje se analizará los resultados obtenidos y se extraerá sus conclusiones.

De las tablas de resultados (tablas 5-7), en verde estarán marcados los mejores modelos y en rojo los peores.

De la tabla de resultados, cada columna tiene la información:

- *ID exp*: Nombre que se le ha llamado al modelo para su identificación.
- *Transefer learning*: Si se ha usado o no dicha técnica para no empezar los entrenamientos desde cero, sino a partir de los valores o pesos de una red anterior donde su arquitectura fuera idéntica y con ello no perder tanto tiempo de entrenamiento. En caso de no estar vacía, significa que se usaron los pesos del modelo indicado para el *transfer learning*.
- *Trained epochs*: Total de épocas realizadas y la que mejor resultado ha dado. Su explicación detallada (la funcionalidad de los paréntesis) se encuentra al inicio de este apartado.
- *Training time*: Tiempo aproximado que dura una época en entrenar.
- *Learning rate, latten space y batch size*: Son los valores de tales hiperparámetros.
- *Train y validation loss*: Corresponden al valor de las pérdidas de entrenamiento y validación de la mejor época, la indicada en *trained epochs*.
- *Diif. Tain/val*: Es simplemente la resta entre los dos tipos de pérdidas. Sirve para ver su diferencia y así tener una idea de cuánto de *overfitting* hay sin tener que ver las curvas de aprendizaje, ya que en todas las épocas se ha observado que, en general, siguen la misma diferencia. Se puede corroborar

en el archivo de GitHub⁴ nombrado anteriormente donde en el hay la totalidad de las pérdidas de todas las épocas de cada modelo.

Modelos con la arquitectura AE

| ID exp. | Transfer learning | Trained epochs | Training time (aprox.) | learning rate | Latten space | Batch size | Train loss | Validation loss | Test loss | Diff. Train/val. |
|----------|-------------------|----------------|------------------------|---------------|--------------|------------|---------------|-----------------|---------------|------------------|
| AE-1 | - | 11 (8) | 25 min | 1.00E-04 | 16 | 32 | 0.013 | 0.018 | 0.018 | 0.005 |
| AE-2 | - | 7 (5) | 5 min | 1.00E-04 | 128 | 128 | 0.0075 | 0.0173 | 0.0165 | 0.0098 |
| AE-3 | - | 14 (7) | 15 min | 1.00E-04 | 532 | 128 | 0.0105 | 0.0168 | 0.0165 | 0.0063 |
| AE-4 | AE-1 | 13 (9) | 25 min | 1.00E-03 | 16 | 32 | 0.0132 | 0.0172 | 0.0171 | 0.004 |
| AE-5 | AE-2 | 17 (10) | 5 min | 1.00E-03 | 128 | 128 | 0.0079 | 0.0166 | 0.0158 | 0.0087 |
| AE-6 | AE-3 | 33 (12) | 15 min | 1.00E-03 | 532 | 128 | 0.0155 | 0.0179 | 0.018 | 0.0024 |
| AE-7 | AE-4 | 24 (10) | 5 min | 1.00E-03 | 16 | 128 | 0.0116 | 0.0175 | 0.0171 | 0.0059 |
| AE-MEL-1 | AE-3 | 42 (13) | 5 min | 1.00E-04 | 532 | 128 | 0.0095 | 0.016 | 0.018 | 0.0065 |

Tabla 5: Mejores resultados de los modelos con la arquitectura AE. Elaboración propia.

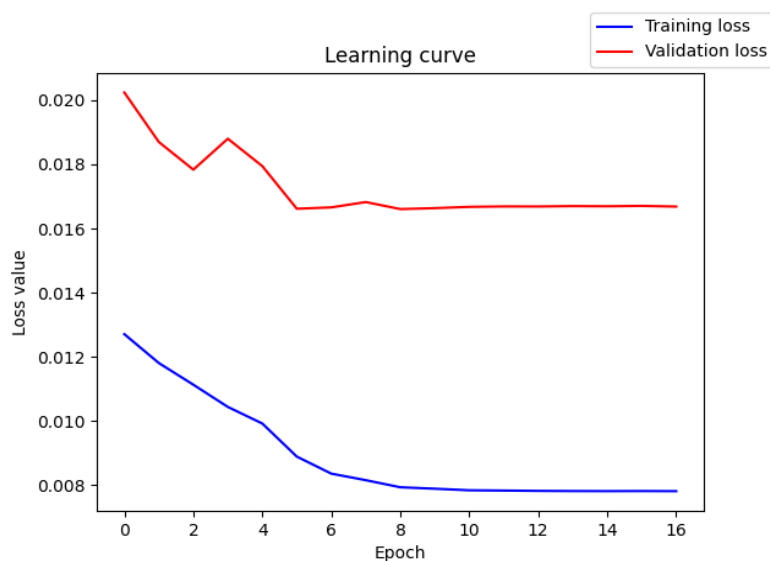


Figura 12: Curva de aprendizaje del modelo AE-5. Elaboración propia.

De los resultados mostrados en la tabla 5 se extrae la siguiente información:

- 1) El tamaño del *latten space* no parece afectar mucho al tiempo de entrenamiento total, a diferencia del *batch size* que lo incrementa

⁴ <https://github.com/joan95vila/VIU-IA-TFM>

significativamente, necesitando incluso las mismas épocas de entrenamiento que un tamaño de *batch* más elevado.

- 2) Para evitar un gran *overfitting*, las dimensiones del *latten space* y del *batch* parece ser que tienen que ser inversamente proporcionales, si uno es bajo el otro debe ser alto y al revés. Esta afirmación no puede ser tomada con total certeza debido a la falta de más experimentos que lo corroboren. Aun así, el *overfitting* está claramente presente en cada uno de los modelos, tal y como se puede apreciar en la figura 12 y la columna “*Diff. Train/Val.*” de la tabla 5 para los diferentes modelos.

Con la curva de aprendizaje del mejor modelo (figura 12), el AE-5, el que tiene menor pérdida de test, un tiempo de entrenado corto y un *overfitting* similar al de los demás modelos, aunque ligeramente mayor, confirmamos que, efectivamente, con este tipo de arquitecturas y la aplicación de la técnica *direct mapping* para generar las predicciones no se consigue evitar un *overfitting* exagerado.

Así mismo, los modelos no parecen ser capaces de aprender debido a que las pérdidas se mantienen constantes a lo largo del tiempo. Parecen haber llegado a un límite de aprendizaje o fase de estancamiento.

- 3) Para los *learning rate* probados no parece que estos tengan un efecto significativo en ninguna parte. Solo se observa una ligera mejora y empeoramiento en las pérdidas, dependiendo de las configuraciones de los otros hiperparámetros, sin embargo, sí parece mejorar el *overfitting* ligeramente, sobre todo en configuraciones donde los valores del hiperparámetro *latten space* sea alto.
- 4) De los experimentos AE-1 y AE-7, donde la única modificación es el tamaño del *batch*, podemos observar que aparentemente un mayor tamaño de este no solo favorece la rapidez de aprendizaje, sino también a la generación de mejores en los resultados aun siendo estas muy sutiles.

A continuación, se muestran algunas de las predicciones realizadas por el modelo AE-5 y su subconjunto de test. En cada muestra, se expone el espectrograma

perteneciente al audio combinado con el ruido y las canciones (el superior e *input* del modelo), el correspondiente al de la canción (el del medio y objetivo del modelo) así como el espectrograma predicho (el inferior).

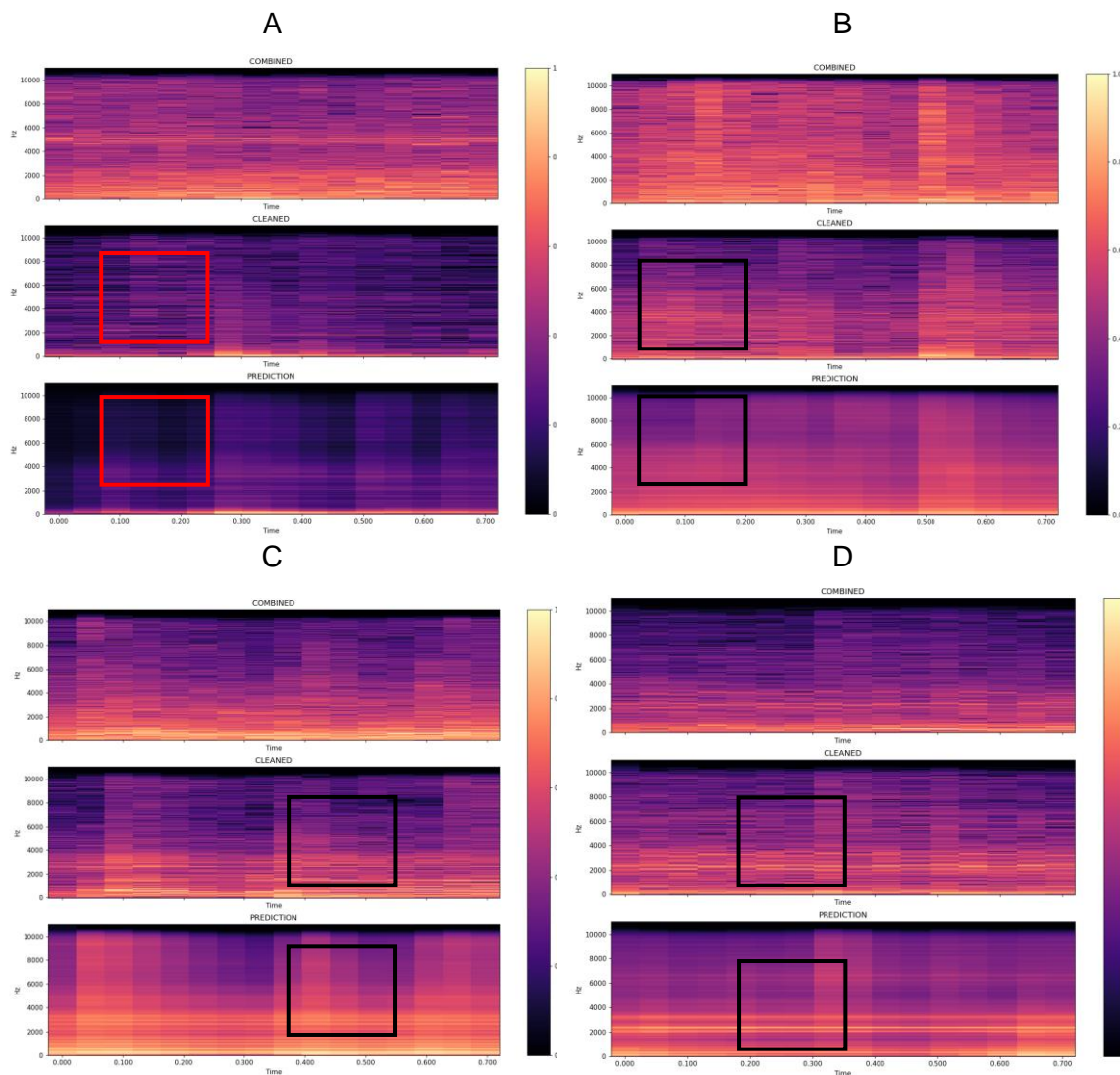


Figura 13: Muestras de espectrogramas predichos del modelo AE-5. Elaboración propia.

Con los rectángulos negros y rojos se muestran algunas de las diferencias claras entre el espectrograma objetivo y las predicciones obtenidas.

Analizando las muestras vemos que los espectrogramas generados parecen haber aprendido donde se encuentran las frecuencias que pertenecen al ruido versus a las que pertenecen a las canciones aunque sin llegar a ser capaces de reconstruir de forma precisa los espectrogramas, quedando muy difuminados (borrosos) alrededor de cada frecuencia, es decir, creando una especie de máscara (líneas) con mayor o

menor intensidad (energía) en función si detecta que son frecuencias que corresponden más al ruido o a las canciones.

Con estos resultados se concluye que hay un estancamiento global de aprendizaje en todos los modelos. Probablemente y compartiendo la opinión de otros autores como [6], debido al uso de la MSE como función de pérdidas, forzando seguramente a los modelos a aprender y generar predicciones que se parezcan más a una máscara que a un espectrograma en sí, impidiendo aprender el objetivo de eliminar el ruido de las canciones.

Es por ello que, tal y como se indica en el estudio nombrado, sería conveniente intentar poner otras funciones de pérdida o directamente dejar al modelo que la aprenda, como, por ejemplo, usando otro tipo de arquitecturas como las del tipo GAN, donde, precisamente, son ellas mismas quienes van aprendiendo la función de pérdidas al mismo tiempo que el objetivo. Visto los resultados, también se puede probar de aplicar la técnica *masking* en lugar de *direct mapping*, tal y como ya hacen la mayoría de estudios del estado del arte.

Por otro lado, la ausencia de buenos resultados y el gran *overfitting* que se obtiene, se sospecha que muy probablemente sea gracias al uso de la técnica de *direct mapping* en el tipo de arquitectura de *autoencoder* estándar, donde esta, debido a su naturaleza de su espacio de la *latten* (explicada en el punto 3.5.2) *Autoencoder y variational autoencoder*), tienden a crear un gran *overfitting* y por ende, ser una mala arquitectura en generar datos no vistos durante su entrenamiento, sobre todo si estos son muy distintos.

Con las predicciones actuales es imposible reconstruir un audio reconocible de dichos espectrogramas, dado que, si bien un error pequeño de varios píxeles para la reconstrucción de imágenes de carácter general como paisajes es aceptable y prácticamente imperceptible, para la reconstrucción de una señal a partir de un espectrograma un error pequeño puede suponer fatal para su reconstrucción a audio, haciéndolo completamente irreconocible.

En lo que respecta a los modelos basados en los Mel-espectrogramas, no se ha observado ninguna mejora ni empeoramiento global respecto a los modelos que han usado el espectrograma estándar, solo se ha observado una pequeña mejora en el

tiempo de aprendizaje, no obstante, debido a las transformaciones extra que se requieren para generar los Mel-espectrogramas, se ha llegado a la conclusión de que, para el caso actual, no vale la pena usarlos como datos de entrada. Como veremos a continuación, lo mismo pasa con los modelos que usan la arquitectura VAE.

