TP4 TDVI

Ezequiel Grinblat, Luca Mazzarello y Camila Migdal

November 2024

1 Introducción

Para este proyecto se nos planteo el desafío, de implementar un codificador musical basado en redes neuronales con el fin de generar vectores representativos en un espacio latente para cada canción. Estos vectores, una vez obtenidos, podrán ser decodificados para reproducir el sonido original. A lo largo de este informe, desarrollaremos el proceso realizado y las distintas pruebas utilizadas para obtener lo más cercano a los sonidos originales.

2 Diseño del Autoencoder: CELAutoencoder

El modelo CELAutoencoder implementado es una arquitectura basada en redes neuronales convolucionales. Este modelo consta de dos componentes principales: un encoder que reduce progresivamente la dimensionalidad de la entrada y un decoder que reconstruye la entrada original a partir de la representación comprimida.

3 Encodear la canción en un vector latente

3.1 Estructura del Encoder

El encoder aplica una serie de capas convolucionales, normalizaciones batch y la función de activación LeakyReLU, con el objetivo de extraer características relevantes de la señal de entrada, utilizamos esa función de activación ya que al estar tarabajando con audios el dominio toma valores menores a cero. A continuación, detallamos como está compuesto el encoder:

- Primera capa convolucional: Convierte la señal unidimensional inicial con un canal en una representación de 16 canales, manteniendo el tamaño original gracias al uso de un padding adecuado.
- Emplea capas de max pooling para reducir la longitud de la señal en puntos clave del flujo, logrando una compresión significativa.

- Incrementa progresivamente el número de filtros (32, 64, 128, 256, 512), permitiendo al modelo capturar patrones más complejos y detallados.
- La salida del encoder es un tensor con 512 canales y una longitud reducida, que representa una codificación comprimida de la señal original.

3.2 Estructura del Decoder

El decoder realiza la operación inversa al encoder, utilizando capas de convolución transpuesta para ampliar la dimensionalidad de la señal comprimida y reconstruirla en su forma original. Sus etapas principales son:

- Capas de convolución transpuesta: Incrementan progresivamente la longitud de la señal a medida que disminuyen los canales (512 $\rightarrow$ 256 $\rightarrow$ 128 $\rightarrow$ 64 $\rightarrow$ 32 $\rightarrow$ 16 $\rightarrow$ 1).
- Interpolación: Se usan capas de Upsample para ajustar la longitud exacta de la señal a valores específicos (110250, 55125, 27562, etc.), asegurando compatibilidad en los tamaños.
- Reconstrucción final: Una capa de convolución transpuesta con un canal genera la salida reconstruida, utilizando una activación Tanh para normalizar los valores en un rango entre -1 y 1, ya que estamos trabajando con audios.

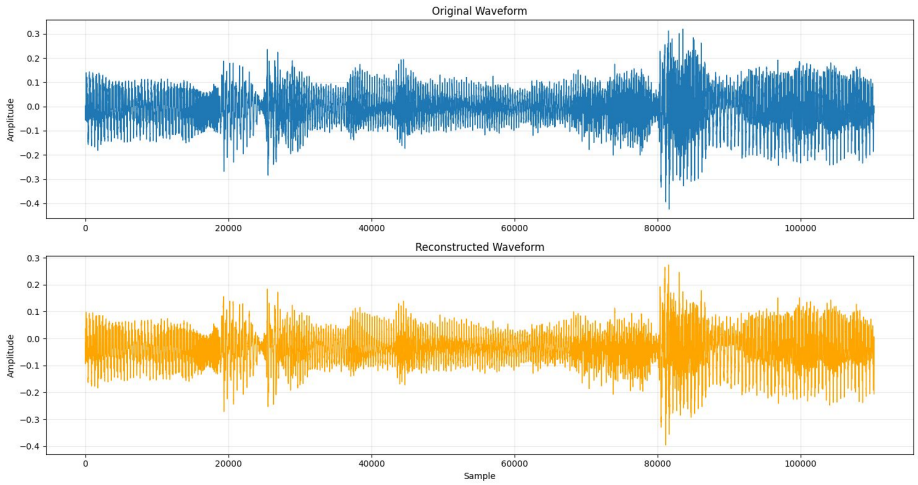


Figure 1: Resultado del CELAutoencoderLR

3.3 Primeros resultados

Podemos observar en la figura 1, que ambas señales mantienen una estructura similar, lo que sugiere que el autoencoder es capaz de conservar las características esenciales de la señal original durante la reconstrucción. Sin embargo,

la amplitud de la señal reconstruida parece más suavizada en comparación con la señal original, lo que indica una posible pérdida de detalles finos debido al proceso nombrado. En los proximos pasos buscaremos minimizar la pérdida de detalles y mejorar la calidad de la reconstrucción.

