GRADO EN INGENIERÍA MATEMÁTICA E INTELIGENCIA ARTIFICIAL

TRABAJO FIN DE GRADO

AudioMind: Modelos Inteligentes para Procesamiento y Transformación de Sonido mediane IA Generativa

Autor: Mario Freire Arias

Director: David Contreras Bárcenas

Madrid

Junio de 2025

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

‘AudioMind: Modelos Inteligentes para Procesamiento y Transformación de Sonido mediane IA Generativa’

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el

curso académico 2024/25 es de mi autoría, original e inédito y

no ha sido presentado con anterioridad a otros efectos.

El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido

tomada de otros documentos está debidamente referenciada.

Fdo.: Mario Freire Arias Fecha: 11 / 06 / 2025

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: David Contreras Bárcenas Fecha: 11 / 06 / 2025

A familia, que me han apoyado en todo lo que he hecho en mi vida que merezca la pena.

AUDIOMIND: MODELOS INTELIGENTES PARA PROCESAMIENTO Y TRANSFORMACIÓN DE SONIDO MEDIANE IA GENERATIVA

Autor: Freire Arias, Mario
Director: Contreras Bárcenas, David.
Entidad Colaboradora: ICAI – Universidad Pontificia Comillas

Resumen

Este trabajo explora las tareas de reducción de ruido, la compresión de señales y la separación en fuentes dentro del campo de procesamiento de audio mediante distintas técnicas de IA. En estas tareas se han explorado dos representaciones de audio: forma de onda (tanto en el dominio del tiempo como en el de la frecuencia) y *embeddings* de audio. Estos estudios además están sujetos a la restricción de poder realizarse con una capacidad limitada de memoria de GPU (6 GB) durante el entrenamiento. Los modelos resultantes ofrecen distintas pistas acerca de los requisitos de escala en cada caso.

Palabras clave: *embeddings*, reducción de ruido, compresión de señales, separación de fuentes, CNN, Autoencoder, Transformer,

1. Introducción

Si bien no tan pronunciado como otros campos (como NLP o CV), en los últimos años el procesamiento de audio ha experimentado una revolución impulsada por distintas técnicas de IA. Aplicaciones como asistentes de voz, sistemas de videoconferencia o aplicaciones de streaming (especialmente en el caso de la música) se benefician enormemente de señales limpias, ligeras y fácilmente manipulables. Sin embargo, este tipo de señales no siempre se pueden obtener por ruidos ambientales, cuellos de botella de ancho de banda o la superposición de fuentes no deseadas con otras que sí lo son. Esto afecta a la calidad del material que se obtiene, generando un impacto negativo sobre el usuario y los distintos sistemas.

Las técnicas clásicas en estas tareas se quedan cortas. Si bien es cierto que reducen el ruido en cierta medida o que ajustan la tasa de bits para reducir el tamaño de las señales, también traen consigo consecuencias no deseadas que limitan su aplicabilidad. El desarrollo de distintas arquitecturas de IA promete soluciones a estos problemas, en ocasiones conllevando una renuncia parcial a la liviandad computacional de las técnicas tradicionales.

2. Definición del proyecto

Antes de entrar al desglose por tareas de los objetivos del trabajo, conviene matizar que este trabajo no es uno en el que la mayor carga de trabajo vaya dirigida a optimizar un objetivo muy concreto mediante prueba y error. Al haber varias tareas, necesariamente, deberemos renunciar a un mayor grado de profundidad. Además, al carecer de una base real de conocimientos en el campo (tanto a nivel teórico como práctico), los resultados alcanzados tendrán la apariencia de poco destacables. Efectivamente, lo son *en comparación* al resto: equipos de gente especializada, que además tiene más recursos al alcance de su mano (tiempo, dinero, infraestructura, conocimientos previos, experiencia, etc.).

En cuanto a la tarea de reducción de ruido, buscamos construir un modelo que cumpla su cometido minimizando los daños colaterales a la señal. Se compararán distintas arquitecturas

según la métrica ∆SNR. En esta tarea haremos uso de los embeddings generados por el modelo *wav2vec 2.0* [1].

En cuanto a la tarea de compresión, se mirará la ratio de compresión (CR) y la *Log-Spec-Distance* (LSD), métrica empleada para medir la pérdida de calidad resultante de descomprimir la representación que genera el modelo.

En la tarea de separación por fuentes, las métricas consideradas serán SI-SDR, SIR y SAR para ubicar el modelo.

3. Descripción del modelo/sistema/herramienta

En el proyecto se han empleado distintas arquitecturas para las distintas tareas. Estas arquitecturas están detalladas en mayor detalle en el cuerpo del trabajo, en el presente resumen simplemente se hace una mención de estas.

En la tarea de reducción de ruido se usan tres arquitecturas distintas en cada uno de los dos modos: TCN (*Temporal Convolutional Network*), una versión de la arquitectura U-Net [2] y por último FCLDNet (*Fully Connected Low-Dimensional Network*).

Para la tarea de compresión empleamos tres arquitecturas distintas, pero todas ellas algún tipo de *autoencoder*. En primer lugar, un Autoencoder Convolucional en su totalidad. La segunda arquitectura es un híbrido entre Autoencoder Conv-GRU (Convolucional con GRU). Por último, un Autoencoder TFC-TDF (*Time-Frequency Convolution Time-Distributed Fully-connected*). Esta última arquitectura trabaja con espectrogramas en lugar de tensores *waveform*.

Para la tarea de separación por fuentes [3] se optó por una arquitectura más potente: un Conv-Transformer (las convoluciones reducen la dimensionalidad y los *Transformer* hacen el trabajo más duro).

4. Resultados

Como en este trabajo hay muchas arquitecturas con sus respectivas variantes, se ha hecho un filtrado para incluir solo las más destacables a nivel de resultados obtenidos. Cualquiera de las mencionadas que no aparezcan a continuación no lo hacen por motivos de espacio, nada más.

Para la tarea de reducción de ruido, fijémonos en la Tabla 1. Es destacable como para una misma arquitectura en un mismo orden de magnitud del número de parámetros, la métrica de ΔSNR varía tanto.

Modelo	*Nº párams*	*Tamaño (MB)*	*∆SNR*
TCN-w	373.772	1,51	-2,43
TCN-e	209.792	0,82	3,93

U-Net-e	705.664	2,77	5,35

Tabla 1-Comparativa de arquitecturas de reducción de ruido. El apellido 'e' o 'w' se refiere al tipo de dato que toma el modelo: embedding o waveform (incluye espectrograma)

Para la tarea de compresión, veamos Tabla 2. Vemos algún resultado interesante, por ejemplo, que el modelo híbrido Conv-GRU tiene el menor LSD, a la vez que mantiene una buena ratio de compresión. El modelo convolucional no parece tener buen resultado

Modelo	*Nº párams*	*Tamaño (MB)*	*CR*	*LSD*
Conv	192.517	0,80	16,00	66,66
Conv-GRU	123.040	0,53	8,00	7,96
TFC-TDF	468.242	1,92	2,00	8,10

Tabla 2-Comparativa entre diferentes modelos de compresión

Para la tarea de separación por fuentes, veamos la Tabla 3. Podemos ver que los valores de SIR y SAR no son malos, pero el valor de SI-SDR deja mucho que desear. Esto nos indica que la tarea es desafiante para esta configuración, potencialmente solucionable con mejores técnicas o una mayor escala.

Modelo	*Nº Páram*	*Tamaño (MB)*	*SI-SDR*	*SIR*	*SAR*
Conv-Transformer	287.553	1,100	-34,05	-3,71	4,52

Tabla 3- Tabla de datos de la arquitectura para separación por fuentes.