Modelos con la arquitectura VAE

De los modelos con la arquitectura VAE, se han obtenido los siguientes resultados.

| ID exp. | Transfer learning | Trained epochs | Training time | learning rate | Latten space | Batch size | Weigth KL | Train | | Validation | | Test | | Diff. Train/val. Reconstruction |
|-----------|-------------------|----------------|---------------|---------------|--------------|------------|-----------|----------------|------|----------------|------|----------------|------|---------------------------------|
| | | | | | | | | Reconstruction | KL | Reconstruction | KL | Reconstruction | KL | |
| VAE-1 | - | 71 (12) | 5 min | 1.00E-03 | 16 | 128 | 0.0001 | 0.0143 | 5.25 | 0.0191 | 5.19 | 0.0193 | 5.31 | 0.0048 |
| VAE-2 | - | 14 | 10 min | 1.00E-03 | 128 | 128 | 0.0001 | 0.0163 | 5.13 | 0.0189 | 5.13 | 0.019 | 5.19 | 0.0026 |
| VAE-3 | - | 34 (19) | 10 min | 1.00E-03 | 128 | 128 | 0.001 | 0.013 | 1000 | 0.0176 | 1000 | 0.0176 | 1000 | 0.0046 |
| VAE-4 | | 8 | 10 min | 1.00E-03 | 128 | 128 | 1 | 0.018 | 1000 | 0.019 | 1000 | 0.019 | 1000 | 0.001 |
| VAE-5 | | 13 | 30 min | 1.00E-03 | 532 | 128 | 1 | 0.016 | 1000 | 0.0186 | 1000 | 0.0187 | 1000 | 0.0026 |
| VAE-6 | - | 8 | 50 min | 1.00E-04 | 532 | 32 | 1 | 0.0231 | 0.08 | 0.0233 | 0.14 | 0.0236 | 0.14 | 0.0002 |
| VAE-7 | VAE-3 | 9 (2) | 10 min | 2.00E-05 | 128 | 128 | 0.001 | 0.015 | 1.53 | 0.02 | 1.49 | 0.0202 | 1.5 | 0.005 |
| VAE-8 | VAE-3 | 12 (3) | 10 min | 2.00E-05 | 128 | 128 | 0.005 | 0.0166 | 0.7 | 0.066 | 0.21 | 0.0213 | 0.67 | 0.0494 |
| VAE-MEL-1 | | 22 (10) | 5 min | 1.00E-03 | 128 | 128 | 0.0001 | 0.0155 | 5.05 | 0.017 | 5.01 | 0.0193 | 5.09 | 0.0015 |
| VAE-MEL-2 | VAE-MEL-1 | 19 (9) | 5 min | 1.00E-03 | 128 | 128 | 1 | 0.0215 | 0 | 0.0216 | 0 | 0.0251 | 0 | 0.0001 |

Tabla 6: Mejores resultados de los modelos con la arquitectura VAE. Elaboración propia.

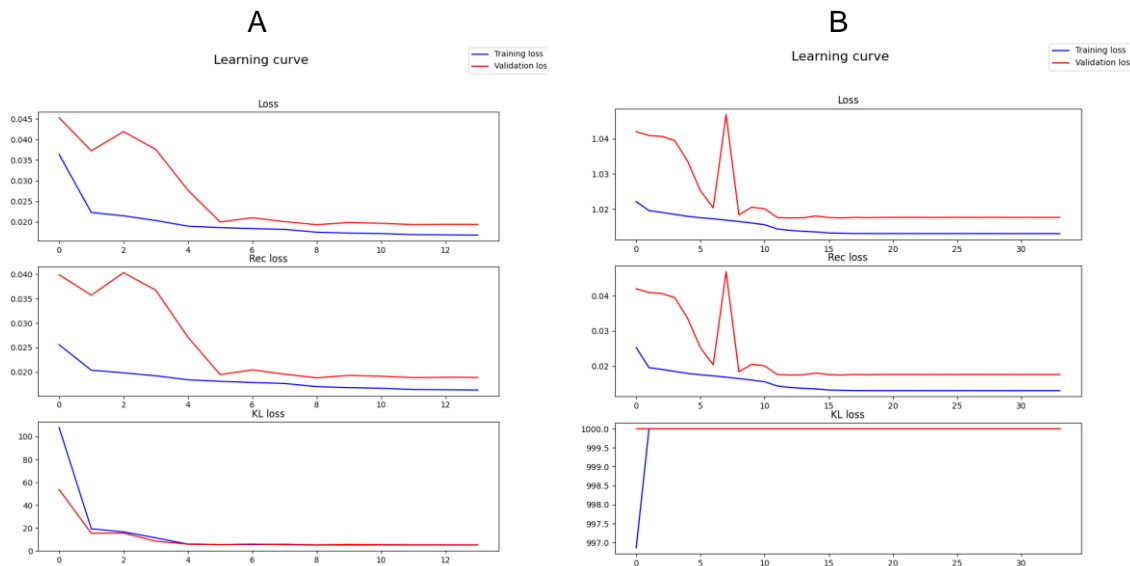


Figura 14: Curvas de aprendizaje de los modelos VAE-3 y VAE-4. Elaboración propia.

La primera observación destacable es la clara reducción del *overfitting* en la mayoría de los modelos, tal y como se puede ver en las curvas de aprendizaje de la figura 14. Sin embargo, como se explicará a lo largo de este punto, los resultados obtenidos respecto a los modelos con arquitectura AE son más pobres.

A diferencia de las AE, para analizar los resultados obtenidos de las VAE se tendrá en cuenta las dos funciones de pérdida que se tienen para este tipo de arquitecturas, la de KL y la de reconstrucción, siendo la primera la que nos indique que tanto se parecen los vectores de dimensionalidad reducida (*latent space*) a una distribución normal, y, por consiguiente, cuanto de bien debería ir la red en predecir espectrogramas de audios nuevos (nunca vistos durante el entrenamiento), y la segunda la que nos indique cuanta similitud tienen los espectrogramas generados respecto a los deseados.

En la tabla 6 hay mostradas las distintas configuraciones o modelos distintos probados con una arquitectura VAE. En lo que respecta a la selección de la mejor VAE (VAE-3), la que mejor resultado haya dado, el valor de la pérdida de reconstrucción del subconjunto test será la que más peso se le dará, seguido de la pérdida KL. Esto es así, ya que, como se ha dicho, la pérdida por reconstrucción es realmente la que indica que tan buenos son los resultados obtenidos respecto a los deseados. De nada serviría tener vectores *latent* en un espacio de distribución normal si estos no generarán espectrogramas medianamente reconocibles.

De la misma tabla, y como es lógico, debido a lo acabado de mencionar, se observa que el hiperparámetro que más efecto parece tener en los resultados es el peso de las pérdidas de KL (*weight KL*), el encargado de establecer la importancia de la pérdida KL, junto con la combinación del hiperparámetro *learning rate*.

Cuando los valores del hiperparámetros *weight KL* son elevados (0.001 o más) y el *learning rate* es alto (0.001 o más), estos tienen un impacto negativo en los resultados finales de las pérdidas KL, pero positivo en las de reconstrucción. Por otra parte, cuando los valores del peso KL son pequeños (menos de 0.001, obviando casi por completo dicha pérdida) y el *learning rate* sigue siendo alto, los resultados se invierten, siendo ahora las pérdidas de reconstrucción elevadas y las de KL bajas.

Analizando el hecho anterior, aunque aparentemente pueda parecer contraintuitivo, debido a que, en dar más importancia a las pérdidas de KL, estas deberían de ser menores y no mayores, no hay indicios de error. Se sospecha que este hecho puede ser debido a la normalización que se aplica a los datos a la salida de cada capa de las redes neuronales encontradas en cada uno de los modelos (salida normalizada entre

0 y 1), es decir, valores cercanos a los que daría una distribución normal, valores entre -1 y 1 en la mayoría de casos, donde, en ser multiplicarlos por un factor mínimo (cercano a cero), correspondiendo al valor del peso de las pérdidas de KL, hace que el valor de la pérdida de KL sea relativamente pequeña, haciendo pensar al modelo que el espacio creado en las *latten* ya este optimizado cuando en realidad no lo está.

Sin embargo, cuando los valores del peso de KL son grandes, haciendo que las pérdidas de KL cobren mucha más importancia, las pérdidas KL pequeñas pero no ínfimas, multiplicadas por el peso de KL, hacen que la pérdida de KL sea mucho mayor, y, por lo tanto, el modelo empieza a intentar optimizar la distribución del espacio creado de las *latten* al mismo tiempo que minimice el error o perdidas de reconstrucción, suceso que por los datos que se tienen no parece ser viable, incrementando así el valor de las pérdidas KL llegando incluso al límite superior del *clipping* aplicado a los modelos para mantener su estabilidad y convergencia.

Al mismo tiempo, se observa que, si el *learning rate* es bajo, independientemente del valor asignado en el peso del KL (alto o bajo), las pérdidas de reconstrucción se mantienen elevadas y las de KL siguen siendo bajas. De ello se deduce que, si bien un *learning rate* reducido facilita el aprendizaje de un espacio gaussiano normalizado en los vectores *latten* generados, en este espacio los modelos no son tan hábiles en generar sus predicciones. Mientras que para una *learning rate* elevada, la convergencia a un espacio gaussiano se hace más complicada.

En resumen, un valor elevado del *learning rate*, dificulta la convergencia de la *latten* a un espacio gaussiano y cuando se tiene un peso de KL elevado, el modelo no es capaz de generar buenos espectrogramas, mientras que, para un valor de peso pequeño, su generación es ligeramente mejor. Finalmente, ni las pérdidas de KL ni las de reconstrucción parecen ser alteradas por el valor que tomen los hiperparámetros de *batch size* y el tamaño del *latten space*.

Por otra parte, el hecho de que los mejores modelos hayan sido cuando la pérdida de KL se haya disparado, llegando a activar el *clipping* hace remarcar que los datos a predecir muy probablemente no puedan ser generados a partir de una distribución gaussiana, y, en consecuencia, la función de perdida KL se debería de asumir otro tipo de distribución, cambiar la comparación de una distribución gaussiana por otra.

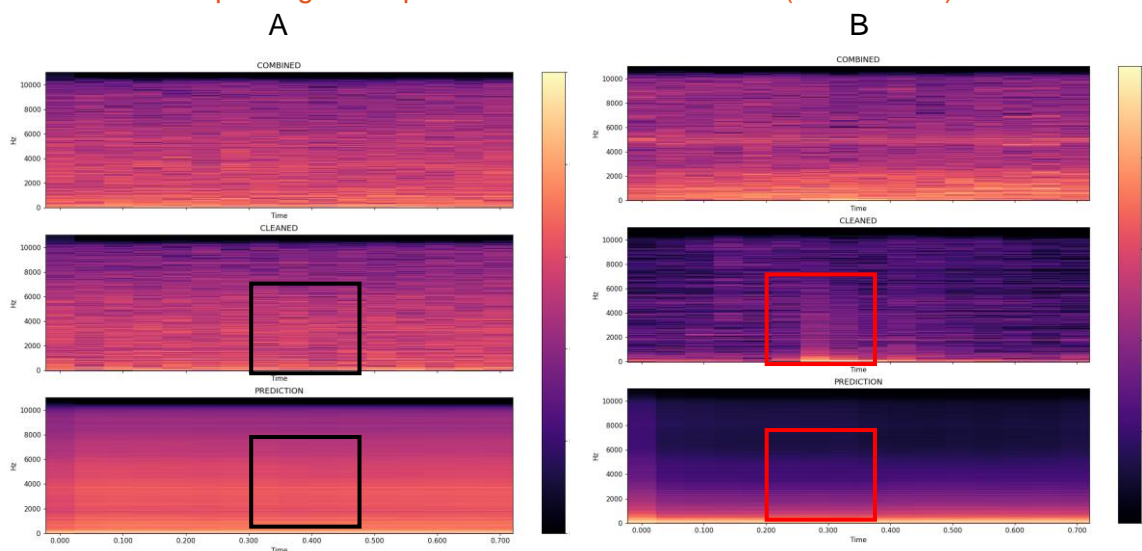
Otra causa de los valores obtenidos para las pérdidas, sobre todo las KL, hace pensar que sea fruto de la aleatoriedad, donde dependiendo de los pesos asignados inicialmente a las redes, estos pueden hacer que estas converger hacia un lado u otro, hacia unas mejores pérdidas de KL o a unas mejores pérdidas de reconstrucción.

Por lo que respecta al tiempo de entrenamiento, este está marcado casi en su totalidad por el tamaño del *batch*, teniendo una influencia mínima la longitud del espacio latente (*latten space*). Por último, a diferencia de lo que inicialmente se puede pensar, el *learning rate* en este caso no parece influir en el número de épocas necesarias para llegar al límite de aprendizaje para los modelos creados.

La dimensionalidad del espacio latente no parece tener un gran impacto en los resultados finales. Por ese motivo, se recomienda en caso de usar una VAE, explorar configuraciones de otros hiperparámetros y establecer un *latten space* de dimensionalidad pequeña.

Algunas de las predicciones realizadas por los modelos VAE-3 y VAE-6, con una KL baja y alta respectivamente, se muestran a continuación.

Espectrogramas predichos del modelo VAE-3 (KL *loss* alto)



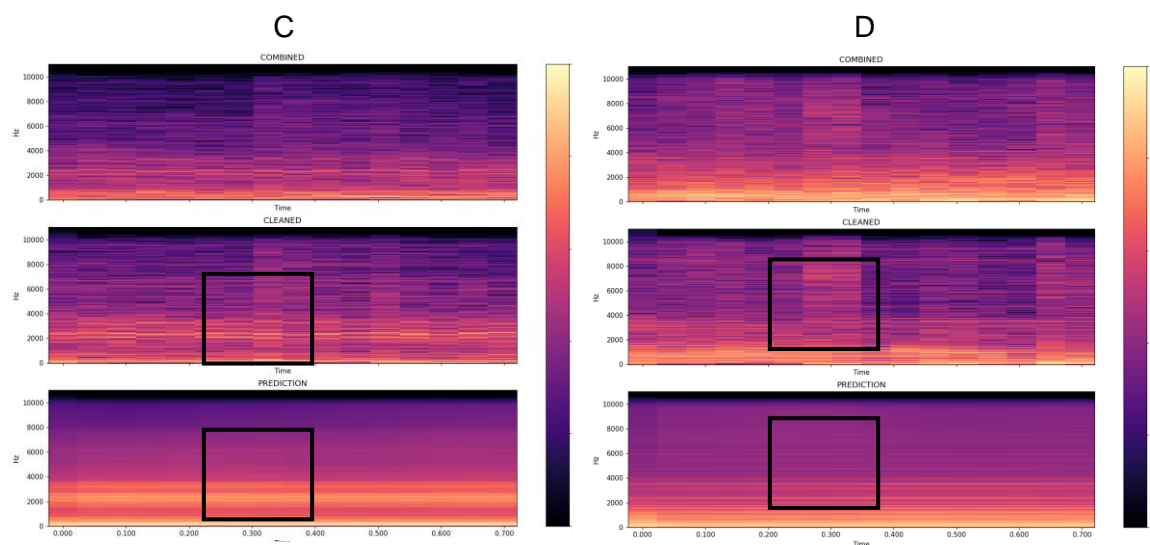


Figura 15: Muestras de espectrogramas predichos del modelo VAE-3 (KL loss alta). Elaboración propia.