Con este modelo entrenado, obtuvimos una train loss de 0.006 y un tamaño de vector latente minimizado a 512x6890, habiendo partido de un input de 1x110250.

4 Análisis exploratorio de vectores latentes

Para explorar el tamaño de los vectores latentes, decidimos realizar tres alternativas de comparación. Partiendo del generado anteriormente, de un tamaño de: 512x6890 , vamos a generar uno más pequeño y otro más grande para nuestro analisis exploratorio.

Para ello, realizamos primero el CELAutoencoder2 buscando obtener un espacio latente más grande (512x13781). Como resultados obtuvimos una train loss de **0.0043** y la siguiente reconstrucción:

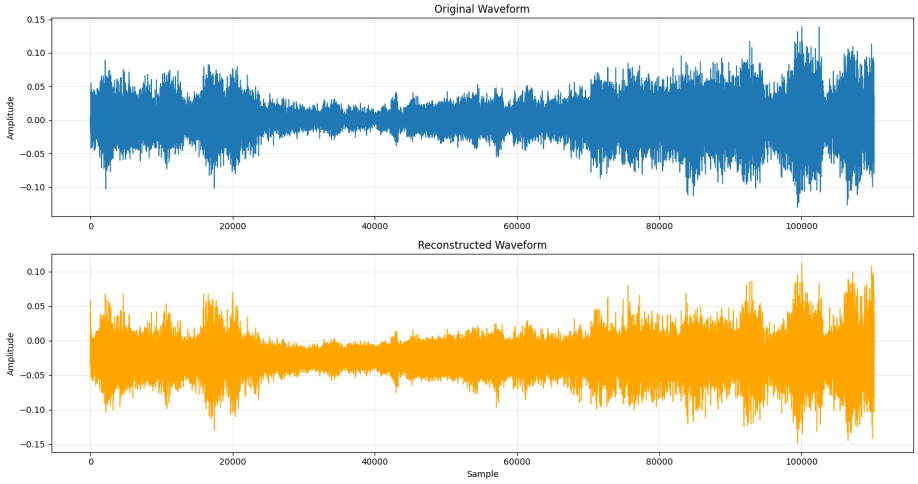


Figure 2: Resultados del CELAutoencoder2

Luego, para el caso de un espacio latente mas pequeño, utilizamos el CELAutoencoder3, autoencoder cuyas deimensiones de su espacio latente es de 256x861. Como resultados obruvimos una train loss de 0.0146

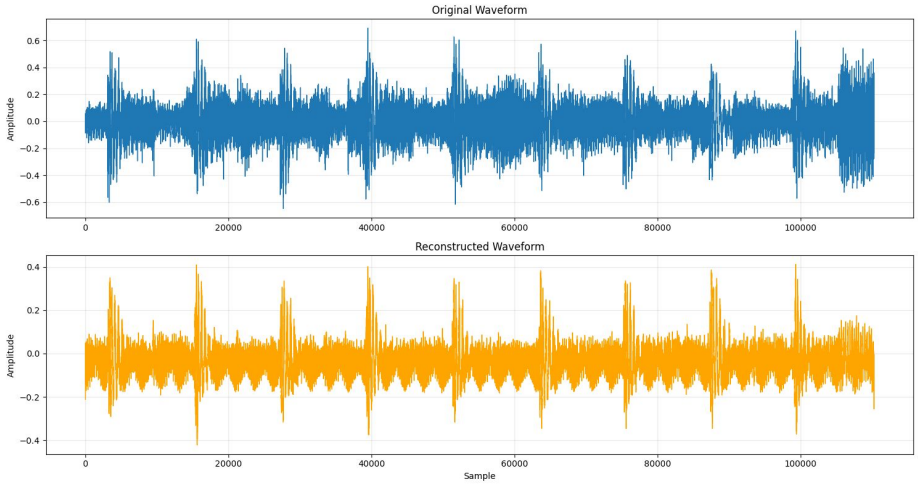


Figure 3: Resultados del CELAutoencoder3

Por ultimo realizamos una tercer prueba, con el espacio de tamaño intermedio pero cambiando la función de activación SiLU, debido a que tiene una mejor capacidad que LeakyReLU para modelar relaciones más complejas. Obtuvimos train loss de: 0.0057.