5. Conclusiones

En conclusión, este trabajo pone de manifiesto que el uso de representaciones basadas en *embeddings* suele ofrecer un mejor rendimiento con un coste de parámetros y tamaño de modelo más contenidos (en reducción de ruido), mientras que en la compresión los autoencoders híbridos consiguen un equilibrio óptimo entre ratio y calidad (LSD) frente a modelos puramente convolucionales o espectrogramáticos. Los resultados obtenidos en separación de fuentes, evidencia que las métricas SI-SDR siguen siendo muy bajas, lo que subraya la dificultad intrínseca de esta tarea y la necesidad de escalado o de técnicas más especializadas. Todo ello, logrado ajustándose a la limitación de 6 GB de memoria de GPU, sugiere que las técnicas de IA para audio tienen un gran potencial, pero requieren cuidadosas elecciones de representación, diseño de modelos y recursos, así como futura investigación para mejorar la robustez y la eficiencia.

6. Referencias

[1] Baevski, A., Zhou, Y., Mohamed, A., & Auli, M. (2020). wav2vec 2.0: A framework for self-supervised learning of speech representations. Advances in Neural Information Processing Systems, 33, 12449–12460
[2] Ronneberger, O., Fischer, P., & Brox, T. (2015). U-Net: Convolutional networks for biomedical image segmentation. In Proceedings of the International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI 2015) (pp. 234–241). Springer, Cham. https://doi.org/10.1007/978-3-319-24574-4_28
[3] Hennequin, R., Khlif, A., Voituret, F., & Moussallam, M. (2020). Spleeter: A fast and efficient music source separation tool with pre-trained models. Journal of Open Source Software, 5(50), 2154. https://doi.org/10.21105/joss.02154

AUDIOMIND: INTELLIGENT MODELS FOR AUDIO PROCESSING AND TRANSFORMATION USING GENERATIVE

Author**:** Freire Arias, Mario
Supervisor: Contreras Bárcenas, David.
Collaborating Institution: ICAI – Pontifical Comillas University

Abstract

This work explores the tasks of noise reduction, signal compression, and source separation within the field of audio processing using various AI techniques. Two audio representations are investigated for these tasks: the waveform (in both the time and frequency domains) and audio embeddings. All studies are also constrained by the requirement to operate within a limited GPU memory capacity (6 GB) during training. The resulting models provide valuable insights into the scaling requirements for each case.

Keywords: embeddings, noise reduction, signal compression, source separation, CNN, autoencoder, transformer

7. Introduction

Although not as pronounced as in other fields (such as NLP or CV), audio processing has undergone a revolution in recent years driven by various AI techniques. Applications like voice assistants, videoconferencing systems, and especially music streaming services benefit enormously from clean, lightweight, and easily manipulable signals. However, such ideal signals are not always attainable due to environmental noise, bandwidth bottlenecks, or the overlap of unwanted sources with desired ones. These issues degrade the quality of the final output, negatively impacting both users and downstream systems.

Classical techniques for these tasks often fall short: they may reduce noise to some extent or adjust bitrates to shrink file sizes, but they also introduce unwanted artifacts that limit their applicability. The development of new AI architectures promises solutions to these problems—albeit sometimes at the expense of the computational efficiency of traditional methods.

8. Project Definition

Before breaking down the project's objectives by task, it is important to clarify that this work does not aim to optimize a single, narrowly defined objective through extensive trial and error. Because multiple tasks are addressed, we must necessarily forgo a deeper level of specialization. Moreover, lacking a solid real-world foundation in the field (both theoretical and practical), the results achieved may seem modest compared to those of well-resourced, specialized teams with greater access to time, funding, infrastructure, prior knowledge, and experience.

For the noise reduction task, the objective is to develop a model that minimizes collateral damage to the signal. The different architectures are compared using the ΔSNR metric.

For the compression task, we evaluate the compression ratio (CR) and the Log-Spec-Distance (LSD), a metric used to measure the loss in signal quality when reconstructing from

the model's representation. In this task, we will make use of the embeddings generated by the *wav2vec 2.0* model [1].

Finally, for the source separation task, the models are evaluated using the metrics SI-SDR, SIR, and SAR.

9. Description of the model/system/tool

Various architectures were used for the different tasks, described in detail in the full report. A brief overview is provided here.

For the noise reduction task, three architectures were tested in both modes (with waveform and with embeddings): TCN (Temporal Convolutional Network), a variant of U-Net [2], and FCLDNet (Fully Connected Low-Dimensional Network).

For the compression task, three autoencoder-based architectures were used: a fully convolutional autoencoder, a Conv-GRU hybrid autoencoder, and a TFC-TDF autoencoder (Time-Frequency Convolution + Time-Distributed Fully Connected), which operates over the spectrograms of the signal instead of the waveform.

For the source separation task [3], a more powerful architecture was used, a Conv-Transformer, which combines convolutional layers for downsampling with Transformer blocks for the heaviest part of the computation.

10. Results

Since this work includes many architectures with their respective variants, a selection has been made to include only the most noteworthy ones in terms of the results obtained. Any of those mentioned earlier that do not appear below are omitted for space reasons only.

For the noise reduction task, let us look at Table 1. It is noteworthy how, for the same architecture and within the same order of magnitude in the number of parameters, the ΔSNR metric varies so much.

Model	*Param. Count*	*Size (MB)*	*ΔSNR*
TCN-w	373.772	1,51	-2,43
TCN-e	209.792	0,82	3,93
U-Net-e	705.664	2,77	5,35

Table 4-Comparison of noise reduction architectures. The suffix "e" or "w" indicates the model's input type: embedding or waveform (including spectrogram)

For the compression task, see Table 2. We observe some interesting results, for example, that the hybrid Conv-GRU model has the lowest LSD while maintaining a good compression ratio. The convolutional model does not seem to perform well.

Model	*Param. Count*	*Size (MB)*	*CR*	*LSD*
Conv	192.517	0,80	16,00	66,66
Conv-GRU	123.040	0,53	8,00	7,96
TFC-TDF	468.242	1,92	2,00	8,10

Table 5-Comparison of various compression models

For the source separation task, see Table 3. We can see that the SIR and SAR values are not bad, but the SI-SDR value leaves much to be desired. This indicates that the task is challenging for this configuration, potentially solvable with better techniques or a larger scale.

Model	*Param. Count*	Tamaño (MB)	SI-SDR	SIR	SAR
Conv-Transformer	287.553	1,100	-34,05	-3,71	4,52

Table 6- Results for the source separation architecture

11. Conclusions

This study shows that, for the noise reduction task, using embeddings leads to better results with fewer parameters and smaller model sizes. For the compression task, hybrid autoencoders achieve a better balance between compression ratio and reconstruction quality. Finally, for source separation, the extremely low SI-SDR obtained shows that this task is particularly complex and would require more sophisticated architectures or larger models. All this has been achieved within the constraints of a 6 GB GPU, offering valuable lessons for the design and training of audio models under limited resources.

12. References

[1] Baevski, A., Zhou, Y., Mohamed, A., & Auli, M. (2020). wav2vec 2.0: A framework for self-supervised learning of speech representations. Advances in Neural Information Processing Systems, 33, 12449–12460

[2] Ronneberger, O., Fischer, P., & Brox, T. (2015). U-Net: Convolutional networks for biomedical image segmentation. In Proceedings of the International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI 2015) (pp. 234–241). Springer, Cham. https://doi.org/10.1007/978-3-319-24574-4_28

[3] Hennequin, R., Khlif, A., Voituret, F., & Moussallam, M. (2020). Spleeter: A fast and efficient music source separation tool with pre-trained models. Journal of Open Source Software, 5(50), 2154. https://doi.org/10.21105/joss.02154

Índice de la memoria

Índice de tablas

1. INTRODUCCIÓN

El procesamiento de audio ha cobrado un interés mayor en los últimos años, impulsado tanto por el avance de las técnicas de IA como por la demanda de aplicaciones prácticas que requieren señales de alta calidad. Mientras que campos como el procesamiento de lenguaje natural (NLP) y la visión por computador (CV) acaparan buena parte de la atención académica e industrial, la vía auditiva ha quedado, en cierta medida, en un segundo plano, aun si es fundamental para la comunicación humana, el entretenimiento y multitud de sistemas que pasan desapercibidos, pero forman parte de nuestro día a día.