Espectrogramas predichos del modelo VAE-6 (KL loss bajo)

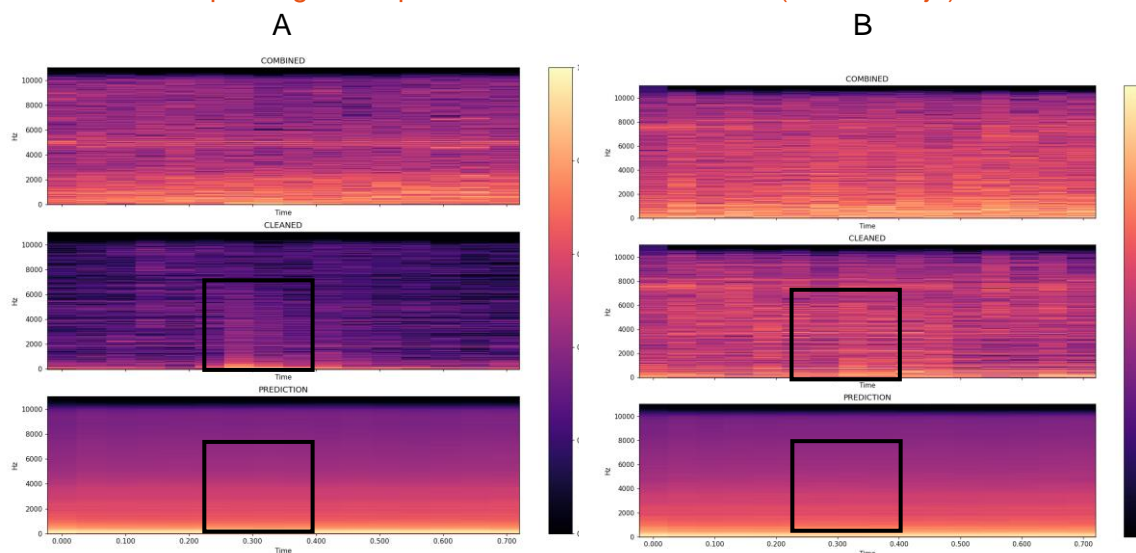


Figura 16: Muestras de espectrogramas predichos del modelo VAE-6 (KL loss bajo). Elaboración propia.

De los espectrogramas generados, se observa claramente que son mucho peores que los de los modelos AE. Esta vez, las VAE a diferencia de las predicciones de las AE, han generado definitivamente un tipo de máscara extremadamente sencilla, separando prácticamente solo las bajas frecuencias de las altas, siendo aún mucho más exagerado con los modelos donde el peso de la KL se le ha dado una mayor importancia, pero su pérdida ha sido baja (ej. VAE 6). Los modelos con el peso de la KL alta parecen todos generar exactamente la misma predicción. Una vez más, se reafirma que los datos no pueden ser generados a partir de una distribución gaussiana en el caso de usar los datos actuales y usar *direct mapping*.

Por ello, se concluye que, tal y como se hace en la mayoría de los estudios de la literatura, en utilizar este tipo de arquitecturas, debido a los resultados obtenidos, se puede asegurar casi con total certeza que los mejores resultados se deberían de obtener en tener como objetivo predictor el estimar una máscara y no directamente el espectrograma. Una vez obtenida la máscara sencillamente sería multiplicada por la matriz/espectrograma original y obtener así solo la canción sin ruido.

Para tener una referencia inicial, una red o modelo sin entrenar, una que genera simplemente ruido gaussiano en sus predicciones, durante su entrenamiento inicial se observó que en sus primeras iteraciones daba unas pérdidas de reconstrucción entre 0.08 y 0.05 junto a unas pérdidas de KL algo más variadas, aunque rondando los 50. Por ende, comparando con los resultados obtenidos vemos que los modelos antes de llegar a su fase de estancamiento acaban por aprender algunos patrones básicos, aunque insuficientes para la generación de buenos resultados, simbolizando que ninguno de ellos ha sido capaz de aprender a extraer el ruido de las canciones correctamente.

Es por ello que, tanto el uso de la VAE como AE para la eliminación de ruido de audio mediante espectrogramas y usando *direct mapping* en lugar de *masking* ha sido un error.

Modelos con la arquitectura U-Net

Finalmente, se muestran los resultados de los modelos con arquitectura U-Net. Siendo estos mucho mejores que la de las arquitecturas anteriores debido seguramente al uso de *skip-connections* en ayudar a la red a recordar la información original de los *inputs* en el proceso de reconstrucción, la parte del *decoder*.

| ID exp. | Input type | Transfer learning | Trained epochs | Training time (aprox.) | Dropout rate | learning rate | Bottleneck | Batch size | Train loss | Validation loss | Test loss | Diff. Train/val. |
|-------------|-----------------|-------------------|----------------|------------------------|--------------|---------------|------------|------------|------------|-----------------|-----------|------------------|
| U-Net-1 | Spectrogram | - | 4 | 3:30 horas | - | 1.00E-03 | 1024 | 16 | 0.0154 | 0.025 | 0.0193 | 0.0096 |
| U-Net-2 | Spectrogram | U-Net-1 | 12 | 3:30 horas | 0.3 | 1.00E-03 | 1024 | 16 | 0.0123 | 0.016 | 0.0157 | 0.0037 |
| U-Net-3 | Spectrogram | U-Net-2 | 4 | 3:30 horas | 0.5 | 1.00E-03 | 1024 | 16 | 0.016 | 0.02 | 0.0197 | 0.004 |
| U-Net-MEL-1 | Mel-spectrogram | - | 14 (13) | 3 horas | 0.3 | 1.00E-03 | 1024 | 16 | 0.0111 | 0.016 | 0.0241 | 0.0049 |

Tabla 7: Mejores resultados de los modelos con la arquitectura U-Net. Elaboración propia.

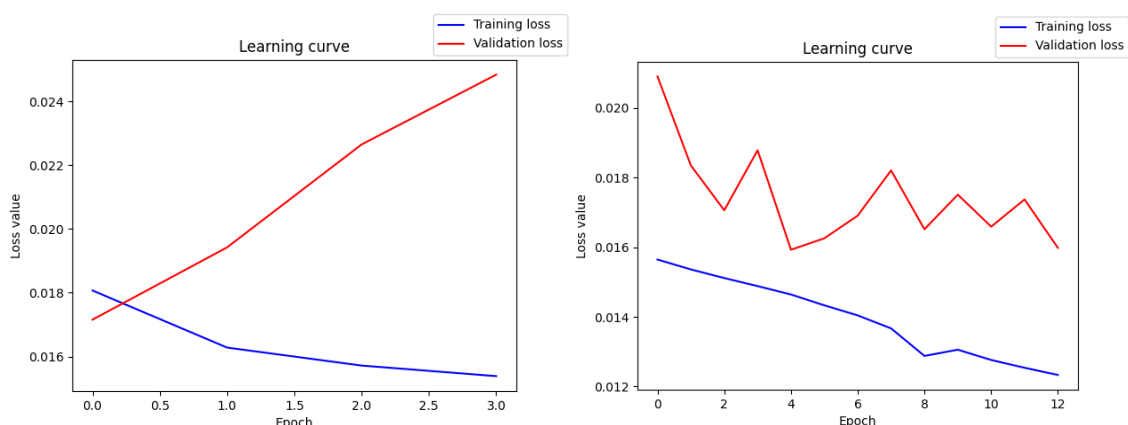


Figura 17: Curvas de aprendizaje de los modelos U-Net-1 y U-Net-2. Elaboración propia.

En lo que respecta a las U-Net, debido a que, como se puede observar de la tabla 7, tardan mucho más en entrenar, más de 3 horas por época y como mínimo necesitando unas cuantas de ellas para empezar a tener unos resultados asumibles, solo se han realizado experimentos probando con distintos valores del *dropout* y teniendo como *input* y objetivo, espectrogramas estándar y Mel-espectrogramas.

En caso de no aplicar *dropout* (modelo U-Net-1), se observa un claro *overfitting*, por otro lado, en aplicar demasiado *droupout* (modelo U-Net-3), hace que el modelo se estanque rápidamente en su aprendizaje y este ya no sea capaz de seguir aprendiendo. En aplicar mucho *droupout* se simplifica el modelo hasta el punto de no ser capaz de extraer patrones mínimos para su aprendizaje. Con un valor mediano como 0.3 (modelo U-Net-2), se observan los mejores resultados, permitiendo simplificar el modelo lo suficiente como para evitar tener mucho *overfitting*, pero no tanto como para deshabilitar la capacidad de aprendizaje.

Con lo que respecta a los datos de entrada y a predecir se observa que no hay mucha diferencia entre tener espectrogramas estándar o Mel-espectrogramas, sin embargo, estos últimos parecen ser más inestables en sus predicciones (tienen más variabilidad de error a lo largo de las épocas), donde, si además le sumamos que para la recuperación del audio se tienen que hacer más procesos que los espectrogramas estándar, perdiendo información por el camino, es preferible usar los espectrogramas estándar.

En lo que respecta al tiempo de entrenamiento por época, no hay una gran diferencia respecto a utilizar un tipo espectrograma u otro, a pesar de que el estándar tenga más datos de entrada y, por tanto, sea un modelo más pesado.

Comparando algunos de los espectrogramas predichos de las U-Net con los que se predijeron utilizando las otras arquitecturas de red (AE y VAE), aunque las pérdidas son más o menos las mismas, siendo un poco mejores en las U-Net, los espectrogramas predichos son de mucha mejor calidad, tal y como se puede observar en las imágenes de las figuras 18 y 19. Ahora ya no se generan simples “máscaras” que intentan minimizar la función de pérdidas global, haciendo como una media/suavizado de los valores, sino que esta vez los modelos sí parecen haber aprendido la estructura de los espectrogramas, intentando eliminar, aunque tampoco con mucho éxito, el ruido de los audios de las canciones.

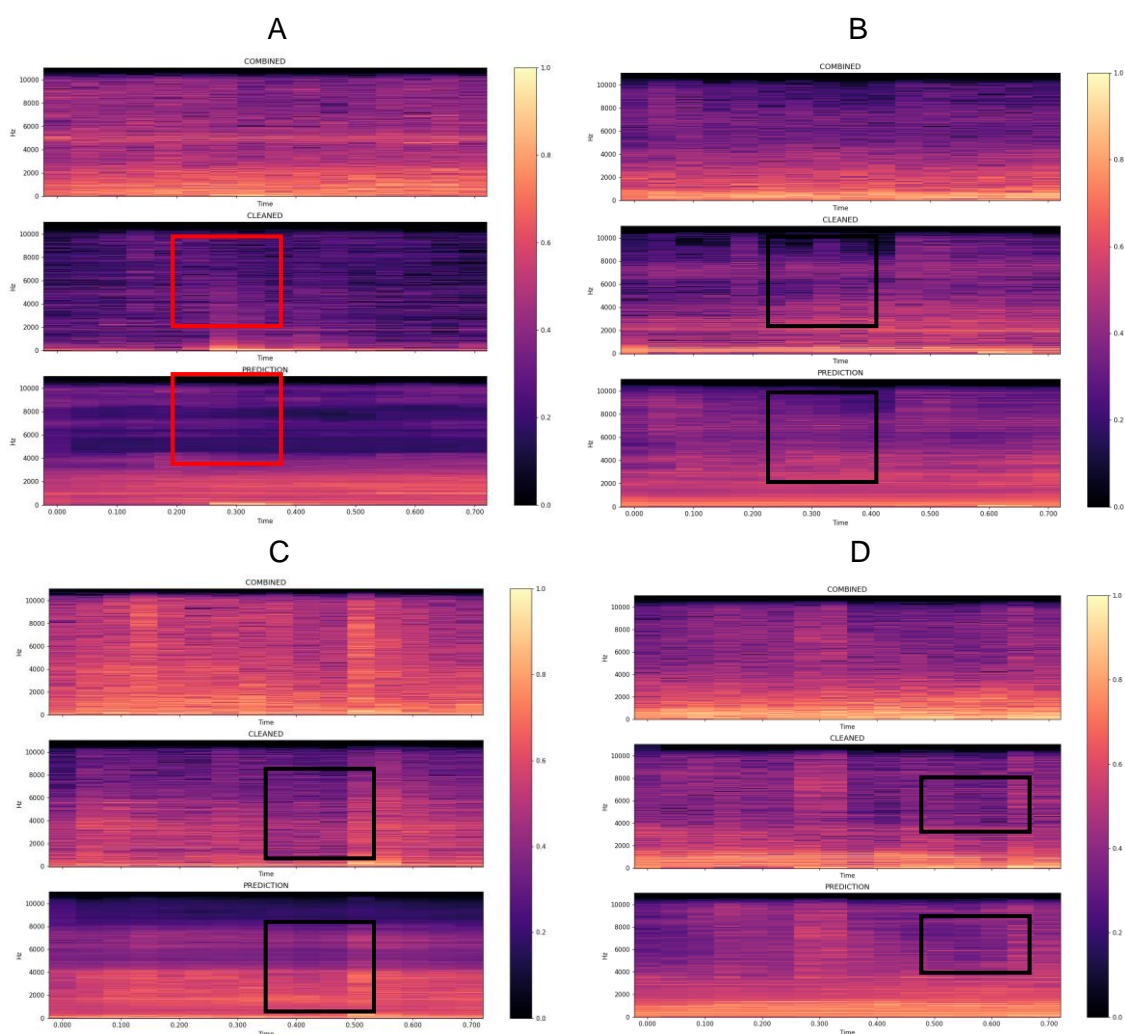


Figura 18: Muestras de espectrogramas predichos del modelo U-Net-2. Elaboración propia.