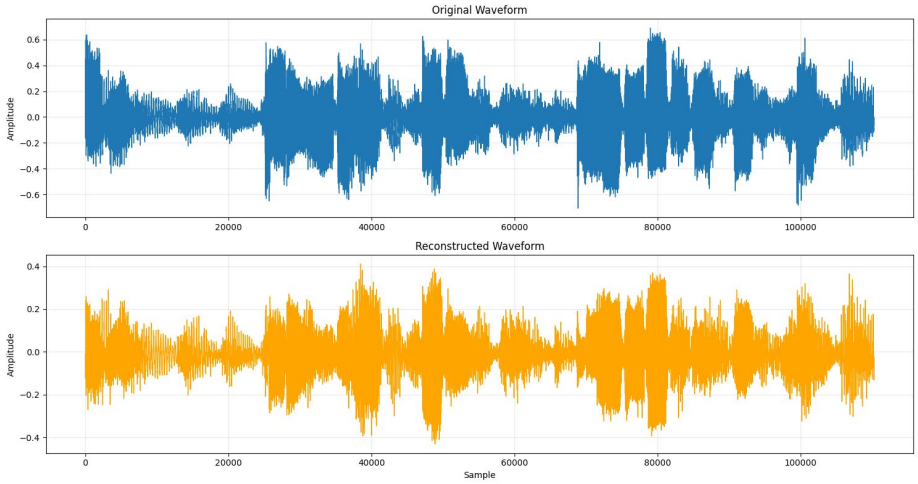


Figure 4: Resultados del CELAutoencoderSiLU

Tras entrenar tres modelos con diferentes tamaños de espacio latente, observamos variaciones significativas en las métricas de reconstrucción. El primer autoencoder, con un espacio latente de 512×6890, logró una *train loss* de 0.006. Al aumentar el espacio latente a 512×13781, el modelo redujo la *train loss* a

0.0043, evidenciando que un espacio mayor permite captar patrones más complejos y mejorar la calidad de la reconstrucción, en el modelo generado. Por otro lado, al reducir el espacio latente a 256×861, la *train loss* incrementó a 0.0146, indicando que un espacio más pequeño puede sacrificar demasiada información esencial. Finalmente, al mantener un tamaño intermedio pero cambiando la función de activación a SiLU, se obtuvo una *train loss* de 0.0057. En este experimento, nos llamo la atención el hecho de obtener mejores resultados en un espacio más grande pero inferimos que la pérdida de información en el espacio latente más pequeño supera los beneficios de la compresión, lo que resulta en un peor desempeño medido por la *train loss*.

5 Modelos de Clasificación

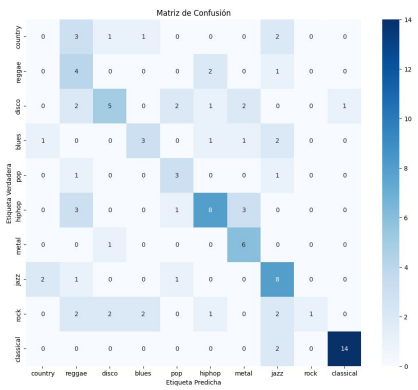
5.1 Clasificador propio

Dada la busqueda del mejor autoenconder, pasamos al armado del clasificador. Durante el desarrollo, probamos tanto con el encoder del CELAutoencoder2 como con el del CELAutoencoderSiLU como con el del CELAutoencoderLR para el entrenamiento , ya que estos tres modelos ofrecieron los mejores resultados en términos de train loss.

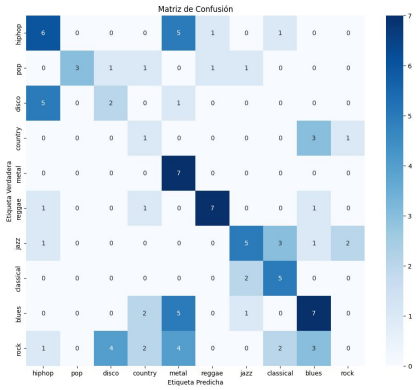
El clasificador incluye una capa de Adaptive Average Pooling para reducir la dimensionalidad, seguida de capas completamente conectadas con funciones de activación LeakyReLU y dropout para prevenir el sobreajuste. Finalmente, la última capa lineal mapea las características extraídas a las probabilidades de pertenencia a los géneros musicales, considerando un total de 10 clases. A continuación podemos observar los resultados obtenidos.