Tareas como la reducción de ruido, la compresión (y necesaria descompresión) eficiente de señales o la separación de fuentes son cada vez más críticas en distintos servicios, incluyendo teleconferencia, asistentes de voz, plataformas de *streaming* o sistemas de reconocimiento automático de audio. La aparición de arquitecturas profundas y distintas formas de representar la información, junto con una capacidad de cómputo cada vez más accesible, abre la puerta a soluciones que superan con creces los resultados clásicos, aunque también plantean nuevos retos en términos de diseño, memoria y tiempo de entrenamiento.

1.1. CONTEXTO Y MOTIVACIÓN

A lo largo del grado, se nos ha guiado a los alumnos por las dos grandes temáticas en IA hoy: NLP y CV. Lo estudiado ha servido para formarnos una idea de los principales problemas y planteamientos en éstos, dejando entrever un enorme mundo teórico, experimental e industrial detrás de ambos. Sin embargo, el contenido cubierto ha quedado cojo de una pata: el audio, el cual no se ha cubierto en ningún momento (al menos de manera significativa).

Por otra parte, y tomando por inspiración algunos trabajos realizados a lo largo del grado, se plantea la posibilidad de desarrollar sistemas de audio portátiles de muy pequeño peso que puedan tener un impacto tangible.

1.2. OBJETIVOS

Así pues, el propósito principal del presente trabajo es el de servir como excusa para aprender sobre este campo: tareas, retos, arquitecturas frecuentemente empleadas, frentes abiertos, conjuntos de datos, métricas comúnmente utilizadas, etc. Éstos son solo algunos de los temas de los que se pretendía aprender en el momento de escoger esta temática.

Dicho esto, todo buen aprendizaje necesita ser práctico. Por este motivo se escogieron una serie de tareas suficientemente asequibles para un principiante, pero con suficiente impacto como para poder justificarse.

Pasemos ahora a definir ahora objetivos claros y mesurables. En primer lugar, se han de emplear únicamente los recursos disponibles por el alumno: un portátil con una tarjeta gráfica de 6 GB de capacidad. Con esto, se realizarán experimentos en tareas de reducción de ruido, compresión de señales y separación por fuentes.

En los tres casos de estudio se busca estudiar el impacto de diferentes representaciones de los datos (dominios clásicos –tiempo y frecuencia- y embeddings) y diseñar e implementar diferentes arquitecturas para intentar resolver cada caso. Todos estos requisitos de diseño tienen como fin poder extraer aprendizajes concretos y conclusiones prácticas acerca de estas cuestiones.

1.3. PLANIFICACIÓN Y VIABILIDAD ECONÓMICA

En cuanto a los costes del trabajo, el único coste de este trabajo es el de la electricidad consumida y el tiempo invertido. En términos económicos, dado que se emplean conjuntos de datos públicos y una plataforma computacional personal ya poseída previamente, estos costes no se tendrán en cuenta.

En cuanto a la planificación en el calendario del trabajo, se ha seguido una muy similar a la propuesta

en el Anexo B. Frente a la hoja de ruta inicial del proyecto, el resultado final experimentó dos grandes cambios para hacerlo más coherente, viable y productivo. En primer lugar, se sustituyó la tarea de *style transfer* del trabajo en pro de un estudio de las tareas empleando *embeddings* de audio generados mediante el modelo *Wav2vec* de Facebook [1]. El otro gran cambio que experimentó el proyecto respecto de la planificación inicial del mismo fue la renuncia a realizar la tarea opcional de compleción de melodías, que no se pudo desarrollar debido a una falta de tiempo.

1.4. ESTRUCTURA DEL TRABAJO

El trabajo consta de tres partes principales según las tareas que se busquen realizar. Al ser tres (reducción de ruido, compresión de audio y separación por fuentes), existe una división en tres grandes bloques, que vertebra todo el trabajo realizado.

2. ESTADO DEL ARTE

Dado que el objetivo de este proyecto no es un recorrido exhaustivo por el estado del arte (SOTA) a lo largo del tiempo, vamos a mencionar los modelos, resultados o trabajos más destacados de cada tarea. Esto incluye tanto resultados históricamente relevantes por introducir novedades que cambien el paradigma del momento como el estado del arte actual.

2.1. REDUCCIÓN DE RUIDO

El primer resultado relevante de esta tarea es *RNNoise* [2]. Este trabajo introduce por primera vez un sistema neural de reducción de ruido a la literatura. Es un sistema basado en la arquitectura GRU [3] y de pequeño tamaño (unos 100.000 parámetros) que puede ejecutarse en tiempo real en una CPU con bajo uso a la vez que mejora los resultados clásicos. Tal fue su impacto que plataformas como Zoom o Discord lo incorporaron al tiempo de su publicación.

En cuanto al SOTA de esta tarea, se trata de *ZipEnhancer* [4], un modelo de 11.3M parámetros.

2.2. COMPRESIÓN

El primer resultado históricamente relevante es *SoundStream* [5]. Esto es porque es el primer método completamente *end-to-end* de una única etapa neural capaz de funcionar en tiempo real en la CPU de un teléfono. Para ello emplea técnicas que este trabajo no aspira a cubrir, como *Residual Vector Quantization* (RVQ) [6] o funciones de pérdida adversariales. Este trabajo supuso una superación del paradigma clásico por uno completamente neural.

El SOTA de esta tarea es *EnCodec* [7], un *autoencoder* convolucional que además combina RVQ, pérdida adversarial multiescala y otros factores. Es capaz de dar una calidad alta (95/100) en MUSHRA con decodificaciones en tiempo real en una CPU.

2.3. SEPARACIÓN POR FUENTES

El primer resultado a mencionar es *Spleeter* [8]. El mayor motivo que explica el impacto de este trabajo es que dio pie a una librería de Python que separaba cualquier canción en sus componentes en unos segundos sin necesidad de hardware especializado. A nivel técnico, emplea una arquitectura U-Net [9] que opera sobre los espectrogramas de las canciones.

El SOTA en esta tarea es *MDX 2023 Ensemble* [10]. Al ser un ensamblaje de varios modelos, merece mención el SOTA de un único modelo, que pertenece al *fine-tuned Hybrid Transformer Demucs v4* [11]. Combina bloques transformer con una hibridación de U-Net en el dominio temporal y espectral.

2.4. RELACIÓN CON EL TRABAJO DESARROLLADO

Evidentemente, acercarse siquiera remotamente a cualquier resultado SOTA queda fuera del alcance de este trabajo. Esto se debe a la diferencia existente entre dichos trabajos y el presente: tiempo de desarrollo y experimentación, recursos computacionales y económicos, conocimientos

previos y experiencia en el sector o el número de personas que participan en el proyecto son solo algunas de las principales diferencias que explican los diferentes objetivos y resultados entre competiciones internacionales de equipos con respaldo de diversas instituciones y el trabajo de fin de grado de un alumno sin conocimiento previo de la materia. Baste recordar que el objetivo de este trabajo no es mejorar ningún resultado, sino aprender sobre un tema antes desconocido.

3. METODOLOGÍA

Dado que el proyecto está formado por tres tareas distintas, se tratará cada una por separado en su respectiva sección.

3.1. REDUCCIÓN DE RUIDO

3.1.1. DESCRIPCIÓN TÉCNICA

En esta tarea trabajamos en el espacio X de las muestras de audio, x. Buscamos desarrollar un modelo $f: X \to X$ que, dada una muestra ensuciada con ruido $\tilde{x}$, devuelva la muestra libre de dicho ruido, x. Formalmente, buscamos que $f(\tilde{x}) = x, \forall\, x \in X$.

Para ello, en primer lugar, debemos definir el operador ■̃, que consistirá en añadir ruido blanco a la muestra con un cierto nivel de magnitud (en relación con la intensidad de la señal). A este nivel de ruido se le llama *Signal to Noise Ratio* (*SNR*) y se mide en decibelios. En nuestro caso, se probaron valores de SNR entre 5 y 20.