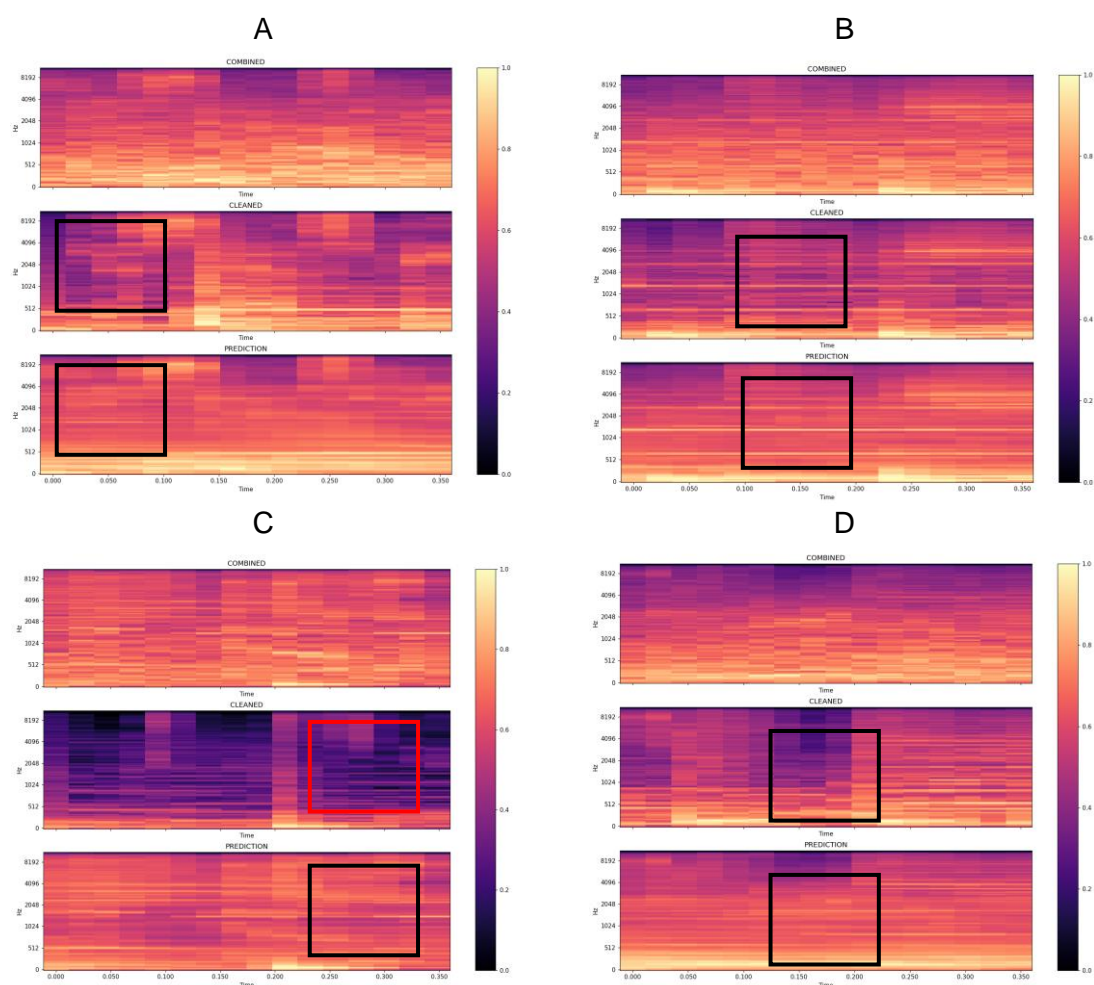


Figura 19: Muestras de espectrogramas predichos del modelo U-Net-Mel-1. Elaboración propia.

Efectivamente, vemos que los espectrogramas generados por las U-Net son mucho mejores que los generados por los modelos con arquitecturas AE y VAE. De este hecho, podemos concluir que el añadido de *skip-connections* ha sido un éxito, ya que estas, supuestamente debido a ser la principal diferencia entre este tipo de arquitecturas, han hecho posible la recuperación de la información inicial y dejando como faena para el *bottleneck* (o *latten space*) solo la parte de aprender donde se encuentra el ruido para que después, con el *decoder* y la ayuda de las *skip-connections*, sean capaces de eliminar el ruido y copiar la información de la canción.

Aprovechando que las U-Net han sido las que mejor resultado han dado y para verificar visualmente que ni con los mejores modelos generados se puede reconstruir una señal de audio (sonido) reconocible, a continuación, se mostrarán las formas del audio (*waveform*) reconstruidas a partir de los espectrogramas predichos por la U-Net-

2, así como se compararán con las que correspondían al audio original mezclado con ruido y sin ruido (la predicción deseada, el objetivo de las modelos).

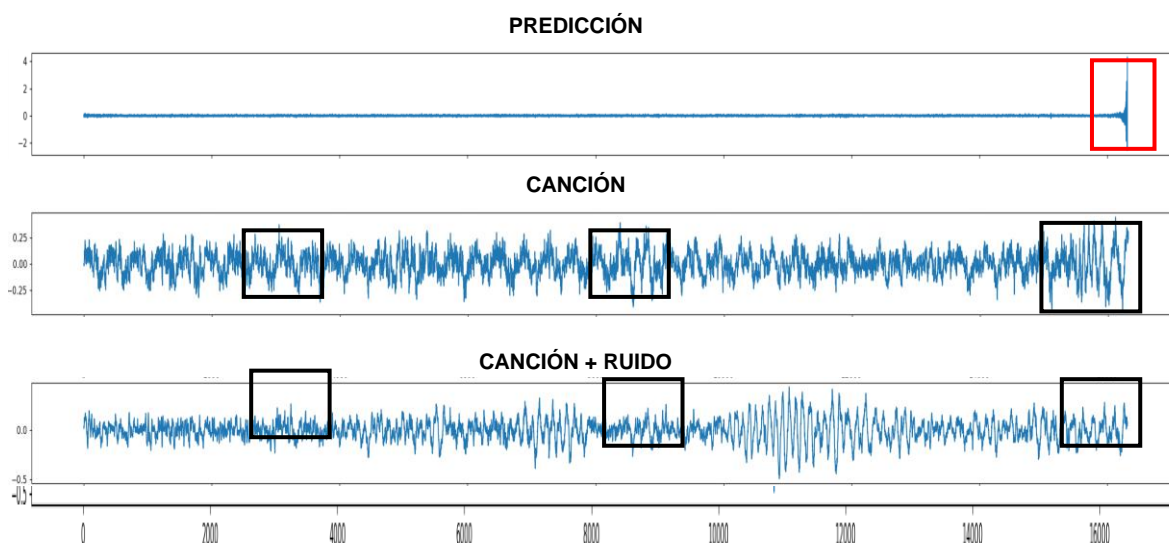


Figura 20: Muestra de las “waveform” de los audios originales y su predicción del modelo U-Net-2. Elaboración propia.

Mientras que en las formas del audio correspondientes al *input* del modelo y la canción (predicción deseada) tienen bastantes similitudes, tal y como se puede observar con las partes destacadas con rectángulos negros. Apparently this is not the case for the *waveform* generated from the predictions, where, to begin with, one can observe a significant alteration in its final part, a spike in amplitude (highlighted with a red rectangle).

Notwithstanding, to prove that it is not just about this spike, but that there is no resemblance with the other two *waveform*, in the figure 21, we show this same one but amplified, in such a way that the spike goes out of the range and with it we can examine with more clarity the shape of the audio for the rest of the prediction.

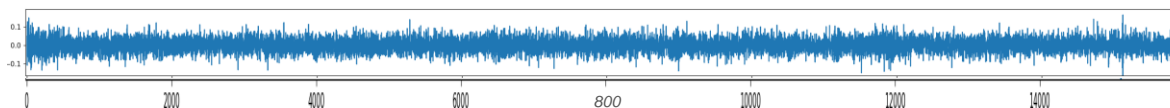


Figura 21: Muestra ampliada de la “waveform” de la predicción del modelo U-Net-2. Elaboración propia.

Effectively, it is perceived as not only no resemblance, but also the shape of the audio of the prediction is very similar to what one would expect from an audio that belonged to white noise (Gaussian).

6. Conclusiones

En este trabajo se ha intentado crear un modelo que sea capaz de eliminar el ruido de fondo en una canción. Para ello, se han empleado y comparado varios modelos con tres arquitecturas de red distintas. Las canciones usadas han sido mayoritariamente de los géneros musicales rock y pop con un ruido ambiente correspondiente al de una ciudad (tráfico) y cafetería (murmullo de gente). Las arquitecturas seleccionadas han sido los *autoencoders* (AE), *variational autoencoders* (VAE) y U-Net. En cada una de ellas se probaron diferentes configuraciones de hiperparámetros y dos *inputs* de entrada diferentes, el espectrograma y Mel-espectrograma, siendo estos también el objetivo a predecir.

Los hiperparámetros probados fueron el *learning rate*, con valores de $1e-3$, $1e-4$ y $2e-5$ usando la técnica de *plateau* con una espera de 3 épocas y factor decreciente de 0.2, la dimensión del *latent space* o *bottleneck*, con tamaños de 16, 128 y 256, el tamaño del *batch*, habiendo probado longitudes de 16, 32 y 128, y finalmente, para el caso de las VAE, también se probaron diferentes pesos para la función de pérdidas KL, siendo estos de 0.0001, 0.001, 0.005 y 1. Adicionalmente, para las U-Net al final de cada bloque convolucional se introdujo también una capa de *dropout* con valores de 0 (nulo), 0.3 y 0.5.

6.1. Valoración y discusión de los resultados

Los resultados obtenidos no fueron buenos, ningún modelo fue capaz de predecir un espectrograma lo suficientemente bueno como para poder reconstruir la señal y que su audio fuera reconocible, obteniendo resultados muy similares en casi todos los modelos. Sin embargo, los modelos con arquitecturas VAE y AE fueron los que peores resultados dieron, estimando espectrogramas muy simplistas, borrosos y generalistas, siendo todos muy similares entre ellos y teniendo más bien una representación de máscara más que de un espectrograma.

Se concluyó que la causa de los pésimos resultados fue muy probablemente por alguna, o algunas, de las siguientes razones.

- 1) Los modelos VAE no fueron capaces de generar buenas predicciones a partir de un espacio con distribución gaussiana dado su objetivo (predecir espectrogramas sin ruido) y datos iniciales (espectrogramas con ruido). Para ello sería preciso intentar buscar otro tipo de distribución para ver si esta se adaptara mejor al problema actual.
- 2) Los modelos no fueron lo suficientemente potentes como para extraer toda la información necesaria para generar espectrogramas decentes y de ellos poder reconstruir los audios sin ruido. No son capaces de extraer patrones frecuenciales ni temporales de calidad, probablemente debido a la falta de bloques con algún tipo de recurrencia para facilitar la tarea de reconocimiento de patrones temporales, ya que, los modelos U-Net, los que contienen *skip-connections*, la característica más cercana a la recurrencia, fueron los que mejores espectrogramas predijeron. Se recomienda entonces usar los mismos modelos con el uso de algún bloque recurrente para ver si hay mejoras.
- 3) Debido al uso y naturaleza de las arquitecturas de red AE, VAE y U-Net, naturaleza de clonar, reconstruir secciones corruptas pequeñas sobre los datos de entrada o generar segmentaciones, sobre todo para las dos primeras arquitecturas, junto al uso de la métrica MSE para la evaluación y entrenamiento de estos para analizar sus pérdidas de reconstrucción (minimizar la diferencia entre espectrogramas predichos y objetivo), ayudando al modelo a converger en una solución general y no muy precisa pero que se adapte relativamente bien para todos los datos, se dedujo que:
 - a. No se usaron las mejores arquitecturas de red para la utilización de la técnica *direct mapping*. Ya que dichas arquitecturas no están pensadas para estimar datos tan complejos y precisos como los requeridos para un espectrograma, sino más bien datos simples como para generar una máscara, y, por ende, usar la técnica *masking*.

Las VAE y AE están pensadas para que funcionen correctamente cuando solo tienen que estimar una parte compleja pero pequeña,

como una marca negra de pocos píxeles en una imagen, estimar imágenes de salida y entrada similares, estimar solo datos simples como una máscara a partir de unos datos de entrada o simplemente del espacio de dimensionalidad pequeña reconstruir la imagen original, es decir, estimar la misma imagen de entrada.

- b. Probablemente la función MSE no fue la óptima para ayudar a aprender espectrogramas con *direct mapping* para los modelos usados.
- 4) La elección de la duración de cada espectrograma (cantidad de tiempo que representa) no fue la más idónea (0.74 segundos), siendo los espectrogramas de mayor duración (alrededor de 10 segundos) los que, con bastante seguridad, se adaptaran mucho mejor en la tarea actual de eliminación del ruido a una canción, ya que los 0.74 segundos, fueron escogidos como referencia de estudios donde obtuvieron buenos resultados pero con otro tipo de tarea, generación de voz artificial para recitar frases cortas o incluso palabras sueltas pero no para tratar con canciones, segmentos de audio mucho más largos.

En resumen, de los experimentos y resultados obtenidos se ha visto que con un nivel de certeza elevado que los modelos con arquitecturas AE, VAE y U-Net sin bloques recurrentes y usando la técnica de *direct mapping* para sus predicciones, no son aptos para la eliminación del ruido ambiente en las canciones, o lo que es lo mismo, la generación directa de espectrogramas sin ruido. Es por ello que, para continuar con futuras investigaciones y ayudar en la búsqueda del modelo ideal para la eliminación de ruido, en el apartado 6.4) *Recomendaciones para trabajos futuros*, se da varias ideas de cómo se puede proceder utilizando las mismas arquitecturas de red, ya sean solas o combinadas con otras.

Por otro lado, de la hipótesis inicial nombrada en la introducción de esta memoria: “Los algoritmos que usen una arquitectura del tipo *encoder-decoder* con capas convolucionales y sin el uso de bloques recurrentes usando la técnica de *direct mapping*, son capaces de eliminar el ruido de una canción desde sus espectrogramas”, podemos afirmar con un grado de certeza bastante elevado de que es falsa.

A pesar de esto, se cree que, con más datos y horas de entrenamiento, así como el añadir algunas capas extras de recurrencia como algunos bloques de LSTM, BLSTM, GRU entre otros, para ayudar a los modelos en la extracción de características y relaciones temporales, se pueden obtener buenos resultados aun aplicando *direct mapping*. Incluso mejores que si se aplicara *masking* debido a la predicción directa, sin tener que pasar antes por una máscara.

Basándonos en la literatura y las observaciones de los experimentos realizados, el uso de bloques recurrentes para la obtención de buenos resultados parece prácticamente obligatorio siempre que se quiera usar la técnica *direct mapping*.

6.2. Cumplimiento de los objetivos

Del objetivo inicialmente marcado: “*Separar una canción de su ruido ambiente eliminándolo mediante modelos de aprendizaje profundo*”, finalmente no se ha alcanzado. Sin embargo, si se ha aprendido y avanzado en la comprensión del problema, creyendo cuáles pueden ser las técnicas y arquitecturas de red que deben de, ahora sí, funcionar mejor en la resolución de la tarea de eliminación de ruido.

Los hitos que se marcaron para el cumplimiento del objetivo principal, tales como el desarrollo de un *dataset* personalizado, una parte de código para poder reconstruir los espectrogramas a su audio correspondiente y la creación de los modelos AE, VAE y GAN, se han cumplido.

Por último, los objetivos secundarios también han sido alcanzados, ya que, aunque no se haya podido lograr el objetivo principal, durante el proceso de intentar lograrlo, si se considera haber obtenido una primera visión de cómo se aplica la inteligencia artificial al mundo del sonido y habla, así como, se ha llegado a terminar el proyecto al completo. Habiendo hecho una primera fase de estudio e investigación, tanto de los conceptos como del estado del arte, una implementación práctica de los algoritmos (parte de programación) y, finalmente, la elaboración de esta memoria con el análisis y conclusiones de los resultados obtenidos, incluso pudiendo dar consejos e ideas para futuras investigaciones relacionadas.