Clase	**Precisión**	**Recall**	**F1-Score**	**Soporte**
Country	0.00	0.00	0.00	7
Reggae	0.25	0.57	0.35	7
Disco	0.56	0.38	0.45	13
Blues	0.50	0.38	0.43	8
Pop	0.43	0.60	0.50	5
Hiphop	0.62	0.53	0.57	15
Metal	0.50	0.86	0.63	7
Jazz	0.44	0.67	0.53	12
Rock	1.00	0.10	0.18	10
Classical	0.93	0.88	0.90	16

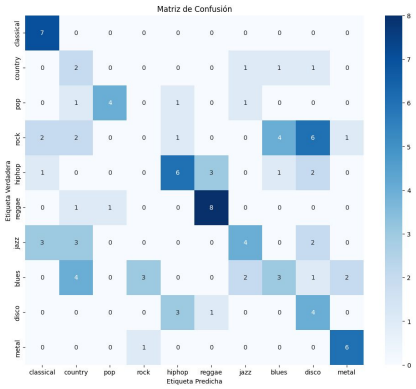
Table 1: Resultados de métricas por clase para clasificación.



(a) Matriz de confusión con CELAutoencoderLeakyReLU



(b) Matriz de confusión con CELAutoencoderSiLU



(c) Reporte de clasificación con CELAutoencoder2

Figure 5: Comparación de matrices de confusión y reportes de clasificación

5.2 XG-Boost

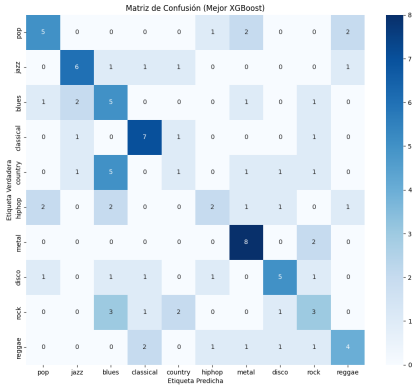
Luego planteamos un nuevo clasificador con el modelo de XG-Boost, donde a partir del vector latente obtenido por nuestros dos enconder realiza la clasificación. A continuación se muestra una tabla, con los mejores resultados obtenidos. Se obtuvieron utilizando el CELAutoencoder2:

Podemos observar que las clases classical y metal obtienen el mejor rendimiento con F1-Scores de 0.70 y 0.67, respectivamente. Sin embargo, clases como country y rock tienen F1-Scores bajos (0.22 y 0.29), lo que indica dificultades para capturar sus características distintivas.

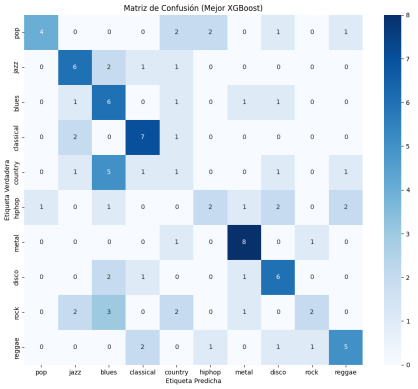
A continuación podemos observar las matrices de confusión realizadas con los distintos autoeconders con el modelo de XG-Boost

Clase	Precisión	Recall	F1-Score	Soporte
Pop	0.80	0.40	0.53	10
Jazz	0.50	0.50	0.50	10
Blues	0.33	0.40	0.36	10
Classical	0.70	0.70	0.70	10
Country	0.25	0.20	0.22	10
Hiphop	0.80	0.44	0.57	9
Metal	0.57	0.80	0.67	10
Disco	0.50	0.70	0.58	10
Rock	0.27	0.30	0.29	10
Reggae	0.50	0.50	0.50	10

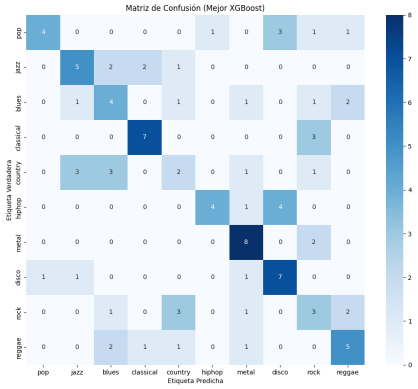
Table 2: Resultados de métricas por clase para XG-Boost.