Por otra parte, la función de pérdida empleada es una combinación lineal regularizada de otras funciones de pérdidas más fundamentales, que buscan optimizar diferentes aspectos. Formalmente, nuestra función objetivo es

$$\begin{aligned} Loss(x, f(\tilde{x})) = \; & \alpha \cdot MSE\big(x, f(\tilde{x})\big) + \beta \\ & \cdot Loss_{Spectral}\big(x, \quad f(\tilde{x})\big) + \gamma \\ & \cdot Loss_{SI-SNR}(x, \quad f(\tilde{x})) \end{aligned}$$

Veamos ahora cada término de forma dedicada. En esta ecuación, $\alpha, \beta, \gamma \in \mathbb{R}^+$ y MSE es la función de pérdida $L2$. Si ahora definimos el operador STFT como la *Short-Time Fourier Transform* [12], podemos definir que $Loss_{Spectral}(x, f(\tilde{x})) = L1(\log(1 + STFT(x)), \log(1 + STFT(f(\tilde{x}))))$. En otras palabras, optimizamos que las frecuencias originales y reconstruidas coincidan en magnitud. Por último, definimos la función $Loss_{SI-SNR}\big(x, f(\tilde{x})\big) = -10 \cdot \log\left(\frac{\|proj\|^2}{\|noise\|^2}\right)$, también llamada *Scale-Invariante SNR loss*. Como su nombre indica, esta función de pérdida es robusta ante cambios de escala y además optimiza la intensidad relativa entre señal correctamente reconstruida frente al ruido que permanece todavía en la muestra.

3.1.2. DISEÑO DEL MODELO

En esta tarea se probaron tres arquitecturas diferentes, las cuales se comentan a continuación.

La primera y la más básica es la arquitectura TCN (*Temporal Convolutional Network*) [13]. Esta red se basa en una arquitectura que compone convoluciones 1D en la dimensión temporal del tensor de audio introducido. La *dilation* de las capas aumenta de forma exponencial con la profundidad, para captar relaciones de cada vez más largo plazo. No se usa *pooling* de ningún tipo dado que el espacio de llegada es el mismo que el de salida, y las no-linealidad que componen la red son ReLUs [14]. De esta forma, el número de parámetros de la red es $O(num_{layers} \cdot hidden_{dimension}^2 \cdot kernel_{size})$. Un último apunte respecto de esta arquitectura es que, si bien en la literatura se pueden encontrar implementaciones causales de esta arquitectura, en este trabajo se ha optado por lo contrario, ya que en ese caso la capacidad de aprendizaje de la red estaría más limitada de lo que ya lo está al eliminar una de las dos direcciones de dependencias inter temporales de los datos.

La segunda arquitectura estudiada es una U-Net 1D ligera [15]. Como tal, esta arquitectura presenta un cuello de botella en el medio de dos partes

simétricas, que se conectan entre sí por conexiones *skip*, las cuales buscan modelar la (en gran parte) invarianza de la salida respecto de la entrada. Tras cada convolución (tanto en la fase de compresión como de expansión) se aplican ReLUs como no-linealidades seguidas de un *Average Pooling 1D*.

La última arquitectura estudiada en esta sección es la FCLDNet (*Fully-Connected Low-Dimensional Network*) [16]. Esta red está compuesta de capas convolucionales con $kernel_{size} = 1$ seguidas de ReLUs, lo que equivale a MLP por instante.

3.1.3. ENTRENAMIENTO Y VALIDACIÓN

Para los entrenamientos de las redes de esta tarea se emplearon las herramientas descritas a continuación. Los tamaños de *batch* se determinan en el momento de ejecución según el tamaño del modelo para maximizar el uso de la GPU (generalmente entre 8 y 16). El optimizador empleado fue *Adam* [17] en todos los casos, con el añadido de usar un escalador para hacer un uso más eficiente de los recursos disponibles. Además, para el entrenamiento y validación se toman ventanas de 5 segundos (frecuencia de muestreo a 22 kHz) con un 25% de solapamiento entre sí.

La proporción en la que se divide el conjunto de datos disponibles es en 70%, 15% y 15% para entrenamiento, validación y test, respectivamente.

Por último, y dado el reducido tamaño de los modelos frente al tamaño del conjunto de datos, se ha descartado usar cualquier tipo de regularización como *dropout*, podado u otras de las técnicas estudiadas.

3.2. COMPRESIÓN

3.2.1. DESCRIPCIÓN TÉCNICA

En esta tarea trabajamos con dos espacios distintos pero relacionados. En primer lugar, el espacio X integrado por las muestras de audio x, y en segundo el espacio Z compuesto por las representaciones z de estos. Con esta construcción, podemos definir nuestro modelo como el par $(f, g) \mid f: X \to Z, g: Z \to X$. Para acabar el planteamiento teórico del problema, establecemos que $\dim(Z) \ll \dim(x)$.

Este planteamiento teórico nos permite intuir los puntos clave de esta tarea: buscamos construir un modelo que, dada una muestra de audio, sea capaz de construir una representación abstracta de esta (a la que llamaremos 'latente') que ocupe un espacio menor que la original y, además, ser capaz de reconstruir la muestra original a partir de la representación latente.

Así pues, démonos cuenta de que podemos componer la función g con la función f, para obtener una reconstrucción de la muestra original. Dicha reconstrucción nos gustaría que fuese lo más cercana posible a la original. Llamaremos a nuestra reconstrucción $\tilde{x} = (g \circ f)(x) = g(f(x))$, lo cual nos será útil para el entrenamiento.

Por tanto, nuestra función de pérdida será la siguiente: $Loss(x, \tilde{x}) = MSE(x, \tilde{x}) + Loss_{Spectral}(x, \tilde{x})$. Aquí, definimos $Spectral_{Loss}(x, \tilde{x}) = MSE(|STFT(x)|, |STFT(\tilde{x})|)$ es decir, la función de pérdida MSE aplicada al valor absoluto de las STFT de la muestra original y la reconstrucción.

Por último, démonos cuenta del matiz que introduce el empleo de la función $Loss_{Spectral}$. A diferencia de la función MSE, que opera punto a punto sobre cada instante de tiempo, el segundo término de error no lo hace, lo cual se debe al empleo de la STFT. Este operador toma elementos del espacio en el dominio del tiempo y los transforma al espacio en dominio de frecuencias dada una ventana que se desplaza por el eje temporal. Así pues, un pequeño desajuste en un único instante de la reconstrucción aparecerá en varias ocasiones y frecuencias, lo que amplificará su señal de error.

3.2.2. DISEÑO DEL MODELO

Teniendo en cuenta la definición anterior del problema, tiene sentido asumir que las arquitecturas

estudiadas serán de una forma u otras variantes de la arquitectura *Autoencoder* [18].

La primera variante es el Autoencoder Convolucional [19]. Esta arquitectura está compuesta por bloques de capas convolucionales con ReLUs como activación con capas de *Average Pool* cada 2 bloques. En la primera parte del modelo (llamada *encoder*) se reduce la dimensionalidad condensando la información, hasta llegar a la representación latente, a la que le aplicamos un *Global Pool*. Con esta variable intermedia, procedemos a repetir el proceso, pero a la inversa (esta parte la realiza el *decoder*).

La segunda de las arquitecturas empleadas es un Autoencoder Híbrido GRU-Convolucional. Esta arquitectura es bastante similar a la anterior, con la salvedad de que una vez concluida la fase del *encoder*, aplicamos un bloque GRU, seguido de una convolucional y una PReLU [20]. Por último, aplicamos la fase de *decoder*. Al incluir un bloque GRU (recordemos que es una simplificación de la LSTM), permitimos una mayor interacción entre las componentes de la representación de los datos que ha hecho el *encoder*, permitiendo así capturar más información. El *decoder* es convolucional.

La última arquitectura es el *TFC-TDF Autoencoder* (*Time-Frequency Convolution Time-Distributed Fully Autoencoder*) [21], el cual emplea los espectrogramas resultantes de aplicar la STFT a los datos de entrada. Esto implica que las convoluciones empleadas son de dos dimensiones en lugar de una única. Con esta salvedad, la arquitectura es mayormente igual. Se definen dos bloques de operaciones (la parte TFC, que son convoluciones 3x3, normalizaciones [22] y ReLUs, y la TDF, que tienen convoluciones 1x1 y no tienen normalizaciones, pero sí una sigmoide al final) que se alternan en cada etapa de forma simétrica.

