Trabajo Práctico 4 - Encoders, decoders y autoencoders

Tecnología Digital VI: Inteligencia Artificial

Federico Giorgi Gastón Loza Montaña Tomás Curzio

24/11/23

Encodear canciones en vectores latentes

Para este primer inciso, dado que en principio no teníamos mucha información acerca de qué tipo de estructuras de redes eran convenientes para este tipo de problemas, nos enfocamos en primer lugar de tener un código del modelo y entrenamiento que funcione correctamente. Una vez alcanzado ese objetivo, procedimos a probar algunas ideas registrando los experimentos con la herramienta `wandb`.

Luego de un poco de investigación acerca de autoencoders para audio[1] y algunos experimentos más, concluimos en una arquitectura de una red sencilla, muy similar a los autoencoders vistos en clase, utilizando la estructura que se puede ver en la figura 1.

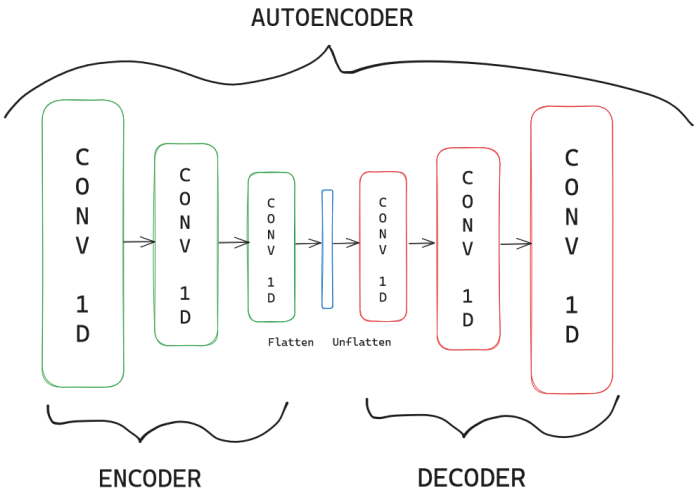

Figura 1: Estructura autoencoder

Principalmente, pudimos identificar que una de las consideraciones más importantates a tener en cuenta era la forma en la que se reducía la dimensionalidad a lo largo de las capas covolucionales. Realizar en primera instancia una capa que aumentara la dimensión a más de 1x110250, ya sea por muchos canales de salida o por un stride chico, para luego reducirla abruptamente en las siguientes dos capas, generaba bastante ruido. En cambio, hacer que la primera capa tal vez no reduzca casi nada la dimensionalidad, pero sí reorganice la información en varios canales para luego reducir la dimensionalidad en las siguientes dos capas genera mejores resultados.

Esta estrategia de ir disminuyendo de a poco o mantener los canales a lo largo de la red, para finalmente flattenear el vector y obtener así nuestro vector latente, nos dio en general buenas codificaciones.

[1] Deep Autoencoders for Music Compression and Genre Classification. *link*

Por ese motivo, seguimos con esta estrategia pero modificando los parámetros de la convoluciones (`output_channels`, `input_channels`, `stride`, `kernel_size` y `stride`). Por esta razón, por fines prácticos y de facilidad de probar varias alternativas, generamos variables en el código que definen estos hiperparámetros de las capas convolucionales.

Con distintas combinaciones de los hiperparámetros de las capas convolucionales realizamos experimentos que dejaran vectores latentes de distintos tamaños, hasta llegar al punto de que la canción sea prácticamente irreconocible. Los audios resultantes se pueden ver en la siguiente tabla, que utiliza la canción Music de Maddona como ejemplo.