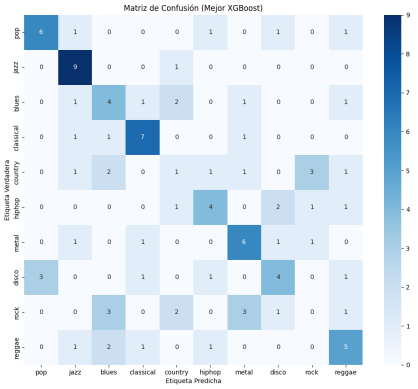
(a) AutoencoderLR - Espacio latente intermedio



(b) CELAutoencoderSiLU - Espacio latente intermedio



(c) CELAutoencoder2 - Espacio latente más grande



(d) CELAutoencoder3 - Espacio latente más chico

Figure 6: Comparación del clasficador XG-Boost para los distintos autoencoders

Podemos notar que, si bien el autoencoder mencionado anteriormente obtuvo los mejores resultados, los demás modelos presentan desempeños muy similares, exceptuando el CELAutoencoder3, que muestra los peores resultados debido a cuyo espacio latente más pequeño.

5.3 PCA

Luego pasamos a la etapa de clustering para lo que utilizamos PCA para facilitar tanto la visualización como la interpretación del comportamiento de los datos en un espacio reducido.

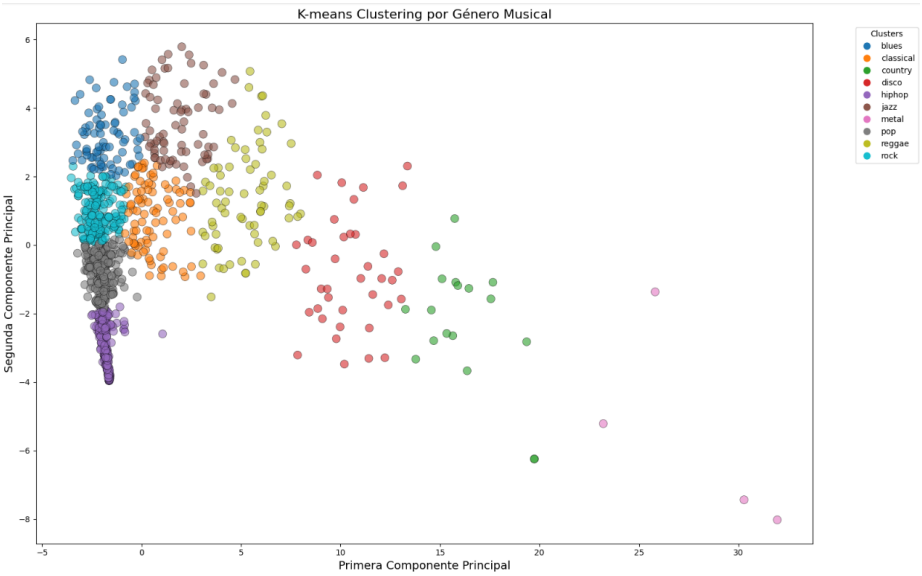
Comenzamos cargando el CELAutoencoderSiLU previamente entrenado, que se encargó de generar un vector latente con las características más relevantes de las canciones. Estas características, fueron escaladas y transformadas mediante PCA para reducir su dimensionalidad a solo dos componentes principales.

Posteriormente, empleamos técnicas de clustering como K-Means, Gaussian Mixture y Hierarchical Clustering para identificar patrones y agrupar las canciones según sus características, utilizando las etiquetas de género como referencia para interpretar los resultados.