3.2.3. ENTRENAMIENTO Y VALIDACIÓN

El esquema de entrenamiento de esta tarea es análogo al entrenamiento para la anterior. Las mayores diferencias presentes son el cambio en la función a optimizar y que, en este caso, a la entrada no se le añade ningún tipo de ruido. En el caso del Autoencoder TFC-TDF, se aplica la STFT para obtener los datos en el dominio adecuado y se toman las pistas de audio completas (en lugar de hacerlo por fragmentos de 5s) para, de ellas, tomar el espectrograma. De lo contrario, esto tiene un impacto tal en el entrenamiento que impediría mejora alguna del modelo, al no ser capaz de captar dependencias suficientes como para mejorar en la tarea.

3.3. SEPARACIÓN POR FUENTES

3.3.1. DESCRIPCIÓN TÉCNICA

Esta tarea puede ser planteada fácilmente si hacemos algunas simplificaciones frente al caso general. La principal es que todas las pistase descompondrán en un mismo conjunto de fuentes predefinidas, en lugar de personalizarse para cada entrada. Además, para hacer el problema más manejable, buscaremos únicamente aislar una de las fuentes de la pista, en lugar de buscar la descomposición completa.

En nuestro caso, vamos a suponer que nuestras pistas del espacio X tienen la forma $x = (x_0, x_1, \dots, x_n)$ para un cierto n. Estas componentes son las pistas asociadas a cada fuente y n es simplemente el número de fuentes que intervienen. Si trazamos la analogía con el tratamiento de imágenes, esta tarea equivaldría a recuperar un canal de color (por ejemplo, azul) a partir de una imagen en blanco y negro.

Para comenzar con el planteamiento teórico, sabemos que las distintas fuentes están agregadas en la pista final, $f(x_0, x_1, \dots, x_n) = x \,/\, f: X_0 \times X_1 \times \dots \times X_n \to X$

Con todo esto, ya estamos en condiciones de plantear el problema. Nuestro modelo $g: X \to X_i \,/\, i \in [0, n] \cap \mathbb{N}$ realizará esta tarea de aislamiento de una de las fuentes. Formalmente, el modelo nos devolverá una estimación de la pista que

buscamos aislar a partir de la pista en conjunto, $g(x) = \tilde{x_i}$

En cuanto a la función de pérdida empleada, es la siguiente: $Loss(x_i, \tilde{x_i}) = \alpha \cdot L1(x_i, \tilde{x_i}) + (1-\alpha) \cdot L1(|STFT(x_i)|, |STFT(\tilde{x_i})|)$. Esta función de pérdida fue escogida principalmente por ser ligera a la hora de calcularse y no ser demasiado compleja de comprender.

3.3.2. DISEÑO DEL MODELO

En esta tarea se incorpora un nuevo tipo de modelo más potente (pero también pesado). La arquitectura en cuestión [23] se basa en la conocida arquitectura *transformer* [24], hibridada con varias convolucionales. Nótese que en esta arquitectura el número de parámetros escala como $O(n^2)$ debido a las capas de atención [25] que incluye el *transformer*.

En primer lugar, las capas convolucionales condensan la información de una localidad relativamente grande a fin de reducir la dimensionalidad. Seguidamente, se realiza una convolución *depth-wise* y otra *point-wise*, las cuales extraen características locales a un coste reducido. A continuación, entramos en la etapa *transformer*, que modela las dependencias locales mediante el mecanismo de atención. Esta capa requiere un *downsampling* agresivo, ya que de lo contrario el tamaño del modelo es excesivo. Por último, recuperamos la dimensión original con una capa convolucional de *upsampling*.

En cuanto a estudiar diseños alternativos, se ha optado por lo contrario debido a varios motivos. En primer lugar, en las demás secciones del trabajo se han incluido ya suficientes arquitecturas basadas en convolucionales. Por otro lado, podemos entender esta tarea como un caso particular de reducción de ruido en el que la señal a eliminar es, en lugar de ruido blanco, todas las demás pistas de la entrada que no nos interesan. Así, y dado que esta tarea se realizó en parte tras el desarrollo de la primera, se ha considerado que tales modelos no serían suficientemente potentes dadas las restricciones de computación activas, ya que, como se verá en la sección de resultados, estas presentan alguna carencia.

3.3.3. ENTRENAMIENTO Y VALIDACIÓN

Para esta tarea, el entrenamiento se ha realizado con fragmentos de 5 segundos de las distintas pistas. El principal motivo de esto es que se busca maximizar el tamaño del modelo para permitir que el mecanismo de atención funcione de la mejor forma posible. De lo contrario, las dependencias máximas que se pueden alcanzar a capturar son de una duración excesivamente baja.

La componente de las pistas que se ha buscado aislar es la vocal, al permitir una clara evaluación subjetiva de la calidad del resultado (las pérdidas de información se hacen evidentes de forma muy sencilla al tener una estructura compleja).

4. EXPERIMENTOS

4.1. OBJETIVOS

Más allá de los objetivos generales de aprendizaje sobre el campo, se puede resumir en líneas generales los objetivos que se tenían al momento de comenzar el trabajo. Principalmente, se ha buscado ser capaz de entrenar modelos para cada tarea con los recursos detallados en la sección 4.3 y que estos modelos pudieran tener alguna utilidad real. Más concretamente, que el modelo haga lo que se esperaría de un modelo aceptable, aun si hay modelos disponibles en abierto con mejores cifras de rendimiento debidas a técnicas de entrenamiento y arquitecturas más avanzadas.

Una vez se introdujo el cambio respecto de la planificación original del trabajo, se añadieron algunos objetivos descritos a continuación. Al introducir en los contenidos del trabajo el uso de *embeddings* de audio, ahora estos se incluyen en los modelos a desarrollar en cada sección.

Estos *embeddings* no se emplearon en la tarea de compresión al ser esto un sinsentido. La propia idea

de '*embedding*' es una abstracción que nos permite construir representaciones más compactas del audio a partir del cual es generado y, como tal, sería redundante llevarlo a cabo (podríamos simplemente escoger un conjunto de *embeddings* de menor dimensión).

Algo distinto ocurre con la tarea de separación de fuentes. En la arquitectura propuesta se introduce una convolución 1D en para hacer una reducción de la dimensionalidad agresiva. Esta operación no tiene sentido para el caso de uso de los *embeddings*, ya que estos son bidimensionales: una dimensión temporal y en la otra la 'firma' de cada instante. Si bien es cierto que no es difícil plantear alternativas que permitan algo similar a lo que hace esta operación para el caso bidimensional, la manera de hacerlo no está para nada claro (podríamos definir distintas operaciones que agreguen la información de ambas). Por otra parte, y mirándolo desde el lado del 'significado', si quisiéramos emplear *embeddings* en este caso, sería mucho más fácil emplear un *transformer* de visión sobre el espectrograma. Esta alternativa fue descartada en su momento, al considerar que se escapaba del objetivo del proyecto.

4.2. CONJUNTO DE DATOS

Los conjuntos de datos que se han empleado son dos diferentes, según a las restricciones técnicas de las tareas acometidas. Por un lado, para las tareas de reducción de ruido y de compresión de audio, se ha optado por emplear el conjunto de datos público llamado GTZAN [26]. Para la tarea de separación por fuentes se ha optado por emplear el conjunto de datos DSD100 [27]. Esta decisión se debe a que para este entrenamiento se necesitan las distintas pistas fuente que, agregadas en una única pista, resultan en la (en este caso) canción que se busca desglosar por fuentes.

En cuanto al conjunto GTZAN, este está compuesto por 10 géneros de música diferentes. Cada género de música incluye 100 archivos WAV, cada uno de unos 30 segundos de duración y muestreados a 22kHz. Este conjunto de datos, además, incluye los mel-espectrogramas de las distintas pistas. También incluye dos archivos con distintas estadísticas de las pistas de completas y de fragmentos de 3 segundos de duración.