Con el primer vistazo, gracias al análisis del estado del arte y estudio de los conceptos, se ha logrado ver y entender de forma superficial las características más básicas del audio, como estas son extraídas y su utilidad, siendo cada una de ellas mejor para una u otra tarea relacionada con el audio, ya sea para la clasificación de sonidos, separación, eliminación o generación y los diferentes tipos de características que se tienen para tratar el audio.

También se han vislumbrado las diferentes técnicas y modelos que se pueden usar para cada tipo de característica, así como nuevamente, su aplicación práctica y los problemas por las que van mejor. Por ejemplo, se ha visto que se pueden usar varias arquitecturas de redes neuronales o, incluso, en algunos casos, como el de la clasificación de sonidos, es mejor aplicar técnicas más convencionales y tradicionales que no tengan redes neuronales, debido a su mayor simpleza, fluidez y rendimiento en algunos problemas.

Resumiendo, aun no habiendo podido alcanzar el objetivo de la eliminación de ruido, no se considera que este proyecto haya sido un fracaso, sino más bien todo lo contrario, ya que, como se acaba de mencionar, ha permitido desarrollar y mejorar las habilidades de investigación, organización, planificación, análisis y, sobre todo, entendimiento del dominio trabajado, viendo varias de las técnicas y características que se usan. En definitiva, es un proyecto que ha servido para expandir el conocimiento global de la IA, así como aprender nuevas técnicas y modelos al mismo tiempo que se refrescaban y asentaban algunos de los contenidos teóricos vistos durante el máster, una de las principales finalidades de un proyecto académico de esta tipología, un proyecto de fin de máster.

6.3. Problemas y contratiempos

Los problemas y contratiempos tenidos a lo largo del proyecto fueron varios.

El primero de ellos fue en la fase de implementación con la compatibilidad entre librerías y sus versiones. Este fue un gran contratiempo que retraso mucho más de lo esperado dicha fase. Este problema fue debido principalmente al uso de una plantilla para la creación de los modelos VAE y AE con tal de ir más rápido en el desarrollo.

El uso de la plantilla hizo que se tuviera que usar una versión antigua de la librería *tensorflow* para poder utilizarla y que esta funcionara, sin embargo, debido a ser una versión antigua muchos fueron los problemas de compatibilidad entre otras librerías y la modificación de los modelos en añadir y modificar las capas de sus redes neuronales ya existentes para la adaptación al proyecto actual.

Al mismo tiempo, durante la modificación de los modelos, debido a la falta de documentación de estos, a causa de la versión antigua de *tensorflow*, al final el uso de la plantilla para querer ir más rápido y tener el código ya con una estructura más o menos ordenada hizo que se fuera mucho más lento que si se hubiera empezado a construir los modelos desde cero con una versión más moderna de las librerías usadas.

Otro de los problemas principales fue la mala gestión en el control de configuraciones de los modelos ya entrenados o la falta de doble verificación del código antes de ejecutarlo. Esto hizo que en algunos casos se tuviera que repetir varios de los entrenamientos por la falta de no haber apuntado alguna información vital, como el tiempo de entrenamiento o las pérdidas de validación o, aún peor, por la falta de atención en introducir varias configuraciones de hiperparámetros que ya habían sido entrenados o haberse equivocado en la asignación de valores en alguno de ellos, ya sea en los datos de entrada o en entrenar modelos con los mismos datos para la validación y entrenamiento.

La solución encontrada, aunque ya tarde, fue la creación de un fichero (por ejemplo, de Excel) para ir apuntando toda la información y configuraciones probadas en lugar de acordarse de memoria o crear carpetas con nombre identificativo e informativo de sus resultados, pero no suficientes, siendo caóticos de encontrar debido al desorden generado por la gran cantidad de carpetas generadas (una por modelo y configuración diferente).

En lo que respecta a la creación del dataset, debido a su gran cantidad de datos y tamaño, cada canción ocupa varios MB, así como la aparente sencillez de crearlos, juntar solo las canciones con el ruido ambiente y separarlo en varios subconjuntos para el entrenamiento, validación y test, hizo que directamente se programara el

código necesario para crearlo y separarlo de forma automática sin antes probarlo con un conjunto de datos más pequeños para comprobar su correcto funcionamiento.

El problema fue, que después de varias horas de ejecución, necesarias para juntar y separar los datos debido a su volumen y peso, un pequeño error en el código hizo que el *dataset* generado tuviera errores y se tuviera que volver a crear varias veces, perdiendo tiempo de entrenamiento de los modelos debido a que, sin datos, no hay entrenamiento posible. En definitiva, para un futuro, aunque el problema de programación parezca sencillo, vale la pena perder, o, mejor dicho, ganar, algo de tiempo adicional probando primero que todo funcione correctamente con un subconjunto de datos más pequeño.

Finalmente, otro imprevisto fue el tener que repasar e incluso en algunos casos tener que estudiar desde cero algunos de los conceptos musicales y teoría de señales, perdiendo otra vez más tiempo del previsto originalmente. Así que, para un trabajo académico de carácter individual y con tiempo limitado, para un futuro será mejor elegir un tema que se domine más de antemano y guardar tiempo para solucionar otros problemas que no se puedan prever, para luego, tener más tiempo para generar más modelos, probar con otras técnicas como *masking*, o probar con más *datasets* distintos para ver la generalización de los modelos creados.

6.4. Recomendaciones para futuros trabajos

Para futuros trabajos, con base en los resultados obtenidos y lo aprendido a lo largo de este proyecto a continuación se proponen varias mejoras e implementaciones para seguir investigando en la búsqueda del modelo ideal que permita separar el ruido de las canciones de forma perfecta, sin la generación de artefactos ni fugas de otras fuentes.

Para ello, la primera recomendación es continuar con la técnica de *masking* y olvidarse del *direct mapping* en el caso de seguir utilizando las arquitecturas VAE, AE o U-Net, tal y como hace la mayoría de la literatura actual que usa redes neuronales para generar audio mediante espectrogramas. Así, el nuevo objetivo de los modelos

debe ser estimar una máscara, para después, multiplicarla con el espectrograma de entrada y generar así el espectrograma predicho sin ruido, solo la canción, en lugar de ir a generar directamente el espectrograma limpio de ruido.

Sin embargo, en caso contrario y querer continuar con la técnica de *direct mapping*, así como seguir usando las mismas arquitecturas, la primera mejora y casi obligatoria debe ser la ampliación de la complejidad de las redes mediante con el uso de bloques recurrentes, ya sean LSTM, BLSTM, RNN, GRU, entre otros.

Proceder investigando con VAE y AE usando *direct mapping* y sin usar ningún tipo de recurrencia no se cree que se puedan llegar a obtener buenos resultados debido a su naturaleza, teniendo a tener *overfitting* o generar predicciones muy simplistas, ideales para problemas de segmentación como *masking*, pero no de *direct mapping*. Es por ello que, a priori, no se recomienda seguir investigando con dicha combinación (VAE o AE junto a *direct mapping*).

No obstante, tal y como hacen algunos estudios [6], [13], [14], aunque estos utilicen la técnica de *masking* en lugar de *direct mapping*, se cree poder usar una mena de recurrencia. El objetivo de los modelos de estos estudios es predecir solo un *frame* (*bin* temporal) en lugar de todo el espectrograma, aunque a estos modelos se les pase el espectrograma entero como datos de entrada (*input*). Con ello, el modelo puede utilizar los *bins* del espectrograma del *input* como información extra para ayudar al modelo a predecir un *bin* en base a su contexto (el propio espectrograma, sus demás *frames*).

El *frame* a predecir puede ser un hiperparámetro adicional a experimentar, pudiendo ser el primero, el último, uno intermedio o incluso el previo o posterior del espectrograma pasado como *input*. En la figura 22 se muestra el concepto.

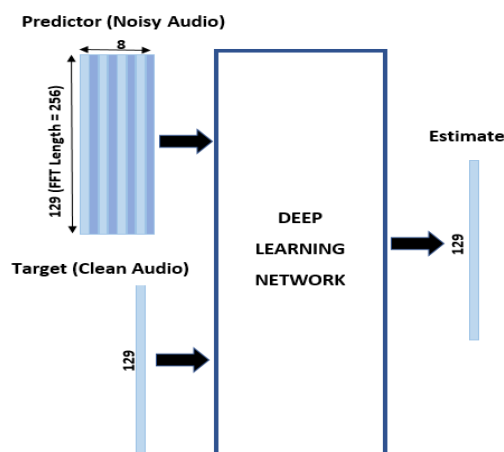


Figura 22: Arquitectura propuesta de input/output. Imagen extraída de [14].

Otra propuesta de mejora para intentar generar espectrogramas más nítidos y no tan borrosos, es la utilización de otro tipo de arquitecturas como las GAN, combinadas o no con las arquitecturas actuales AE, VAE o U-Net. Su funcionalidad es sustituir a la función de optimización MSE y dejar que sea la red GAN quien aprendiera la función de pérdidas durante el propio entrenamiento de la red. Una opción más sencilla puede ser usar meramente otro tipo de función de coste como la BCE (*binary cross entropy*) en lugar de la MSE.

En lo que respecta a los hiperparámetros, en caso de que quisiera seguir comprobando con algunos valores, en vista de los resultados obtenidos, se recomienda sobre todo probar con más valores de diferentes tamaños para el *batch size*, ya que, de los resultados obtenidos, parece ser que es uno de los hiperparámetros que más afectan en la calidad de las predicciones y el tiempo de entrenamiento requerido, así como, sobre todo, probar con diferentes valores de *frame size* y *hop size (overlapping)*, puesto que estos dos últimos, por falta de tiempo, no se han podido comprobar con más de un valor diferente, 2048 y 1024 (50%) respectivamente.

Para el *batch size*, se recomienda usar tamaños grandes, superiores a 500, ya que son los que se han observado que obtienen rendimientos similares o mejores que los modelos con un tamaño de *batch size* más pequeño, pero con un tiempo de entrenamiento mucho menor para alcanzar el mismo conocimiento o aprendizaje,

permitiendo así entrenar muchas más épocas para dar a los modelos más oportunidades de extraer buenos patrones y realizar mejores predicciones.

Otros hiperparámetros que se creen que pueden dar mejor resultado, y, tampoco se han comprobado para más de un valor, es la duración que representa cada espectrograma, recomendando para ello que tengan una duración sobre los 10 segundo, puesto que representan segmentos de canciones y estas normalmente tienen una duración total de unos 3 o 4 minutos.

Se recomienda también, cambiar el número de capas y neuronas de las redes empleadas o, en su defecto, añadir regularizadores extra en las capas neuronales, no usar solo el *batch normalization*, cuya funcionalidad principal no es la de regularizar, *dropout* o el peso de la pérdida KL en las VAE. Algunos ejemplos son la aplicación de una L1, L2, ElasticNet (L1+ L2), tanto en los pesos de la red como en la salida de las funciones de activación o en las neuronas de *bias*, así como añadir más datos variados en el conjunto de entrenamiento.

En lo que respecta a la configuración de hiperparámetros, también puede estar bien probar con otro rango de normalización de los espectrogramas tales como, que sus valores vayan entre -1 y 1, usando la función de activación *tanh* en la capa de salida o de predicción, en lugar de normalizar entre 0 y 1 usar la función sigmoide como se ha hecho en el estudio actual.

Por lo que respecta a los hiperparámetros como la función de *windowing* (actualmente la Hann), funciones de activación para las capas intermedias (actualmente la LeakyReLU) o, incluso el tipo de optimizador (actualmente el Adam), no son hiperparámetros que se crea que cambiando su valor los modelos pasen a mejorar de forma exponencial.

Por último, y por falta de tiempo y predicciones no óptimas, sin haberse implementado en el proyecto, puede estar bien ampliar su fase de evaluación añadiendo un *dataset* de test extraordinario donde sus canciones y ruidos ambientales fueran diferentes versus a los conjuntos usados para entrenar y validar, pudiendo comprobar de esta manera la generalización de los modelos a otro tipo de datos, permitiendo tener un ítem más en la selección y evaluación del mejor modelo. También se podría entrenar

con un *dataset* más general, con más géneros y tipo de ruidos, para ayudar a los modelos a generalizar.

Por otro lado, también es interesante implementar otras métricas de evaluación que no evalúen a los espectrogramas en sí, sino al audio que se reconstruyera de estos, ya que, al final, la intención no es tener un espectrograma sin ruido sino un audio sin ruido y que este sea reconocible y bueno. Estas nuevas métricas deben ser perceptuales. Algunos ejemplos pueden ser la SIR o SNR, midiendo que tan bueno o similares son los audios resultantes respecto a los deseados, mediante ratios que miran la cantidad de ruido original mantenido o artefactos generado en las reconstrucciones de las predicciones versus al nivel máximo deseado o establecido.

6.5. Aplicaciones extraordinarias

En caso de que finalmente se pudiera lograr obtener un método que pudiera separar perfectamente el audio de su ruido ambiente, a parte de la aplicación implícita, también se puede extender a otro tipo de audios, ya sea con distintos ruidos o canciones, incluso combinado con otro tipo de fuentes sonoras como voces, otros géneros musicales, con o sin instrumentos de percusión, etcétera.

Sus implicaciones inmediatas pueden ser la cancelación de audio o ruido general para lograr concentrarse mejor, eliminación de fuentes no deseadas como aplausos y risas enlatadas molestas de algunas series, o incluso facilitar otros modelos a conseguir sus objetivos, como, por ejemplo, la eliminación del ruido para facilitar a los traductores y transcriptores automáticos a detectar mejor las palabras.

7. Referencias

- [1] P. Guduguntla, «Background Noise Removal: Traditional vs AI Algorithms», mar. 18, 2021. <https://towardsdatascience.com/background-noise-removal-traditional-vs-ai-algorithms-9e7ec5776173> (accedido ago. 19, 2022).
- [2] D. Baghdasaryan, «Real-Time Noise Suppression Using Deep Learning», *Nvidia Developer*, oct. 31, 2018. <https://developer.nvidia.com/blog/nvidia-real-time-noise-suppression-deep-learning/> (accedido ago. 19, 2022).
- [3] Startingtodj, «Isolating Instruments in Songs – Best Tools up to Date (2022)», 2022. <https://startingtodj.com/isolating-instruments-in-songs-best-tools-up-to-date/> (accedido ago. 19, 2022).
- [4] Rory PQ, «What Are the Different Types of EQ and Filters? | Icon Collective», jun. 19, 2019. <https://iconcollective.edu/types-of-eq/> (accedido ago. 19, 2022).
- [5] M. B. Ercan, «MUSICAL INSTRUMENT SOURCE SEPARATION IN UNISON AND MONAURAL MIXTURES», 2014.
- [6] T. S. Silva, «Practical Deep Learning Audio Denoising», dic. 18, 2019. <https://sthalles.github.io/practical-deep-learning-audio-denoising/> (accedido ago. 19, 2022).
- [7] R. Hennequin, A. Khlif, F. Voituret, y M. Moussallam, «Spleeter: a fast and efficient music source separation tool with pre-trained models», *J Open Source Softw*, vol. 5, n.º 50, p. 2154, jun. 2020, doi: 10.21105/JOSS.02154.
- [8] F.-R. Stöter, S. Uhlich, A. Liutkus, y Y. Mitsufuji, «Open-Unmix - A Reference Implementation for Music Source Separation», *J Open Source Softw*, vol. 4, n.º 41, p. 1667, sep. 2019, doi: 10.21105/JOSS.01667.
- [9] A. Défossez, «Hybrid Spectrogram and Waveform Source Separation», nov. 2021, doi: 10.48550/arxiv.2111.03600.