5.3.1 K-MEANS

Los resultados del clustering utilizando el método k-means indican una correspondencia limitada entre los grupos encontrados por el algoritmo y las etiquetas de género originales del dataset. Esto se observa en las métricas de evaluación obtenidas:

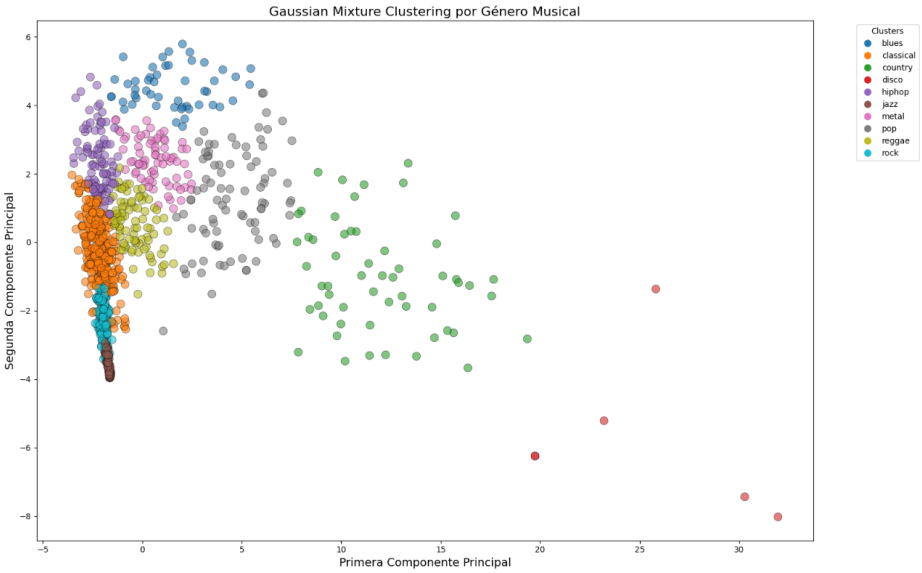
- Adjusted Rand Index (ARI): Con un valor de 0.079, esta métrica refleja que el acuerdo entre las etiquetas originales y las asignadas por el clustering es apenas superior al azar.
- Adjusted Mutual Information (AMI): Con un valor de 0.138, esta métrica sugiere una ligera información compartida entre las etiquetas reales y las del clustering, pero sigue siendo baja.



5.3.2 Gaussian Mixture

Los resultados obtenidos utilizando el modelo Gaussian Mixture muestran un desempeño un poco superior al del clustering con k-means, aunque las métricas siguen indicando una relación débil entre los grupos generados y las etiquetas de género reales. Los valores de las métricas son los siguientes:

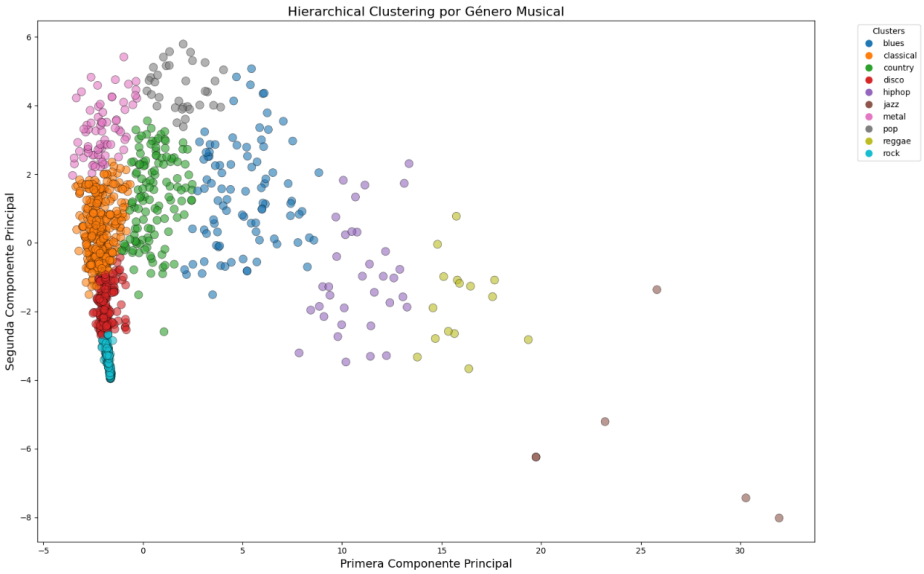
- Adjusted Rand Index (ARI): Un valor de 0.084. Esto indica que el algoritmo no logra identificar bien las diferencias entre los géneros musicales.
- Adjusted Mutual Information (AMI): El valor de 0.139 muestra que la cantidad de información compartida entre las etiquetas reales y las asignadas por el modelo es apenas mejor que el azar.