Sobre el conjunto DSD100, la principal diferencia (aparte de la estructura en la que están organizados los datos) es que este conjunto contiene tanto las pistas componentes como la mezcla. El conjunto tiene dos partes principales: *dev* y *test*, con 100 y 50 pistas respectivamente. Cada una de estas pistas tiene dos tipos de archivos asociados: por un lado, el archivo 'mixture.wav' y por otra las componentes de esta mezcla. Dichas componentes son 4 (en otros conjuntos de datos puede ser distinto): *bass*, *drums*, *vocals* y *others*. Estas pistas (de igual longitud que su respectivo *mixture*) contienen los bajos, batería, voces y otros elementos que intervienen en la canción, respectivamente.

En relación con los tamaños de cada uno de los conjuntos de datos, la versión escogida de GTZAN pesa 1.4 GB, mientras que DSD100 pesa 14.9 GB. Existen otras versiones de tamaños mayores y menores, algunas ocultas tras barreras de registros en organizaciones o de libre acceso.

4.3. CONFIGURACIÓN

Sobre la plataforma de hardware que se usa, la más importante es la tarjeta gráfica empleada: se trata de una NVIDIA GeForce RTX 4050 con 6GB de memoria dedicada. La memoria RAM compartida es indiferente ya que no se emplea para entrenamiento o validación en ningún momento.

En cuanto al soporte de software, revisémoslo de más general a más particular. Se usa WSL para facilitar el trabajo y gestión de librerías, junto con un entorno virtual para controlar las versiones de forma precisa. El código se ha desarrollado con Python 3.11 en su totalidad y para los cálculos pesados se ha empleado PyTorch 2.6.0.

Cada tarea tiene su módulo dedicado, pero todos siguen una misma estructura de archivos y comparten las interfaces a nivel general (el motivo se expone más adelante). Cada tarea tiene cuatro

archivos de código diferentes: el archivo de datos, el archivo de modelos, el archivo de función de pérdida y el archivo principal. Los tres primeros implementan la lógica que sus respectivos nombres indican: el de datos gestiona la carga y preparación de los datos, el de modelos se encarga de definir las arquitecturas y el de función de pérdida define la función de pérdida empleada en cada caso. Por otra parte, el archivo principal de cada tarea gestiona la lógica de entrenamiento y validación, con los detalles propios de cada caso.

Además, existen dos archivos de propósito general. El primero descarga y prepara los conjuntos de datos empleados en las tareas. El segundo crea la instancia con la lógica de los estudios de *optuna* realizados para cada tarea [28], [29]. Aprovecha que los archivos principales comparten una misma interfaz para las principales funcionalidades, de tal forma que permite lanzar mediante un comando varias ejecuciones de las tareas con más o menos detalles sobre las mismas: podemos especificar los parámetros que queramos de forma explícita (por ejemplo, el número de capas que queremos que tenga el modelo) o dejar que *optuna* se encargue de buscar el óptimo entre varios candidatos. De esta forma, los detalles concretos de bajo nivel quedan abstraídos y son gestionados de forma automática desde un mismo punto para todas las tareas.

Respecto de los modelos empleados para extraer los *embeddings* de los fragmentos de audio, se realizaron pruebas con varios. Finalmente, y como ya se ha mencionado, el método escogido para extraer los *embeddings* es el *wav2vec 2.0* de Facebook.

4.4. ANÁLISIS DE RENDIMIENTO

En la sección asociada se aportarán algunas métricas habituales para cada tarea. Para la tarea de reducción de ruido estas serán ΔSNR (*Delta Signal to Noise Ratio*. En el caso de la tarea de compresión se empleará la ratio de compresión (CR) y LSD (*Log-Spectral Dististance*) [30]. Para la tarea de separación por fuentes se emplearán SI-SDR (*Scale-Invariant Signal to Distortion*), SIR (*Signal to Interference*) y SAR (*Signal to Artefact*) [31].

Además, y en general, se aportarán el número de parámetros del modelo, tamaño en memoria durante inferencia, y otras métricas que pudieran ser ilustrativas.

5. RESULTADOS

Al servir como primera toma de contacto con el tema tratado, este trabajo no listará de forma exhaustiva todos los experimentos realizados, sino solo los más destacados de cada sección. Esto incluye los resultados asociados y su posterior valoración o comentario.

5.1. REDUCCIÓN DE RUIDO

Podemos ver los resultados de los experimentos en la Tabla 7.

Fijémonos en la primera conclusión que podemos extraer de la tabla: parece claro que la tarea se completa mejor en el caso de los modelos que emplean embeddings que en el caso base para tamaños de modelos comparables. Una posible explicación es que, al estar más condensada la información, es más sencillo aprender los patrones y estructuras que en un caso en el que la información está dispersa en miles de muestras por segundo.

Pasando a los resultados más concretos, cabe destacar lo sorprendentemente polarizada que está la arquitectura U-Net según el tipo de entrada que acepte: en un caso es el peor modelo en relación con el tamaño que tiene, mientras que en el otro es el mejor de todos. Esto podría ser explicado por una mayor adecuación de la representación de la información frente a la otra, dado que los tamaños son comparables.

Otro aspecto reseñable de los resultados es el buen desempeño comparativo del modelo convolucional en el modo ‘w’ frente a las otras dos arquitecturas: un ΔSNR un orden de magnitud superior a los otros dos casos. Sin embargo, esta ventaja se desvanece cuando se pasa al modo ‘e’ de los modelos.

Por último, nótese que solo en el segundo modo de operación de los modelos ('e') se logra una mejoría de la señal. En el caso alternativo, el mejor caso es una degradación no prohibitiva.

Modelo	*Nº Páram*	*Tamaño (MB)*	*ΔSNR*
TCN-w	373.772	1,51	-2,43
U-Net-w	606.721	2,40	-13.54
FCLD Net-w	335.372	1,36	-14.98
TCN-e	209.792	0,82	3,93
U-Net-e	705.664	2,77	5,35
FCLD Net-e	328.832	1,30	4,31

Tabla 7- Comparativa exhaustiva de arquitecturas de reducción de ruido. El apellido 'e' o 'w' se refiere al tipo de dato que toma el modelo: embedding o waveform (incluye espectrograma)

Un apunte que merece mención acerca de la elaboración de los embeddings es el modelo empleado. En un primer momento, se pretendía usar el modelo openl3. Sin embargo, y por problemas del entorno de ejecución, resulto imposible conseguir que tal modelo efectivamente funcionase. En ambos casos (el presente y el alternativo), los embeddings son las representaciones en un espacio latente de los fragmentos codificados por el modelo en cuestión. Existen formas alternativas de extraer representaciones en forma de embedding. La más destacada es la basada en recetarios ('*cookbooks*'). Estas representaciones se construyen proyectando el tensor de audio sobre un número arbitrario de distintas direcciones del espacio consideradas representativas. Estas direcciones no tienen por qué corresponderse con algo fácilmente comprensible, sino que podría ser una firma auditiva arbitraria, y la magnitud en su respectiva dirección 'cuánto de este sonido en concreto existe en la muestra'. De esta forma, y tomando un número grande de direcciones muestreadas (en cualquier caso, inferior al tamaño de la muestra original) y guardando las coordenadas de la muestra respecto de esta, podemos transmitir dichas coordenadas y reconstruir la muestra original con un grado de precisión arbitrariamente pequeño (dictado por el número de dichas direcciones, también dependiente de la orientación espacial relativas entre los ejes de coordenadas tomados).

5.2. COMPRESIÓN

Sobre la tarea de compresión, los resultados se encuentran reflejados en la Tabla 8. Estos resultados merecen varios comentarios. En primer lugar, como no es de extrañar, la compresión demasiado ambiciosa del modelo puramente convolucional nos entrega unos resultados pobres, con excesiva pérdida de información (alta LSD), aun si el modelo tiene un buen número de parámetros. La otra parte (y más interesante) de los resultados nos la dan las restantes arquitecturas. Por una parte, vemos que el modelo más ligero es el que obtiene la mínima pérdida de calidad, preservando un CR de 8. Esto es especialmente llamativo, teniendo en cuenta que la tercera arquitectura es mucho más pesada y tiene un CR de 2. A priori esto nos haría pensar que la pérdida de calidad sería incomparablemente inferior que el modelo Conv-GRU, pero no es así. Esto puede significar una limitación intrínseca de la arquitectura. Una interpretación alternativa es que la información ya está bastante comprometida en el caso de la arquitectura TFC-TDF debido a la STFT que se aplica para obtener la entrada al modelo.