Vector latente	
1x110250 (Original)	*link*
1x55112	*link*
1x32151	*link*
1x24496	*link*
1x18376	*link*
1x9184	*link*

En cuanto a los otros hiperparámetros, tomamos las siguientes decisiones:

- En cuanto al `learning rate`, corrimos un `hyperopt` con algunas iteraciones y el resultado obtenido (aproximadamente 0.015) fue muy cercano al `learning rate` que elegimos inicialmente de 0.02. Corrimos algunos experimentos con el learning rate obtenido del `hyperopt` y la diferencia era prácticamente nula, entonces a fines de poder comparar entre los distintos experimentos que ya habíamos realizado, decidimos mantener este parámetro en 0.02.
- Para la definición de cantidad de `epochs`, luego de correr varios experimentos, como se puede ver en la figura 2, notamos que luego de aproximadamente las 20 épocas, ya no se observa una disminución significativa en la función de pérdida, por lo que decimos dejar este hiperparámetro en 40 ya que era el doble de dónde veíamos cierto estancamiento en la función de pérdida pero aún manteníamos un tiempo de entrenamiento bastante bajo que nos permitiera experimentar de manera rápida.
- Con respecto al `batch size` decimos dejar el tamaño que venía dado en el notebook de ejemplo del clasificador y no tuvimos necesidad de alterarlo.

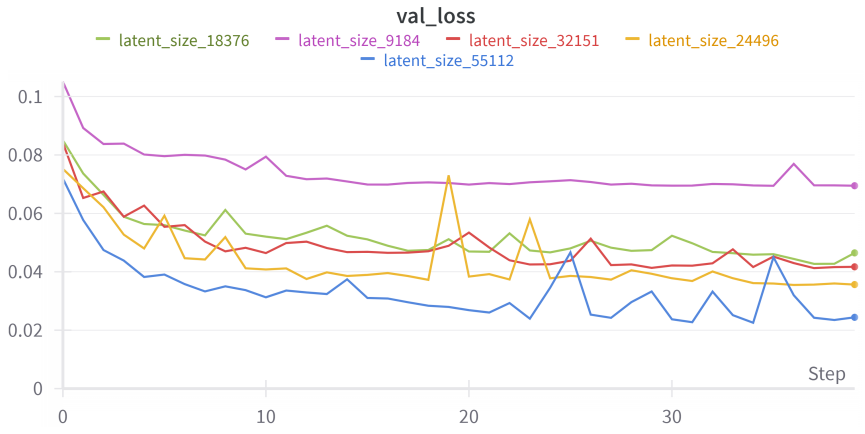

Figura 2: Función de pérdida en validación para cada época

Para no solo evaluar la eficiencia del autoencoder en base a cómo se escucha el audio, también observamos como difieren tanto la waveform como el espectograma de la canción original con los de los distintos audios encodeados y luego decodeados, observados en la tabla. Estos fueron los resultados:

ACA PONER LAS IMAGENES DE WAVEFORM Y ESPECTOGRAMA

Observando y escuchando, decidimos quedarnos como "vector de mínimo tamaño posible" el de 1x18376, ya que consideramos que mantiene una similitud razonable con el audio original, a diferencia del vector de 1x9184, que es prácticamente irreconocible. Con esto, ya podemos avanzar a un análisis exploratorio de los vectores latentes obtenidos.

Análisis exploratorio de vectores latentes

Para el análisis exploratorio de los vectores latentes procedimos a probar distintos métodos vistos a la largo de la materia que nos permita adquirir información interesante.

En primer lugar, exploramos qué resultaba de hacer un clustering con `k-medias`. Para ello, ejecutamos el algoritmo de la libería `sklearn` con valores crecientes de `k` para evaluar cómo varía la función que mide la variabilidad intra cluster para poder observar en cuántos clusters se podría dividir los vectores latentes, con la expectativa que sea una cantidad similar a la cantidad de géneros a los que pertenecen las canciones. Como se puede ver en la figura 3, aún con `k` igual a 40 el modelo sigue con valores elevados de la función objetivo, y no se logra obervar ningún "codo" en el gráfico. Por lo tanto, es claro notar que aún los vectores latentes más chicos (de 1x9184), tienen demasiados atributos y por ende muchas dimensiones para que el algoritmo de `k-medias` puede encontrar una clustering relativamente chico. Por ese motivo, procedemos a analizar otras opciones.