5.3.3 Hierarchical Clustering

Los resultados obtenidos utilizando el modelo de Hierarchical Clustering muestran un desempeño que, aunque ligeramente superior al azar, sigue siendo modesto en cuanto a la capacidad de identificar una relación clara entre los grupos generados y las etiquetas de género reales. Las métricas obtenidas son las siguientes:

- Índice Rand Ajustado (ARI): Con un valor de 0.082, esto sugiere que no ha logrado capturar adecuadamente las diferencias entre los géneros musicales, y la segmentación generada por el modelo es débil.
- Información Mutua Ajustada (AMI): El valor de 0.125 muestra que la cantidad de información compartida entre las etiquetas reales y las asignadas por el modelo es baja. Aunque hay cierta correspondencia, esta sigue siendo limitada.



6 Encodear música nueva

En esta sección, se evaluó la capacidad del autoencoder para generar una representación latente de una canción que no forma parte del conjunto de datos utilizado para su entrenamiento, con el objetivo de verificar su desempeño al trabajar con música nueva. El proceso comenzó con la carga de una canción MP3, la cual fue convertida a formato WAV y luego preprocesada para ajustarse a los requisitos del modelo. Esto incluyó la conversión a mono, el re-muestreo al sample rate objetivo y el padding de la canción para que tuviera una duración uniforme de 5 segundos. Posteriormente, se normalizó la señal de audio y se ajustó su forma para ser compatible con el modelo de autoencoder.

Este procedimiento fue realizado utilizando tres modelos de autoencoder diferentes: CELAutoencoderSiLU, CELAutoeconder2 y CELAutoencoder3. Con los modelos CELAutoencoderSiLU y CELAutoencoder2, los resultados fueron satisfactorios, ya que las canciones reconstruidas mostraron una buena calidad y fidelidad respecto a las originales. Sin embargo, con el CELAutoencoder3, que posee un espacio latente más pequeño, los resultados fueron significativamente peores, ya que la capacidad de este modelo para capturar la información latente necesaria para la reconstrucción de la canción fue limitada. Esto sugiere que el tamaño del espacio latente influye directamente en la capacidad del modelo para generalizar y reconstruir adecuadamente música fuera del conjunto de entrenamiento.

Una vez preprocesado el audio, el modelo fue utilizado para generar una versión reconstruida de la canción a partir de su representación latente. Se compararon la forma de onda original y la reconstruida, tanto visualmente como

mediante la reproducción de ambos audios. La calidad de la reconstrucción puede ser evaluada observando las diferencias en las formas de onda y la fidelidad del audio reconstruido respecto al original. Este procedimiento permite evaluar si el autoencoder puede generalizar su capacidad de codificación y decodificación a nuevas muestras de audio, fuera del conjunto de entrenamiento utilizado.

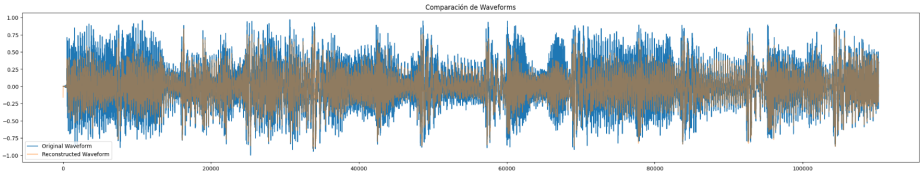


Figure 7: Autoencoder aplicado a la canción "Me quiere mucho, poquito o nada"

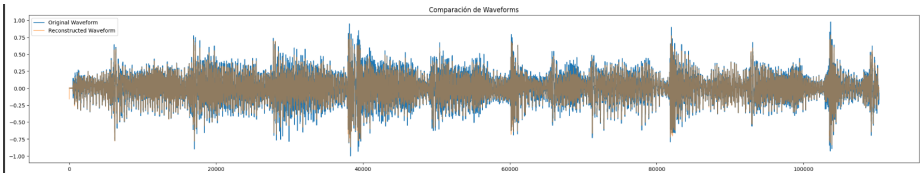


Figure 8: Autoencoder aplicado a la canción "Flores Amarillas"

7 Generación de música

Durante el procesamiento del audio con el autoencoder, hemos realizado diversas modificaciones en el espacio latente con el objetivo de observar cómo estas alteraciones impactan las características del audio reconstruido. Al manipular esta representación compacta, es posible generar nuevas variantes del audio que reflejan los cambios aplicados.