Modelo	*Nº Páram*	*Tamaño (MB)*	*CR*	*LSD*

Conv.	192.517	0,80	16,00	66,66
Conv.-GRU	123.040	0,53	8,00	7,96
TFC-TDF	468.242	1,92	2,00	8,10

Tabla 8- Comparativa de arquitecturas de compresión

Como último comentario en esta tarea, cabe destacar un pequeño detalle a la hora de entrenar el modelo TFC-TDF. En los entrenamientos de las otras dos arquitecturas, los datos eran segmentos de 5s del conjunto de datos. Sin embargo, como TFC-TDF usa espectrogramas, esto mismo no se puede hacer en este caso. Esto es porque se aplica una STFT, y si tomamos segmentos de 5s no se le permite al modelo capturar las dependencias a largo plazo suficientes en ese modo de representación como para aprender. Tanto es así que, para ensayos realizados en un primer momento bajo esta configuración, la función de pérdida no variaba de forma sostenida ni siquiera en el conjunto de entrenamiento.

5.3. SEPARACIÓN POR FUENTES

Los resultados para la tarea de separación por fuentes están reflejados en la Tabla 9. Los resultados de esta tarea son decepcionantes, incluso teniendo unos objetivos limitados. El SI-SDR indica que la señal pierde mucha calidad al construir la señal deseada, mientras que el valor de SIR nos indica que incluso empeora en lo que a suprimir interferencias se refiere. Por último, el valor de SAR nos indica que los artefactos generados no son excesivamente graves (en comparación a las demás métricas).

Modelo	*Nº Páram*	*Tamaño (MB)*	*SI-SDR*	*SIR*	*SAR*
Conv-Transformer	287.553	1,100	-34,05	-3,71	4,52

Tabla 9- Resultados de la tarea de separación por fuentes

Vayamos ahora punto por punto para intentar arrojar algo de luz acerca del porqué de estos resultados. En primer lugar, fijémonos en el SI-SDR. Esta métrica mide (en dB) la calidad global de la estimación frente a la señal de referencia. Esto significa que son preferibles valores positivos y de gran tamaño en lugar del valor obtenido. Por otra parte, la métrica SIR mide la energía de otras fuentes que está presente en la señal estimada. De nuevo, son preferibles valores positivos y de gran tamaño. Por último, fijémonos en el valor de SAR, el cual mide la cantidad de artefactos que introduce el algoritmo (entendemos artefacto como errores no debidos ni a interferencias ni a errores de escala). En este caso, también son preferibles valores positivos y altos.

Una vez tenemos las direcciones que nos gustaría que nuestras métricas tomaran, podemos proceder al análisis. Démonos cuenta de que existe un desequilibrio bastante importante en el rango de valores, con uno claramente peor que el resto (SI-SDR), otro malo pero no al extremo del anterior (SIR) y otro que es mejorable (SAR). Si además tenemos en cuenta que estos valores están en escala logarítmica (al medirse las métricas en dB), nos damos cuenta de que el problema parece residir en la calidad global que es capaz de arrojar el modelo (despreciamos los otros dos).

Ahora que ya conocemos lo que parece ser el principal problema del sistema, podemos intentar dar explicaciones del porqué de estos resultados. Las causas de este problema pueden venir de varias direcciones. Por una parte, puede ser un problema de la arquitectura en sí. Esto podría ser el caso si el downsampling realizado es demasiado agresivo o de demasiada poca calidad. Se podría arreglar con un proceso más paulatino, empleando una representación de datos más compacta (espectrogramas o embeddings) o probando otras

arquitecturas distintas a la planteada (por ejemplo, arquitecturas híbridas). Otra posibilidad es que la raíz del problema sea la función de pérdida aprendida. Recordemos que esta es una función menos elaborada que en otras tareas con mejores resultados, por lo que esta podría ser modificada de distintas formas para intentar lograr mejores propiedades, por ejemplo, incluyendo una componente invariante a la escala como en otros casos. La última alternativa es que la raíz del problema sea simplemente una cuestión de escala. Incluso para arquitecturas en principio potentes, puede ser que no estemos logrando un rendimiento bueno ya que resulta imposible captar relaciones y dependencias que dañan el rendimiento.

6. CONCLUSIONES Y TRABAJOS FUTUROS

La primera de las conclusiones que podemos extraer de la realización de este proyecto es lo amplio que es el campo del tratamiento de audio, así como lo poco que se llega a cubrir en un único trabajo. Sin embargo, que esto no nos impida extraer algunas conclusiones de cada una de las tareas y de los experimentos realizados.

En cuanto a la reducción de ruido, la primera gran conclusión que podemos extraer es que, coincidiendo con nuestro conocimiento a priori, los *embeddings* son una forma de representar la información más potente que la forma de onda. Por otra parte, pudimos comprobar que distintas arquitecturas tienen un mejor desempeño relativo en distintas condiciones: la arquitectura convolucional opera mejor con ondas, mientras que la arquitectura U-Net lo hace con *embeddings*.

En cuanto a la tarea de compresión, resultó evidente que la funcionalidad acota la CR que podemos lograr (si no podemos recuperar la información, resulta inútil que la podamos comprimir mucho). Por otra parte, quedó de manifiesto que no se cumple siempre que el modelo más pesado tenga un rendimiento mejor que otro más ligero. Otros factores como el diseño de la arquitectura pueden jugar un papel en los resultados que se pueden obtener en una tarea concreta.

Respecto de la última tarea, quedó de manifiesto que es, de lejos, la más compleja de las tres. El rendimiento dadas las restricciones de tiempo, escala, conocimientos y experiencia previos deja en relieve esto mismo, al obtenerse resultados poco convincentes, incluso empleando arquitecturas supuestamente potentes.

Respecto de los objetivos iniciales, éstos quedan cubiertos: se ha investigado y aprendido acerca del tratamiento de audio y las principales tareas disponibles. Se han considerados distintos aspectos para la compleción del proyecto: arquitecturas, funciones de pérdida, conjuntos de datos, etc. Los experimentos llevados a cabo se han mantenido dentro de los límites de la infraestructura prestablecida de 6 GB de memoria durante el entrenamiento. Además, los experimentos han garantizado que la experiencia sea práctica y no puramente teórica.

Por último, y acerca de los trabajos futuros posibles estos pueden ir en varias direcciones. La primera y más evidente es la profundización en algún aspecto del presente trabajo: distintas funciones de pérdida, otras arquitecturas no consideradas, etc. Esta posibilidad es particularmente interesante en la tarea con peores resultados obtenidos (separación por fuentes) y que, además, puede ofrecer un aprendizaje más rico. Alternativamente, también se pueden desarrollar experimentos con otras tareas diferentes no consideradas aquí. En cualquier caso, el espectro de posibilidades es amplio y muy rico. Este trabajo ha servido únicamente (y de manera satisfactoria) como una introducción al campo, y se puede afirmar con firmeza que ha cumplido su propósito.

7. BIBLIOGRAFÍA

[1] A. Baevski, H. Zhou, A. Mohamed, y M. Auli, «wav2vec 2.0: A Framework for Self-Supervised Learning of Speech Representations», 22 de octubre de 2020, *arXiv*: arXiv:2006.11477. doi: 10.48550/arXiv.2006.11477.

[2] J.-M. Valin, «A Hybrid DSP/Deep Learning Approach to Real-Time Full-Band Speech Enhancement», 31 de mayo de 2018, *arXiv*: arXiv:1709.08243. doi: 10.48550/arXiv.1709.08243.

[3] M. Abboush, C. Knieke, y A. Rausch, «GRU-Based Denoising Autoencoder for Detection and Clustering of Unknown Single and Concurrent Faults during System Integration Testing of Automotive Software Systems», *Sensors*, vol. 23, n.º 14, p. 6606, jul. 2023, doi: 10.3390/s23146606.