- [10] Oksana, «Meet LALAL.AI Cassiopeia: A Milestone in Music Source Separation», mar. 01, 2021. <https://www.lalal.ai/blog/meet-lalal-ai-cassiopeia-a-milestone-in-music-source-separation/> (accedido ago. 19, 2022).
- [11] L. Oudre, «Automatic Detection and Removal of Impulsive Noise in Audio Signals», *Image Processing On Line*, vol. 5, pp. 267-281, nov. 2015, doi: 10.5201/IPOL.2015.64.
- [12] M. Dogra, S. Borwankar, y J. Domala, «Noise Removal from Audio Using CNN and Denoiser», pp. 37-48, 2021, doi: 10.1007/978-981-33-6881-1_4.
- [13] S. R. Park y J. W. Lee, «A Fully Convolutional Neural Network for Speech Enhancement», *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH*, vol. 2017-August, pp. 1993-1997, sep. 2016, doi: 10.48550/arxiv.1609.07132.
- [14] Inc. The MathWorks, «Denoise Speech Using Deep Learning Networks». <https://www.mathworks.com/help/deeplearning/ug/denoise-speech-using-deep-learning-networks.html> (accedido ago. 20, 2022).
- [15] M. Zhao, D. Wang, Z. Zhang, y X. Zhang, «Music removal by convolutional denoising autoencoder in speech recognition», *APSIPA ASC*, pp. 338-341, feb. 2016, doi: 10.1109/APSIPA.2015.7415289.
- [16] A. Jansson, E. J. Humphrey, N. Montecchio, R. M. Bittner, A. Kumar, y T. Weyde, «Singing Voice Separation with Deep U-Net Convolutional Networks», *ISMIR*, oct. 2017.
- [17] F. López Soler, «Music Source Separation Using Deep Neural Networks», Universitat politècnica de catalunya, Barcelona, 2020. Accedido: ago. 20, 2022. [En línea]. Disponible: <https://upcommons.upc.edu/handle/2117/329260>.
- [18] Y. Ke, A. Li, C. Zheng, R. Peng, y X. Li, «Low-complexity artificial noise suppression methods for deep learning-based speech enhancement algorithms», *EURASIP J Audio Speech Music Process*, vol. 2021, n.º 1, pp. 1-15, dic. 2021, doi: 10.1186/S13636-021-00204-9/FIGURES/5.

- [19] O. Slizovskaia, G. Haro, y E. Gómez, «Conditioned Source Separation for Music Instrument Performances», *IEEE/ACM Trans Audio Speech Lang Process*, vol. 29, pp. 2083-2095, abr. 2020, doi: 10.1109/TASLP.2021.3082331.
- [20] L. Pretet, R. Hennequin, J. Royo-Letelier, y A. Vaglio, «Singing Voice Separation: A Study on Training Data», *ICASSP, IEEE*, vol. 2019-May, pp. 506-510, may 2019, doi: 10.1109/ICASSP.2019.8683555.
- [21] E. Manilow, G. Wichern, y J. LeRoux, «Hierarchical Musical Instrument Separation», en *International Society for Music Information Retrieval (ISMIR) Conference*, oct. 2020, pp. 376-383. Accedido: ago. 19, 2022. [En línea]. Available: <https://www.merl.com/publications/TR2020-136>
- [22] F. Roche, T. Hueber, S. Limier, y L. Girin, «Autoencoders for music sound modeling: a comparison of linear, shallow, deep, recurrent and variational models», *Proceedings of the Sound and Music Computing Conferences*, pp. 415-422, jun. 2018, doi: 10.48550/arxiv.1806.04096.
- [23] D. Wang, «Time-Frequency Masking for Speech Separation and Its Potential for Hearing Aid Design», *Trends Amplif*, vol. 12, n.º 4, pp. 332-353, 2008, doi: 10.1177/1084713808326455.
- [24] F.-R. Stöter, S. Uhlich, A. Liutkus, y Y. Mitsufuji, «Open-Unmix - A Reference Implementation for Music Source Separation», *J Open Source Softw*, vol. 4, n.º 41, p. 1667, sep. 2019, doi: 10.21105/JOSS.01667.
- [25] Scrum.org, «What is Scrum?» <https://www.scrum.org/resources/what-is-scrum> (accedido ago. 20, 2022).
- [26] Atlassian, «Kanban - A brief introduction». <https://www.atlassian.com/agile/kanban> (accedido ago. 20, 2022).
- [27] TeamGantt, «TeamGantt: Online Gantt Chart Maker Software». <https://www.teamgantt.com/> (accedido ago. 20, 2022).
- [28] diagrams.net. <https://app.diagrams.net/> (accedido ago. 20, 2022).

- [29] V. Velardo, «Generating Sound with Neural Networks», 2021. <https://www.youtube.com/playlist?list=PL-wATfeyAMNpEyENTc-tVH5tfLGKtSWPp> (accedido ago. 22, 2022).
- [30] A. Kumar, «Deep Learning 21: (3) Variational AutoEncoder : Working details of Variational AutoEncoder», ene. 20, 2019. <https://www.youtube.com/watch?v=YHldNC1SZVk> (accedido ago. 26, 2022).
- [31] DigitalSreeni (Youtube), «Understanding U-Net architecture and building it from scratch», abr. 27, 2021. <https://www.youtube.com/watch?v=GAYJ81M58y8> (accedido ago. 29, 2022).
- [32] R. Aduviri y A. la Fuente, «LXAI at ICML - Poster», 2019. [https://hal.archives-ouvertes.fr/hal-02266937/file/LXAI at ICML - Poster.pdf](https://hal.archives-ouvertes.fr/hal-02266937/file/LXAI_at_ICML_-_Poster.pdf) (accedido ago. 24, 2022).
- [33] D. Godoy, «Understanding AutoEncoders with an Example: A Step-by-Step Tutorial», jun. 07, 2022. <https://towardsdatascience.com/understanding-autoencoders-with-an-example-a-step-by-step-tutorial-a79d2ea2945e> (accedido ago. 24, 2022).
- [34] «Chapter 14: Autoencoders». <https://www.deeplearningbook.org/contents/autoencoders.html> (accedido ago. 24, 2022).
- [35] N. Singh Chauhan, «Introduction to AutoEncoder and Variational AutoEncoder (VAE)», jul. 28, 2021. <https://www.theaidream.com/post/an-introduction-to-autoencoder-and-variational-autoencoder-vae> (accedido ago. 24, 2022).
- [36] K. Frans, «Variational Autoencoders Explained», ago. 05, 2016. <https://kvfrans.com/variational-autoencoders-explained/> (accedido ago. 24, 2022).
- [37] S. Saha, «A Comprehensive Guide to Convolutional Neural Networks», dic. 15, 2018. <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53> (accedido ago. 24, 2022).

- [38] E. Manilow, P. Seetharaman, y J. Salamon, «Open Source Tools & Data for Music Source Separation», 2020. <https://source-separation.github.io/tutorial/landing.html> (accedido ago. 22, 2022).
- [39] V. Dumoulin, F. Visin, y G. E. P. Box, «A guide to convolution arithmetic for deep learning», mar. 2016, doi: 10.48550/arxiv.1603.07285.
- [40] NATIONAL INSTRUMENTS CORP., «Comprender FFTs y Funciones Ventana», ago. 09, 2022. <https://www.ni.com/es-es/innovations/white-papers/06/understanding-ffts-and-windowing.html> (accedido ago. 26, 2022).
- [41] H. Bandyopadhyay, «Autoencoders in Deep Learning: Tutorial & Use Cases», jul. 19, 2022. <https://www.v7labs.com/blog/autoencoders-guide> (accedido ago. 26, 2022).
- [42] A. Asperti y M. Trentin, «Balancing reconstruction error and kullback-leibler divergence in variational autoencoders», *IEEE Access*, vol. 8, pp. 199440-199448, 2020, doi: 10.1109/ACCESS.2020.3034828.
- [43] R. Aduviri, A. De, y L. Fuente, «Towards Disentangled Representations via Variational Sparse Coding», jun. 2019, Accedido: ago. 24, 2022. [En línea]. Available: <https://hal.archives-ouvertes.fr/hal-02266937>
- [44] E. Mathieu, T. Rainforth, N. Siddharth, y Y. W. Teh, «Disentangling Disentanglement in Variational Autoencoders», *36th International Conference on Machine Learning, ICML 2019*, vol. 2019-June, pp. 7744-7754, dic. 2018, doi: 10.48550/arxiv.1812.02833.
- [45] X. Guo, X. Liu, E. Zhu, y J. Yin, «Deep Clustering with Convolutional Autoencoders», en *Lecture Notes in Computer Science*, 2017, vol. 10635 LNCS, pp. 373-382. doi: 10.1007/978-3-319-70096-0_39.
- [46] Z. Rafii, A. Liutkus, F. R. Stoter, S. I. Mimilakis, D. Fitzgerald, y B. Pardo, «An Overview of Lead and Accompaniment Separation in Music», *IEEE/ACM Trans Audio Speech Lang Process*, vol. 26, n.º 8, pp. 1307-1335, ago. 2018, doi: 10.1109/TASLP.2018.2825440.

- [47] B. Leshowitz, «Measurement of the Two-Click Threshold», *J Acoust Soc Am*, vol. 49, n.º 2B, p. 462, ago. 2005, doi: 10.1121/1.1912374.
- [48] Geek3, «Window functions».
https://commons.wikimedia.org/wiki/File:Mplwp_window-functions-symmetric.svg
 (accedido ago. 24, 2022).
- [49] V. Valerdo, «Complex Numbers for Audio Signal Processing», ago. 13, 2020.
<https://www.youtube.com/watch?v=DgF4m0AWCgA> (accedido ago. 25, 2022).
- [50] N. Tomar, «What is UNET?», ene. 19, 2021. <https://medium.com/analytics-vidhya/what-is-unet-157314c87634> (accedido ago. 25, 2022).
- [51] H. Lamba, «Understanding Semantic Segmentation with UNET», feb. 17, 2019.
<https://towardsdatascience.com/understanding-semantic-segmentation-with-unet-6be4f42d4b47> (accedido ago. 25, 2022).
- [52] O. Ronneberger, P. Fischer, y T. Brox, «U-net: Convolutional networks for biomedical image segmentation», *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9351, pp. 234-241, 2015, doi: 10.1007/978-3-319-24574-4_28/COVER.
- [53] S. Pigeon, «RESTAURANT AMBIENCE • 10H Busy Coffee Shop Background Noise», jul. 29, 2018. https://www.youtube.com/watch?v=h2zkV-l_TbY (accedido ago. 22, 2022).
- [54] White Noise Soundscapes, «City Traffic White Noise», abr. 06, 2017.
<https://www.youtube.com/watch?v=8s5H76F3SIs> (accedido ago. 22, 2022).
- [55] faroit *et al.*, «MUSDB18», 2018. <https://sigsep.github.io/> (accedido ago. 22, 2022).
- [56] faroit, «tracklist.csv», ago. 28, 2018.
<https://github.com/sigsep/website/blob/master/content/datasets/assets/tracklist.csv>
 (accedido ago. 22, 2022).

- [57] W. Weng y X. Zhu, «INet: Convolutional Networks for Biomedical Image Segmentation», *IEEE Access*, vol. 9, pp. 16591-16603, 2021, doi: 10.1109/ACCESS.2021.3053408.
- [58] V. Velardo, «Defining the Fourier Transform with Complex Numbers», ago. 17, 2020. <https://www.youtube.com/watch?v=KxRmbtJWUzI&list=PL-wATfeyAMNqlee7cH3q1bh4QJFAaeNv0&index=12> (accedido ago. 29, 2022).
- [59] V. Velardo, «Demystifying the Fourier Transform: The Intuition», ago. 10, 2020. <https://www.youtube.com/watch?v=XQ45IgG6rJ4&list=PL-wATfeyAMNqlee7cH3q1bh4QJFAaeNv0&index=10> (accedido ago. 29, 2022).
- [60] V. Velardo, «Audio Signal Processing for Machine Learning», jun. 18, 2020. <https://www.youtube.com/watch?v=iCwMQJnKk2c&list=PL-wATfeyAMNqlee7cH3q1bh4QJFAaeNv0&index=1> (accedido ago. 29, 2022).
- [61] V. Velardo, «Extracting Mel Spectrograms with Python», sep. 17, 2020. https://www.youtube.com/watch?v=TdnVE5m3o_0&list=PL-wATfeyAMNqlee7cH3q1bh4QJFAaeNv0&index=18 (accedido ago. 29, 2022).
- [62] V. Velardo, «Mel-Frequency Cepstral Coefficients Explained Easily», oct. 05, 2020. https://www.youtube.com/watch?v=4_SH2nfbQZ8&list=PL-wATfeyAMNqlee7cH3q1bh4QJFAaeNv0&index=19 (accedido ago. 29, 2022).

8. Anexos

8.1. Anexo I: Explicación detallada de la FT y la creación de los espectrogramas

Para calcular cada componente o valor de cada frecuencia, la FT se sirve de la propiedad matemática donde si dos señales se multiplican entre ellas y posteriormente se calcula su área resultante, se aplica la integral, esta será más grande cuando más similares sean las dos señales originales que se han usado para multiplicar. El área será positiva o negativa en función de la diferencia de fases que se tengan en ambas señales a comparar. La magnitud o amplitud del espectrograma para cada frecuencia en cada *bin* temporal será precisamente el valor del área en valor absoluto.

Para seleccionar la función o señal a multiplicar por la original se siguen los siguientes pasos:

- 1) Seleccionar una frecuencia determinada y que sea más baja que la de Nyquist.
- 2) Optimizar la fase de la señal, aplicando alguna técnica de optimización de tal forma que se maximice el área calculada en el siguiente paso 3).
- 3) Realizar la multiplicación entre ambas señales, obteniendo así el valor de la magnitud.