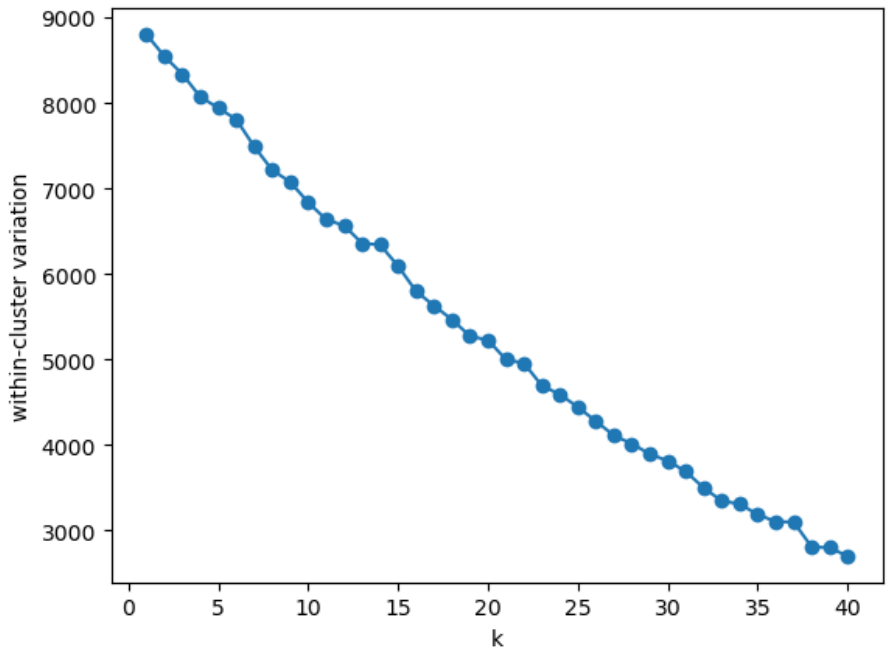

Figura 3: Variabilidad intra clusters a medida que aumenta k (para los vectores latentes de 1x18376)

Otro de los métodos que realizamos en esta exploración, es la análisis de componentes principales (PCA). Para ello, utilizamos el método ya provisto en `sklearn.decomposition` para ejecutar el análisis, habiendo previamente escalado los valores de los vectores, como se suele hacer antes de realizar un PCA.