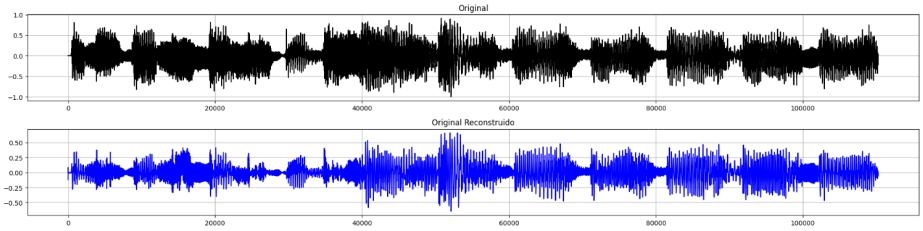
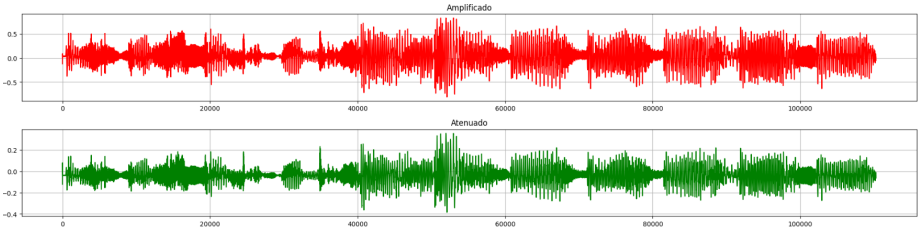


Figure 9: grafico original y autoencoder de la canción Osito Gominola

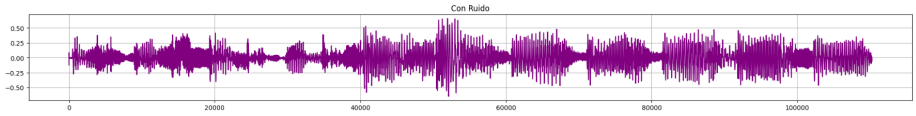
Las modificaciones realizadas incluyen:

- **Amplificación y Atenuación**: Multiplicamos el vector latente por factores como 1.5 (amplificación) o 0.5 (atenuación). Esto simula un cambio

en la intensidad de las características del audio, parecido a subir o bajar el volumen.



- **Adición de Ruido**: Incorporamos un pequeño nivel de ruido gaussiano al espacio latente. Esta operación permite analizar la robustez del modelo y cómo responde a perturbaciones menores, generando una situación con interferencias.



- **Suavizado**: Aplicamos un suavizado mediante un *pooling* en el espacio latente para ver cómo se ven afectadas las transiciones bruscas entre componentes del audio, generando versiones más "planas" del sonido reconstruido.

Estas transformaciones son posibles gracias a la naturaleza vectorial y continua del espacio latente, lo que facilita operaciones directas. Además, el autoencoder puede decodificar estas variaciones para generar salidas plausibles, ya que ha sido entrenado para mapear representaciones latentes a canciones o sonidos. Esto nos deja ver lo util y adaptable que son los autoencoders tanto para la manipulación de audio como para la experimentación para comprender cómo distintas modificaciones afectan las características percibidas de un sonido.

8 Conclusión

En este trabajo práctico, tuvimos que diseñar e implementar un autoencoder para la codificación y reconstrucción de canciones, lo que nos permitió explorar el uso de distintos modelos e hiperparámetros para el procesamiento de señales de audio. A lo largo del proyecto, aprendimos cómo modifica el tamaño del espacio latente la calidad de la reconstrucción y la importancia de las funciones de activación para capturar patrones complejos.

Además, evaluamos cómo diferentes configuraciones del modelo afectan el desempeño tanto en la reconstrucción del audio como en la clasificación de los distintos géneros de las canciones, lo que nos sirvió para experimentar con técnicas como clustering y reducción de dimensionalidad para el análisis de datos.

Finalmente, al trabajar con datos nuevos y manipular representaciones latentes para generar variaciones de audio, descubrimos la gran versatilidad de estas herramientas. Este trabajo nos ayudó a entender mejor el uso de redes neuronales, a analizar con mayor precisión los resultados obtenidos y a trabajar en torno a estos para mejorar nuestros modelos.