Matemáticamente los pasos anteriores se definen con las fórmulas:

$$\varphi_f = \operatorname{argmax}_{\varphi \in [0,1)} \left(\int s(t) \cdot \sin(2\pi(f \cdot t - \varphi)) dt \right)^{(1)}$$

$$d_f = \max_{\varphi_f} \left(\int s(t) \cdot \sin(2\pi(f \cdot t - \varphi)) dt \right)^{(2)}$$

Siendo $s(t)$ la señal original y d_f el área, energía o magnitud que tiene la señal original para una determinada frecuencia y fase.

Para expresar dichas ecuaciones de forma más simple y compacta, se usa la representación polar, pasando las fórmulas anteriores al plano complejo. Para ello hay que recordar que un número complejo es en realidad un punto dentro del plano complejo o polar. Analíticamente expresado de la forma $a + b \cdot i$ siendo a la parte real o valor en el eje de las abscisas y b el número imaginario o valor en el eje de las coordenadas. La siguiente ilustración puede ayudar a entender la definición.

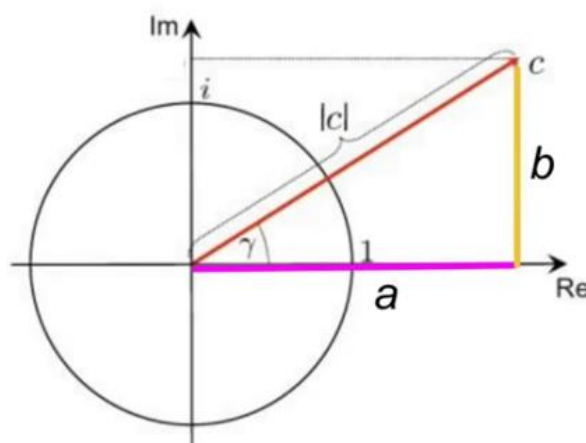


Figura 23: Ilustración del plano complejo. Imagen extraída del video [49] en el minuto 8:45.

Donde $|c|$ representa la magnitud del vector formado por a y b ($\sqrt{a^2 + b^2}$) y γ el ángulo ($\arctan(b/a)$).

Con la fórmula de Euler, definida como: $e^{i \cdot x} = \cos x + i \cdot \sin x$ ⁽³⁾, la expresión de un número complejo (c) en forma polar queda definida como: $c = |c| \cdot (\cos \varphi + i \cdot \sin \varphi) = |c| \cdot e^{i\gamma}$ ⁽⁴⁾.

Transformando ahora las ecuaciones de la transformada de Fourier a coordenadas polares complejas, nos quedan reducidas al número complejo (c_f), donde este es capaz de contener la información de ambos valores, el de la fase y magnitud de la señal. A c_f se técnicamente se le llama coeficientes complejos de la FT.

Si $g(t) = s(t)$ y $\hat{g}(f) = d_f$, entonces:

$$\hat{g}(f) = \int g(t) \cdot e^{-i \cdot 2\pi \cdot f \cdot t} dt = \int g(t) \cos(-2\pi f t) dt + i \int g(t) \sin(-2\pi f t) dt \quad (5)$$

$$\hat{g}(f) = c_f = \frac{d_f}{\sqrt{2}} \cdot e^{-i \cdot 2\pi \cdot \varphi \cdot f} \quad (6)$$

Donde $\sqrt{2}$ es simplemente un término de normalización, proveniente de aplicar el teorema de Pitágoras al círculo polar unitario.

Aislando los parámetros de c_f , podemos obtener la magnitud y fase:

$$d_f = \sqrt{2} \cdot |\hat{g}(f)| \quad (7)$$

$$\varphi_f = -\frac{\gamma_f}{2\pi} \quad (8)$$

Visualmente, podemos observar la función resultante de multiplicar ambas funciones en el plano complejo (parte de dentro de la integral, el integrando).

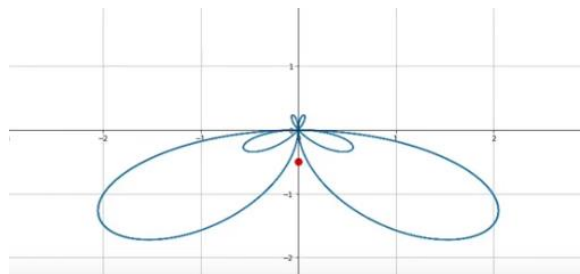


Figura 24: Ejemplo particular de un integrando $\hat{g}(f)$. Imagen extraída del video [58] en el minuto 24:18.

Siendo la curva azul la función del integrando y el punto rojo el centro de gravedad o media de la función evaluada en cada uno de sus puntos o coordenadas. Este punto coincide con el valor que nos devuelve la integral cuando esta es evaluada en un intervalo de tiempo igual al periodo de la señal resultante de la multiplicación, o lo que es lo mismo, el integrando. Dado que la integral nos devuelve la suma de todas las coordenadas de una función para un espacio continuo, sería el sumatorio en un espacio discreto. El valor que esta nos devuelva dependerá del intervalo a la que la evaluemos y su función del integrando, dándonos como resultado en nuestro caso el valor medio mencionado (punto verde) multiplicado por el intervalo de tiempo.

Si en FT aplicamos una frecuencia que no corresponde a la señal original $g(t)$, observamos que tanto el punto verde como rojo son cero, esto es debido a que todos los puntos de la gráfica azul se cancelan entre ellos, por tanto, podemos ver como haciendo esto para todas las frecuencias y cogiendo el punto rojo o verde que frecuencias corresponden a $g(t)$ y cuáles no. Un ejemplo gráfico de una frecuencia ligeramente desviada de la de la señal original nos puede devolver un resultado como el siguiente:

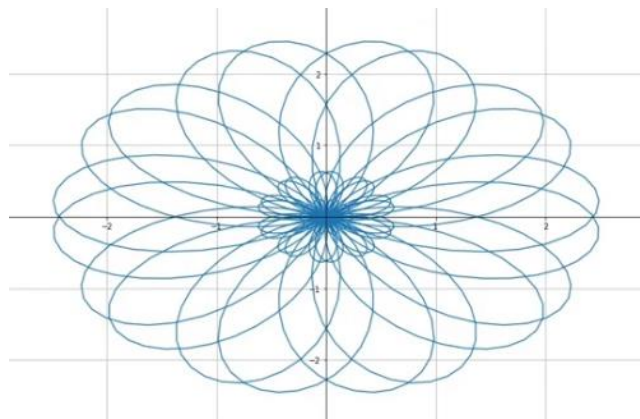


Figura 25: Integrando $\hat{g}(f)$ de frecuencia 1,1. Imagen extraída del video [58] en el minuto 21:15.

Cabe mencionar que, en base a experimentos realizados, se ha observado que los humanos para poder distinguir un evento sonoro de otro sin que ambos los percibamos como uno único, necesitamos una duración mínima de 10ms, de otra forma no podremos percibirlos como eventos acústicos. La resolución temporal, es, por tanto, el tiempo que dura un sonido o evento en concreto. Este dato será importante tenerlo en cuenta en la transformada de Fourier en el momento de seleccionar el tamaño de ventana, ya que uno donde la resolución temporal sea inferior a 10ms no tendrá sentido.

Para realizar la IDFT, se usará la integral:

$$g(t) = \int c_f \cdot e^{i \cdot 2\pi \cdot f \cdot t} df \quad (9)$$

Donde matemáticamente simplemente se están sumando todas las señales de tonos puros, señales sinusoidales de una frecuencia concreta (las c_f). Siendo gráficamente:

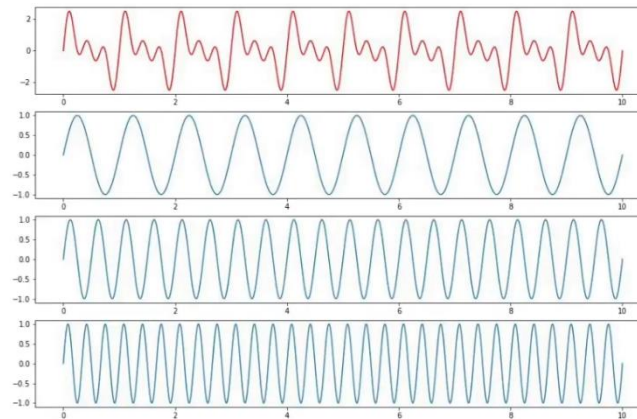


Figura 26: Forma de la suma de varias señales puras. Imagen extraída del video [58] en el minuto 40:26.

Para pasar de la FT a la DFT simplemente hay que sustituir los dominios y condóminos por los siguientes:

$$g(t) \mapsto x(n); t = nT = n/S_r^{(10)}, T: \text{periodo de muestreo.}$$

$$\hat{g}(f) = \int g(t) \cdot e^{-i \cdot 2\pi \cdot f \cdot t} dt \mapsto \hat{x}(f) = \sum_{n=0}^{N-1} x(n) \cdot e^{-i \cdot 2\pi \cdot f \cdot n} \quad (11)$$

Como queremos que la frecuencia también sea discreta y finita, aplicamos otra transformación a la función:

$$\hat{x}(f) = \sum_{n=0}^{N-1} x(n) \cdot e^{-i \cdot 2\pi \cdot f \cdot n} \mapsto \hat{x}(k/N) = \sum_{n=0}^{N-1} x(n) \cdot e^{-i \cdot 2\pi \cdot \frac{k}{N} \cdot n} \quad (12)$$

$$F(k) = \frac{k}{NT} = \frac{kS_r}{N} \quad (13)$$

Tales que $n = [0, N - 1]$ y $k = [0, M - 1]$.

Si $N = M$ se tendrán las ventajas:

- 1) De poder pasar del dominio frecuencial al temporal (IFT) y viceversa (FT) siempre que queramos, dado que tendremos el mismo número de muestras tanto en frecuencia como en tiempo.
- 2) Si ambos valores son iguales es más eficiente computacionalmente.

Sin embargo, por Nyquist ($S_r/2$) sabemos que la mitad de frecuencias serán redundantes $F(N/2) = S_r/2$. Aprovechando este conocimiento, se puede calcular la FT de forma mucho más rápida eliminando precisamente la mitad de las frecuencias, ya que estas por Nyquist sabemos que serán redundantes, una copia de las primeras. Esto es precisamente con lo que se basa la FTT, la transformada rápida de Fourier, pasando de una complejidad $O(N^2)$ a $O(N \log_2 N)$, no obstante, para poder aplicarse se necesita tener una N que sea potencia de 2.

Finalmente, debido a que la FT crea un espectro donde muestra una instantánea (*snapshot*) de toda la canción o sonido, es decir, perdemos la información de las frecuencias que se encuentran solo en un periodo de tiempo corto o segmento de la canción, con la *short time* FT (STFT) nos permite visualizar por cada unidad de tiempo las frecuencias y con qué intensidad o magnitud están sonando, con ello creamos el espectrograma.

La función de la STFT es la siguiente, donde el *windowing* está incluido.

$$S(m, k) = \sum_{n=0}^{N-1} x(n + mH) \cdot w(n) \cdot e^{-i \cdot 2\pi \frac{k}{N} n} \quad (14)$$

Tal que m es el número de *frame* y H el *hop size*. Notar que la N de S(m, k) no será ahora el total de muestras que se tienen en todo el sonido sino solo las que se tienen en un *frame*.

En el *magnitude spectrum* (espectro de magnitud) las frecuencias que nos encontramos, es la media de todas las frecuencias que nos encontramos en todo el sonido/señal analizada.

El espectro es la principal característica que se utilizará para DL y los algoritmos de ML (AI audio).

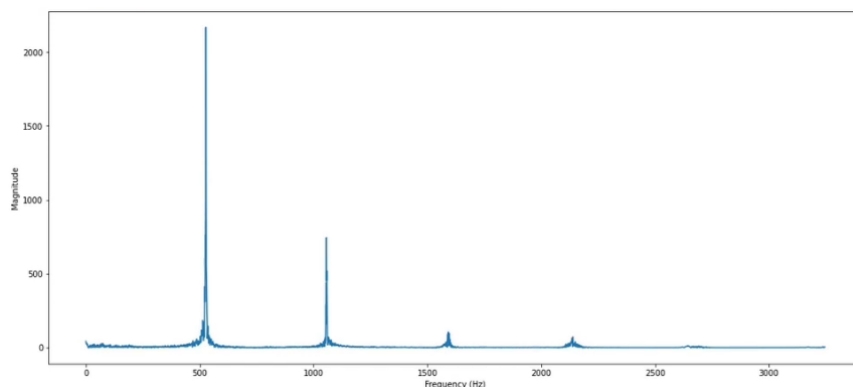


Figura 27: Ejemplo de un espectro frecuencial. Imagen extraída del video [59] en el minuto 10:25.

Espectrograma

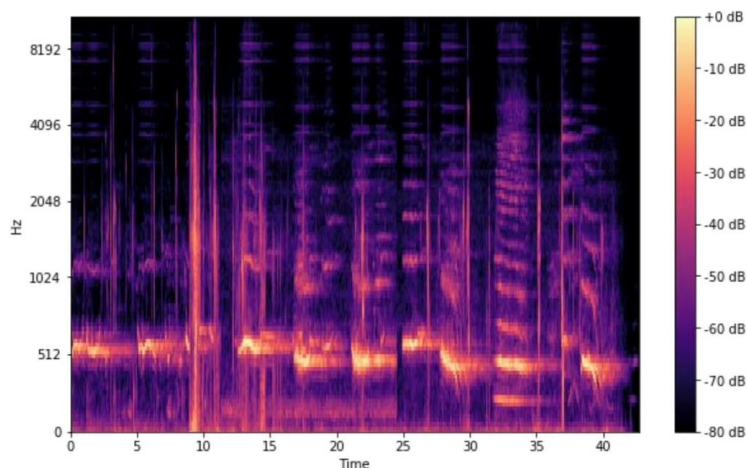


Figura 28: Ejemplo de espectrograma. Imagen extraída del video [60] en el minuto 7:30.

La cantidad de *bins* temporales y frecuenciales que se encuentran en un espectrograma, quedan definidos por las fórmulas:

$$\#frequency\ bins = framesize/2 + 1 \quad (15)$$

$$\#frames = (samples - framesize)/2 + 1 \quad (16)$$

Se divide por 2 ya que como vimos en la explicación de la DFT tenemos un espectro simétrico, donde se divide en la frecuencia de Nyquist, ya que las frecuencias superiores a $f_{Nyquist}$ no existen, son artefactos.

La IFT, la transformada inversa de Fourier, nos permite reconstruir un espectrograma a audio nuevamente, es decir, pasar del dominio frecuencial al temporal nuevamente. No obstante, debido a que muchas veces solo se tiene el espectrograma (sola la magnitud) y no la fase de la señal, para recuperar esta última, normalmente se suele

usar la técnica llamada Griffin-Lim. Básicamente, aplicando técnicas de optimización, partiendo de una inicialización aleatoria de las fases, buscará aquellas que obtengan mejor resultado en la reconstrucción de la señal. Para mejorar la eficiencia y resultado, normalmente la inicialización aleatoria se cambia por los valores de las fases de la señal original, si es que esta se tenía y el objetivo era generar otra con algunas modificaciones, como, por ejemplo, eliminar ruido.