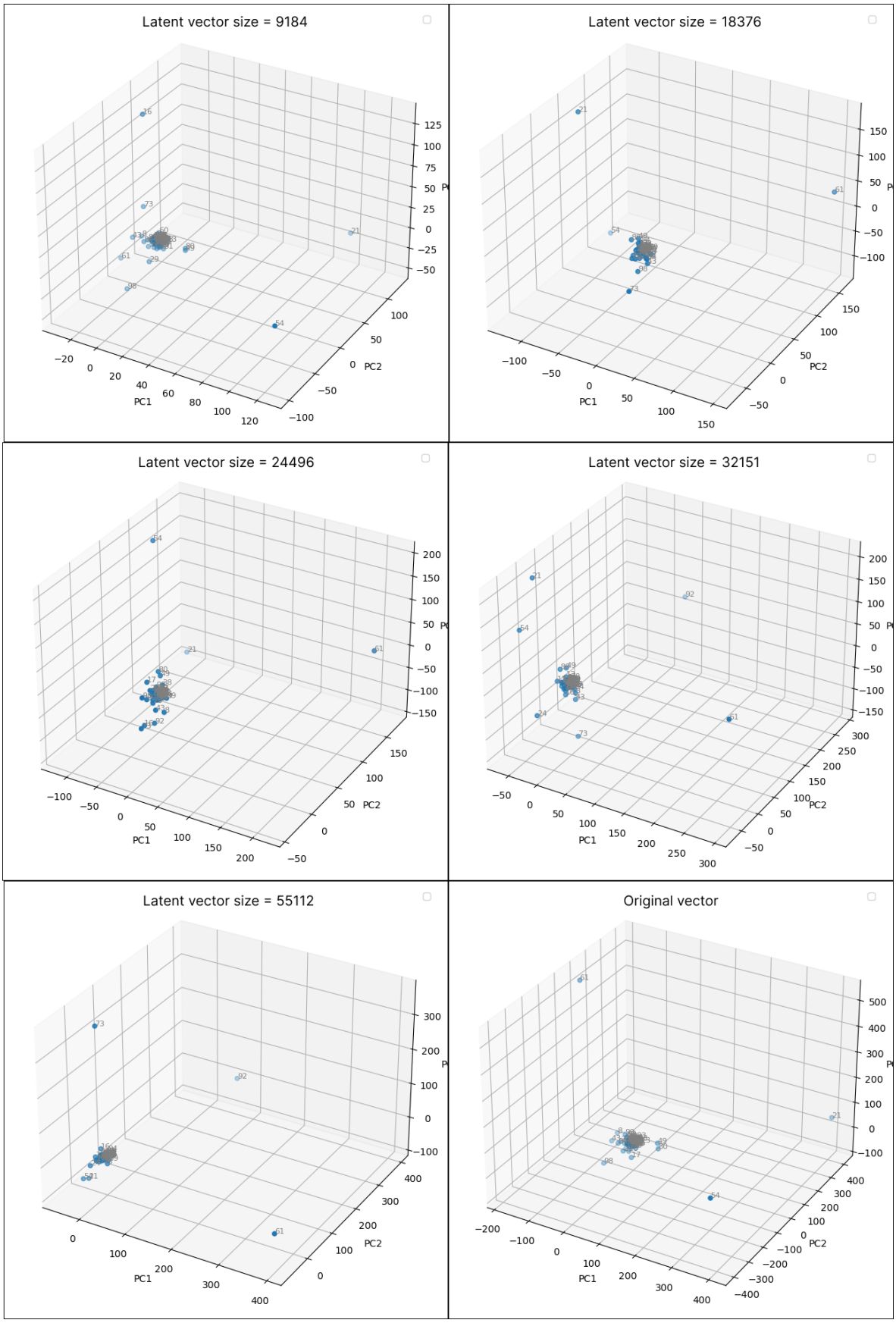

Figura 4: Vectores latentes graficados en sus 3 principales componentes

Para observar resultado del análisis, decidimos graficar los vectores en el nuevo espacio vectorial dado las 3 componentes principales. Como podemos observar en la figura 4, para ninguno de los tamaños de vectores latentes ni para el original se observa una diferenciación clara entre las canciones. Esto muy probablemente se deba al mismo motivo por el que no funcionó `k-medias` dada la gran cantidad de atributos. Sin embargo, podemos notar que los "outliers" que se observan suelen ser los mismos para los diferentes tamaños de vectores latentes (canciones 61, 21 y 54 particularmente). Al escuchar estas canciones, no notamos ninguna característica que nos dé indicio del motivo de su diferenciación con respectoa a las demás canciones. Sin embargo, que coincidan los outliers nos da la idea que a pesar de ser vectores latentes encodeades con diferentes arquitecturas, conservan cierta información de manera similar.

Por último, para poder observar si la información relevante para la clasificación en géneros de las canciones se ve afectada o no por el proceso de codificación, procedemos a realizar para cada tamaño de vector latente y el vector original árboles de decisión para entrenar la clasificación de las canciones en géneros con `sklearn.tree`. Entrenando este clasificador, buscamos poder detectar cuáles y cuántos atributos utiliza el clasificador. Obtenemos esta información mediante el método `feature_importances_` y gráficamos en orden descente los atributos, en este caso coordenadas del vector, según su aporte en la clasificación en entrenamiento.

Como se puede observar en la figura 5, para el vector original y todos los tamaños de vectores latentes, se utilizan sólo entre 14 y 23 coordenadas de los vectores para la clasificación del género de las canciones. Por esta razón, podríamos suponer que la información relevante relacionada al género de las canciones se condensa en promedio en 20 atributos sin importar el tamaño del vector. Este aspecto parece relevante al cosiderar que los vectores latentes fueron obtenidos a tráves de encoders distintos y aún sí conservan información en común, como sucedió con los outliers del análisis de componentes principales.