[4] H. Wang y B. Tian, «ZipEnhancer: Dual-Path Down-Up Sampling-based Zipformer for Monaural Speech Enhancement», 9 de enero de 2025, *arXiv*: arXiv:2501.05183. doi: 10.48550/arXiv.2501.05183.

[5] N. Zeghidour, A. Luebs, A. Omran, J. Skoglund, y M. Tagliasacchi, «SoundStream: An End-to-End Neural Audio Codec», 7 de julio de 2021, *arXiv*: arXiv:2107.03312. doi: 10.48550/arXiv.2107.03312.

[6] Biing-Hwang Juang y A. Gray, «Multiple stage vector quantization for speech coding», en *ICASSP '82. IEEE International Conference on Acoustics, Speech, and Signal Processing*, Paris, France: Institute of Electrical and Electronics Engineers, 1982, pp. 597-600. doi: 10.1109/ICASSP.1982.1171604.

[7] A. Défossez, J. Copet, G. Synnaeve, y Y. Adi, «High Fidelity Neural Audio Compression», 2022, *arXiv*. doi: 10.48550/ARXIV.2210.13438.

[8] 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.

[9] O. Ronneberger, P. Fischer, y T. Brox, «U-Net: Convolutional Networks for Biomedical Image Segmentation», 18 de mayo de 2015, *arXiv*: arXiv:1505.04597. doi: 10.48550/arXiv.1505.04597.

[10] G. Fabbro *et al.*, «The Sound Demixing Challenge 2023 – Music Demixing Track», *Trans. Int. Soc. Music Inf. Retr.*, vol. 7, n.º 1, pp. 63-84, abr. 2024, doi: 10.5334/tismir.171.

[11] S. Rouard, F. Massa, y A. Défossez, «Hybrid Transformers for Music Source Separation», 15 de noviembre de 2022, *arXiv*: arXiv:2211.08553. doi: 10.48550/arXiv.2211.08553.

[12] J. B. Allen y L. R. Rabiner, «A unified approach to short-time Fourier analysis and synthesis», *Proc. IEEE*, vol. 65, n.º 11, pp. 1558-1564, 1977, doi: 10.1109/PROC.1977.10770.

[13] S. Bai, J. Z. Kolter, y V. Koltun, «An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling», 19 de abril de 2018, *arXiv*: arXiv:1803.01271. doi: 10.48550/arXiv.1803.01271.

[14] Nair, Vinod & Hinton, Geoffrey, «Rectified Linear Units Improve Restricted Boltzmann Machines», *Proc. ICML*, vol. 27, pp. 807-814, 2010.

[15] D. Stoller, S. Ewert, y S. Dixon, «Wave-U-Net: A Multi-Scale Neural Network for End-to-End Audio Source Separation», 8 de junio de 2018, *arXiv*: arXiv:1806.03185. doi: 10.48550/arXiv.1806.03185.

[16] T. Silva y F. Moraes, *A Fully Convolutional Neural Network for Speech Enhancement*. Python, Tensorflow. [En línea]. Disponible en: https://github.com/EncoraDigital/SAB-cnn-audio-denoiser

[17] D. P. Kingma y J. Ba, «Adam: A Method for Stochastic Optimization», 30 de enero de 2017, *arXiv*: arXiv:1412.6980. doi: 10.48550/arXiv.1412.6980.

[18] G. E. Hinton y R. R. Salakhutdinov, «Reducing the Dimensionality of Data with Neural Networks», *Science*, vol. 313, n.º 5786, pp. 504-507, jul. 2006, doi: 10.1126/science.1127647.

[19] J. Masci, U. Meier, D. Cireşan, y J. Schmidhuber, «Stacked Convolutional Auto-Encoders for Hierarchical Feature Extraction», en *Artificial Neural Networks and Machine Learning – ICANN 2011*, vol. 6791, T. Honkela, W. Duch, M. Girolami, y S. Kaski, Eds., en Lecture Notes in Computer Science, vol. 6791. , Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 52-59. doi: 10.1007/978-3-642-21735-7_7.

[20] K. He, X. Zhang, S. Ren, y J. Sun, «Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification», 6 de febrero de 2015, *arXiv*: arXiv:1502.01852. doi: 10.48550/arXiv.1502.01852.

[21] J. Chen, S. Vekkot, y P. Shukla, «Music Source Separation Based on a Lightweight Deep Learning Framework (DTTNET: DUAL-PATH TFC-TDF UNET)», en *ICASSP 2024 - 2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, abr. 2024, pp. 656-660. doi: 10.1109/ICASSP48485.2024.10448020.

[22] J. L. Ba, J. R. Kiros, y G. E. Hinton, «Layer Normalization», 21 de julio de 2016, *arXiv*: arXiv:1607.06450. doi: 10.48550/arXiv.1607.06450.

[23] Z. Wu, Z. Liu, J. Lin, Y. Lin, y S. Han, «Lite Transformer with Long-Short Range Attention», 24 de abril de 2020, *arXiv*: arXiv:2004.11886. doi: 10.48550/arXiv.2004.11886.

[24] A. Vaswani *et al.*, «Attention Is All You Need», 2 de agosto de 2023, *arXiv*: arXiv:1706.03762. doi: 10.48550/arXiv.1706.03762.

[25] D. Bahdanau, K. Cho, y Y. Bengio, «Neural Machine Translation by Jointly Learning to Align and Translate», 19 de mayo de 2016, *arXiv*: arXiv:1409.0473. doi: 10.48550/arXiv.1409.0473.

[26] G. Tzanetakis y P. Cook, «Musical genre classification of audio signals», *IEEE Trans. Speech Audio Process.*, vol. 10, n.º 5, pp. 293-302, jul. 2002, doi: 10.1109/TSA.2002.800560.

[27] E. Vincent, A. Yeredor, Z. Koldovský, y P. Tichavský, Eds., *Latent Variable Analysis and Signal Separation: 12th International Conference, LVA/ICA 2015, Liberec, Czech Republic, August 25-28, 2015, Proceedings*, vol. 9237. en Lecture Notes in Computer Science, vol. 9237. Cham: Springer International Publishing, 2015. doi: 10.1007/978-3-319-22482-4 (Pags. 323-332).

[28] D. Britz, A. Goldie, M.-T. Luong, y Q. Le, «Massive Exploration of Neural Machine Translation Architectures», 21 de marzo de 2017, *arXiv*: arXiv:1703.03906. doi: 10.48550/arXiv.1703.03906.

[29] T. Akiba, S. Sano, T. Yanase, T. Ohta, y M. Koyama, «Optuna: A Next-generation Hyperparameter Optimization Framework», 25 de julio de 2019, *arXiv*: arXiv:1907.10902. doi: 10.48550/arXiv.1907.10902.

[30] L. R. Rabiner y B.-H. Juang, *Fundamentals of speech recognition*. en Prentice Hall signal processing series. Englewood Cliffs (N.J.): Prentice-Hall, 1993.

[31] J. L. Roux, S. Wisdom, H. Erdogan, y J. R. Hershey, «SDR – Half-baked or Well Done?», en *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Brighton, United Kingdom: IEEE, may 2019, pp. 626-630. doi: 10.1109/ICASSP.2019.8683855.

[32] Choi, W., Kim, M., Chung, J., Lee, D., & Jung, S., «Investigating U-nets with various intermediate blocks for spectrogram-based singing voice separation», *ISMIR 2020*, 2020, [En línea]. Disponible en: https://archives.ismir.net/ismir2020/paper/000046.pdf

[33] A. Liutkus *et al.*, «The 2016 Signal Separation Evaluation Campaign», en *Latent Variable Analysis and Signal Separation*, vol. 10169, P. Tichavský, M. Babaie-Zadeh, O. J. J. Michel, y N. Thirion-Moreau, Eds., en Lecture Notes in Computer Science, vol. 10169. , Cham: Springer International Publishing, 2017, pp. 323-332. doi: 10.1007/978-3-319-53547-0_31.

[34] K. Cho *et al.*, «Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation», 3 de septiembre de 2014, *arXiv*: arXiv:1406.1078. doi: 10.48550/arXiv.1406.1078.