Tanto la FT como la STFT devuelve la magnitud en escala lineal, es decir, en las unidades de la intensidad (W/m^2), es por ello que posteriormente hay que transformar dichos valores a la escala logarítmica (dB) para que los datos obtenidos sean más interpretables y fáciles de entrenar, tal y como se explica en el cuerpo principal de la memoria.

8.2. Anexo II: Explicación detallada de los Mel-espectrogramas

La función extraída empíricamente para calcular las frecuencias Mel (*Mel-frecuencias*) es: $m = 2595 \cdot \log(1 + f/500)$ ⁽¹⁷⁾ y su inversa es: $f = 700 \cdot (10^{m/2595} - 1)$ ⁽¹⁸⁾. Mel es la abreviación de melodía.

No obstante, para crear el Mel-espectrograma o las frecuencias a la escala Mel (*Mel-sacale*), no basta con transformar las frecuencias a las Mel-frecuencias aplicando la fórmula, sino que primero hay que elegir un número de bandas Mel (*Mel bands*), las nuevas frecuencias que contendrá la *Mel-sacale*, después crear los llamados *Mel filter banks* y finalmente, aplicar estos últimos al espectrograma *vanilla*.

El número de *Mel bands* a elegir dependerá del problema, aunque debido a que estos representan la percepción de como recibimos el sonido y normalmente se trabajará en problemas de *speech recognition* (percepción del habla) o separación musical, ya sea de instrumentos, sonido, entre otros, así como en la música occidental se tienen normalmente 88 notas diferentes como mucho (las de un piano), siendo solo la mitad de estas las que normalmente se suelen encontrar en la música moderna, un número de *Mel bands* típico a elegir es: 40, 60, 90 o 128 como máximo.

Una vez definido el número de *Mel bands*, para crear las Mel-frecuencias que se encontrarán en el Mel-espectrograma, primero hay que crear los *Mel Filter Banks*, para ello se tienen que seguir los siguientes pasos:

- 1) Convertir la frecuencia más baja y la más alta a Mel-frecuencias para crear un rango de Mel-frecuencias.
- 2) Dividir el rango en n partes equidistantes, siendo n el número de *Mel bands* elegido. De esta división se seleccionan n Mel-frecuencias, las que separan el rango. Estas son las llamadas Mel-frecuencias centrales, habiendo una por cada *Mel band*.

- 3) Convertir las Mel-frecuencias centrales a Hercios (frecuencias normales) nuevamente.
- 4) De las frecuencias obtenidas arredondearlas a las frecuencias más cercanas que le corresponda cada una de las originales, las establecidas por el *frame length*. Esto es debido a que seguimos estando en dominio discreto y, por ende, no podemos tener codificadas todas las frecuencias posibles.
- 5) Crear los llamados filtros triangulares. Para crear cada filtro simplemente se trazan dos rectas que conecten la Mel-frecuencias central de cada filtro, ya que hay un filtro por cada una, con la frecuencia normal que le corresponde a la Mel-frecuencia central del filtro anterior y del filtro siguiente. Los filtros se representan de forma matricial de dimensión: $(\# \text{ bands}, \text{framesize}/2 + 1)$, siendo la segunda dimensión la frecuencia de Nyquist. Gráficamente se pueden ilustrar como es mostrado en la siguiente imagen de la figura 29:

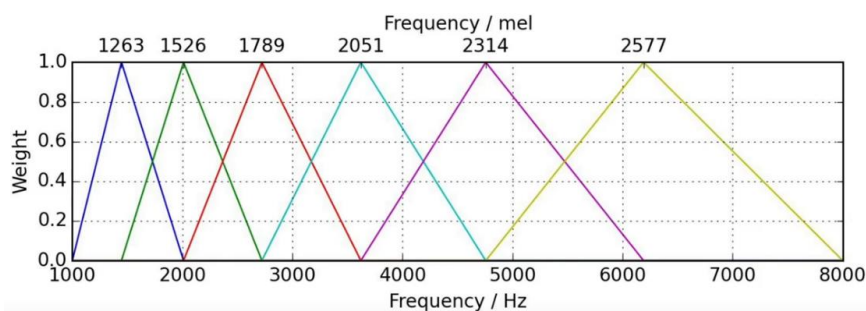


Figura 29: Filtros Mel. Imagen extraída del video [61] en el minuto 4:15.

Finalmente, para obtener el Mel-espectrograma, simplemente hay que multiplicar las matrices M y Y , siendo estas la que representa los filtros y el espectrograma *vanilla* sucesivamente.

$$M = (\# \text{bands}, \text{framesize}/2 + 1) \quad (19)$$

$$Y = (\text{framesize}/2 + 1, \# \text{frames}) \quad (20)$$

$$\text{Mel_espectrograma} = MY = (\# \text{bands}, \# \text{frames}) \quad (21)$$

En resumen, los Mel-espectrogramas convierten las frecuencias (Hz) a *Mel bands*, permitiéndonos tener información perceptiva o psicológica relevante sobre la forma en que los humanos percibimos el *pitch*.

Otra característica del audio es la llamada *Mel-Frequency Ceptral Coefficients* (MFCC). El proceso de extracción es muy similar al del Mel-espectrograma, pero con algunas diferencias, siendo la primera de ellas el cambio de nombre a las diferentes características, quedando estos tal y como se indica en la siguiente lista:

| | | |
|------------------------------------|---|--------------|
| Spectrum (espectro de frecuencias) | → | Ceptrum |
| Frequency (frecuencias) | → | Quefrequency |
| Filtering (filtrado) | → | Liftering |
| Harmonic (armónico) | → | Rhamonic |

Como se puede observar son los mismos nombres al revés.

Para calcular el *cepstrum*, se calcula con la siguiente formula:

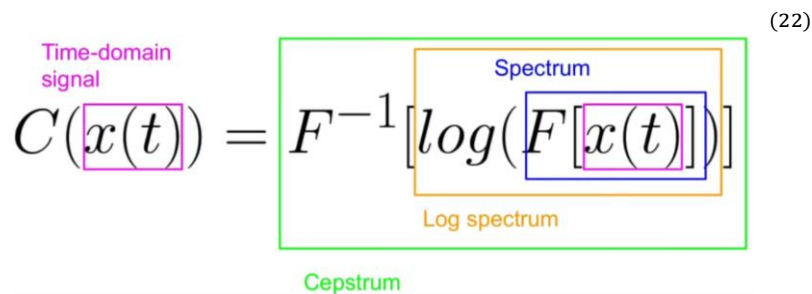
$$C(x(t)) = F^{-1}[\log(F[x(t)])] \quad (22)$$


Figura 30: Formula detallada del ceptrum. Imagen extraída del video [62] en el minuto 8:50.

La idea detrás del *cepstrum* es volver a calcular el espectro del espectro dado que la función que modela el espectro es también una función continua con estructuras periódicas, debido a que tenemos componentes armónicos. Donde cuando tenemos una señal con periodicidades, incluido si se trata de un espectro, podemos aplicar una transformación como la FT para entender los diferentes componentes y ver qué frecuencias están presentes en la señal. Para calcular el *cepstrum*, dado que el primer espectro se encuentra en el dominio frecuencial para calcular el espectro del espectro, se usará la IFT y no la FT.

8.3. Anexo III: Explicación de las propiedades básicas de la música

Frecuencia

Perceptualmente se percibe con lo que llamamos *pitch* o tono. Este puede cambiar entre persona y persona debido a, por ejemplo, su rango auditivo, pudiendo escuchar más o menos frecuencias/tonos. Físicamente corresponde a la cantidad de periodos (patrones idénticos repetidos a lo largo del tiempo) por segundo.

Timbre

Perceptualmente también le llamamos timbre. Está relacionado con los armónicos, la envolvente del sonido, la armonía o melodía, la modulación y la modulación de la amplitud. El timbre es el que nos permite diferenciar los instrumentos, incluso siendo del mismo tipo y familia.

La armonía o melodía indica como se están tocando las notas, si una por una o varias de golpe (acordes). La modulación en amplitud y frecuencial son variaciones periódicas en la amplitud y frecuencia respectivamente, musicalmente hechas con las técnicas de trémolo y vibrato.

Cuando nos referimos a los armónicos, hay que dividirlos en dos tipos, los armónicos parciales, cuando todos los tonos que lo componen son casi múltiplos enteros de la frecuencia o tono fundamental, y los inarmónicos parciales que son cuando todos o algunos de los tonos que lo forman no son múltiplos exactos de la frecuencia fundamental. Dependiendo de los instrumentos, hay algunos que generan mayoritariamente inarmónicos, como los de percusión, y otros generan más armónicos.

La envolvente temporal es quizás el componente del timbre que más lo define, se trata de la forma general de una señal, “su superficie”. Esta forma tiene cuatro partes: *attack*, *decay*, *sustain* y *reléase*. Se refieren a como empieza la señal, como está va decayendo, su periodo de estabilidad y como termina. Cada instrumento del mismo tipo de familia tiene formas características en cada una de las partes.

Cualquier variación en el entorno que se esté tocando un instrumento, el material usado para ello, la persona que lo escucha, su posición geográfica, entre otros, hará alterar uno o más de los componentes que forman el timbre, cambiando en el proceso el sonido generado y/o percibido. Otros factores como la edad de una persona o volumen de la música o su duración (en caso de ser de duración infinitesimal) también alterarán la percepción acústica.

Formalmente, se define como la diferencia entre dos sonidos de la misma intensidad, frecuencia y duración.

Intensidad

Es la propiedad sonora comúnmente conocida como volumen, o, físicamente la amplitud de la señal y está relacionada con la potencia. Su valor nos indica la cantidad de energía que tiene una señal en un momento determinado y se mide en Watts (W). Concretamente con la formula: $I = W/m^2$ ⁽²³⁾, o lo que es lo mismo, potencia por unidad de área.

La intensidad mínima que necesita un humano para poder escuchar un sonido, técnicamente conocida como TOH (*threshold of hearing*, en español, lindar del sonido), es de $10^{-12} [W/m^2]$. Así mismo, el lindar del dolor es el llamado TOP (*threshold of pain*) con un valor de $10 [W/m^2]$.

Debido a que el rango auditivo no dañino para un humano es muy amplio, una diferencia de varios ordenes de magnitud, 13 en concreto, es conveniente utilizar una escala logarítmica para expresar el valor de la intensidad de forma que sea más tratable y entendible, tanto para los humanos como para los algoritmos. La nueva unidad de medida es el decibelio (dB), el cual, representa la ratio entre dos intensidades diferentes, siendo el denominador el valor de referencia. Como valor de referente se selecciona el TOH, nuestro límite auditivo inferior.

La fórmula para calcular los decibelios a partir de la intensidad es:

$$dB(I) = 10 \log_{10}(I/I_{TOH})$$
 ⁽²⁴⁾

Cada 3dB aproximadamente la intensidad se duplica.

8.4. Anexo IV: Requisitos y ejecución del programa

Para la ejecución del programa, se tendrá que tener instaladas todas las librerías con sus versiones correspondientes nombradas en el punto 4.1) *Equipo, tecnologías y librerías usadas*.

Solo deben ser ejecutados los archivos que se encuentran en el bloque de ejecuciones, es decir, el archivo *main*, o, en caso de querer crear, modificar, evaluar o agregar nuevos datos para el *dataset*, los *main* secundarios. El bloque de configuraciones también será necesario abrirlo y modificarlo para establecer los hiperparámetros de los modelos y arquitecturas que se quieran, así como modificar la clase que se encuentra dentro de esta, para cambiar las rutas de los *datasets* en caso de que: 1) se les quiera cambiar de nombre o mover de directorio; 2) se quieran añadir de nuevos.

En el archivo *main* es donde se definen (instancian) y se entrenan los modelos, así como los archivos *main* secundarios se usan para realizar operaciones como la creación de espectrogramas, Mel-espectrogramas, separación y mezcla de audios, creación de gráficos, evaluación de los modelos, etcétera.

Dentro del módulo de funcionalidades, el llamado *mylibs*, las librerías o archivos *python* creados se han separado en tres directorios diferentes según su funcionalidad, llamándose *audio*, *ml* y *utilities*. A continuación, en la figura 32 se ilustra el esquema para resumir la información entre las relaciones de los directorios del proyecto y sus archivos.

Proyecto IA

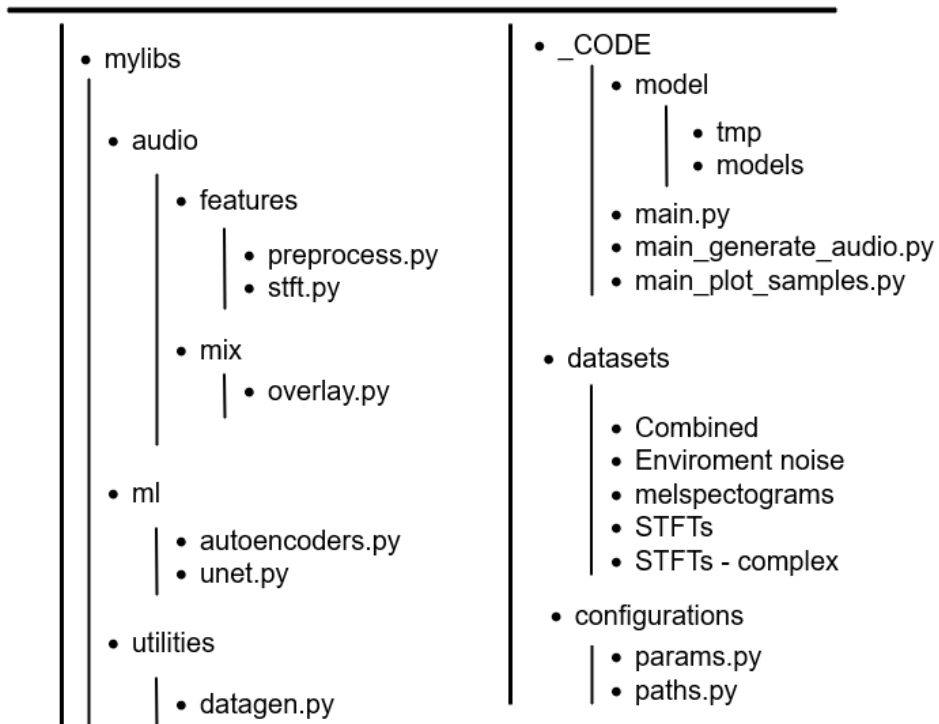


Figura 31: Organización del proyecto de programación. Elaboración propia mediante [28].