Encodeo de música nueva

Con la red definida, en un principio no se podría encodear música nueva, a no ser que tanto su sample rate como su tamaño original, sea el mismo que el de los audios del dataset que estamos utilizando para entrenar y validar nuestra red.

Para poder hacerlo, hay algunas opciones:

1. Adaptar el audio que querramos encodear para cumplir los requisitos de la red.
2. Adaptar la red para el sample rate y tamaño original de aquello que querramos encodear.
3. Adaptar el sample rate que viene por defecto al del audio que querramos encodear, y reducir el tamaño del mismo a los requisitos de la red (recortandolo a 5s como los demás).

Tras el siguiente proceso pudimos encodear música nueva.

1. Entrenar la red normalmente, con el dataset otorgado.
2. Nuestro audio era stereo (2 channels) así que lo pasamos a mono con un convertidor online.
3. Una vez convertido a mono, cortamos un fragmento de 5 segundos con el código provisto.
4. Así, el vector era de tamaño 1x1x220500, que no cumple con las características que necesita la red (1x1x110250). Nuestra hipótesis es que esta diferencia en tamaño se debe a que el sample rate es 44100 en lugar de 20500 como los audios del dataset, justamente el doble de sample rate, el doble de tamaño. Para solventarlo, utilizamos la función de Resampling de PyTorch, obteniendo así un vector con las características necesarias.
5. Realizamos un forward en la red con nuestro vector de audio nuevo una vez que cumplía las características necesarias.
6. Obtuvimos el audio reconstruido, así como su espectograma y waveform.
7. Repetimos el proceso para distintos tamaños de vectores latentes y obtuvimos los que se pueden ver en la siguiente tabla.

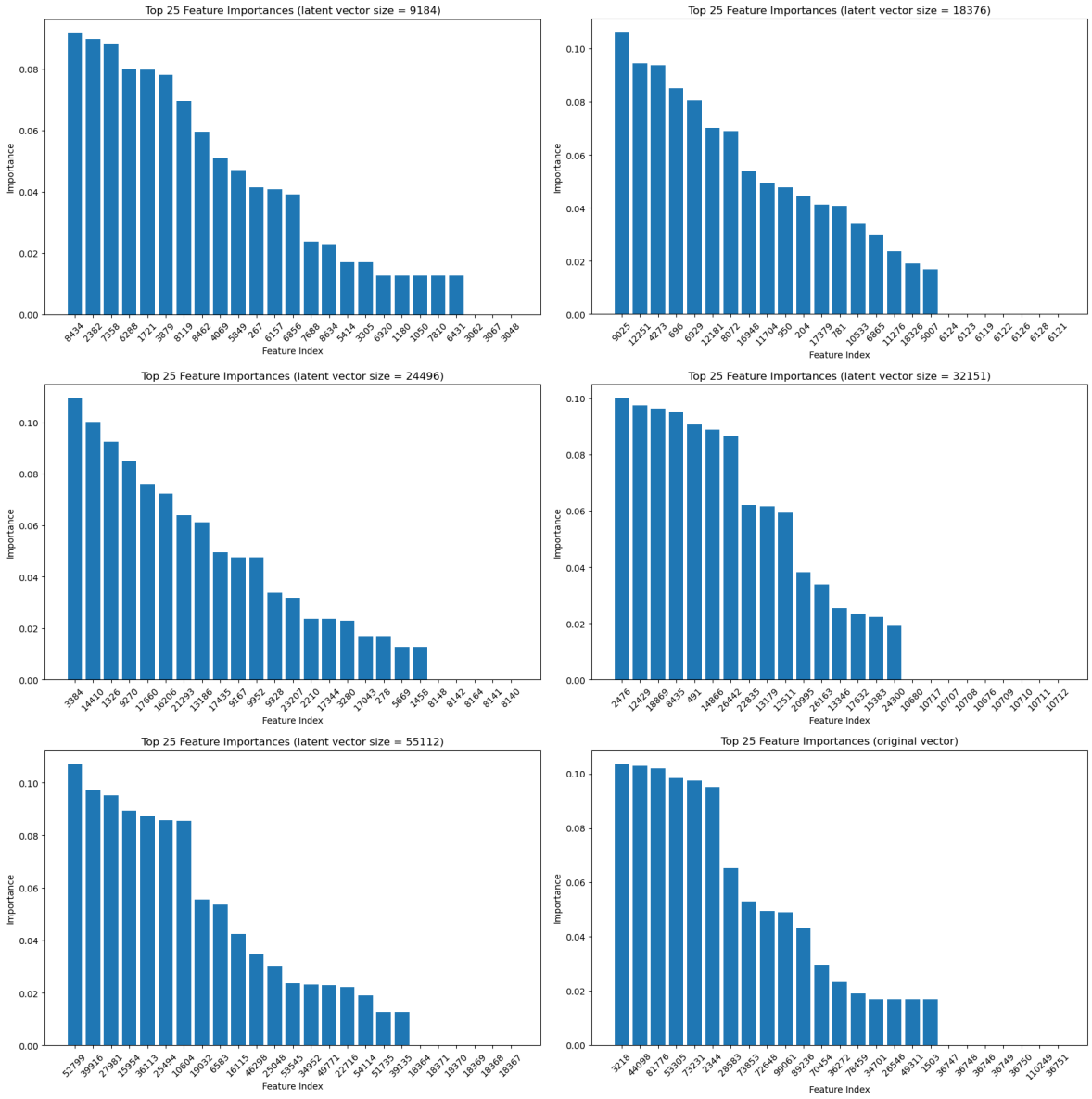

Figura 5: Importancia de atributos a partir de un árbol de decisión para cada tamaño de vector

Al final, no tuvimos que editar el valor de Sample Rate hardcodeado, ya que nuestro nuevo audio funcionaba con el sample rate de 20500 tras hacer el resampling, lo cual nos da mas certeza de que nuestra hipótesis sobre la diferencia del tamaño es correcta.

Vector latente	
2x220500 (Original)	*link*
1x110250 (Original Mono)	*link*
1x55112	*link*
1x32151	*link*
1x24496	*link*
1x18376	*link*
1x9184	*link*

Generación de música nueva

Hemos visto en clase como este tipo de autoencoders que hemos programado, no son lo mejor para generar algo nuevo. Es por eso, que han surgido otros modelos, como pueden ser VAE o GAN. Sin embargo, se pueden probar cosas como generar vectores random del tamaño del espacio latente y pasarlos por el decoder o incluso promediar vectores latentes de otras canciones y decodearlos.

Este tipo de cosas son las que probamos hacer y, si bien lo obtenido no es lo mas satisfactorio, logramos generar audio completamente nuevo a partir de nuestra red.

Promediando vectores latentes de canciones existentes y decodeandolos

Para esto realizamos el forward en testing con la red entrenada, nos guardamos los vectores latentes que quedaban tras el encoding, agarramos los primeros tres y los promediamos. Luego, pasamos esos nuevos vectores latentes por el decoder. Los resultados se pueden escuchar en la siguiente tabla.

Vector latente	
1x55112	*link*
1x32151	*link*
1x24496	*link*
1x18376	*link*
1x9184	*link*

Generando nuevos vectores latentes de manera aleatoria

Para este test, generamos vectores aleatorios del tamaño que quedan los vectores latentes para las distintas redes y pasamos esos vectores por el decoder (entrenado con el dataset original). Logramos ver por qué se nos había dicho que este modelo no era lo mas interesante para generación, ya que obtuvimos simplemente “ruido blanco”. No varió mucho con los distintos modelos, pero de todas formas dejamos los resultados en la siguiente tabla.

Vector latente	
1x55112	*link*
1x32151	*link*
1x24496	*link*

Vector latente	
1x18376	*link*
1x9184	*link*