TP4: Encodeador de música

Tomás Gallo, Guadalupe Molina y Jazmín Sneider

Diciembre 2024

Introducción:

En este informe detallaremos los pasos que seguimos para realizar un autoencoder de audios. Empezaremos describiendo cómo hicimos los diferentes modelos y cuáles fueron los resultados finales de los mismos.

1 Ejercicio 1: Encodear la canción en un vector latente

Para hacer nuestro modelo lo más pequeño posible y que aún conservara las características del audio original fuimos probando de a poco. El modelo comienza procesando un vector de dimensiones [110250]. Sin embargo, para realizar las operaciones de convolución, fue necesario aplicar un "unsqueeze", transformando el vector a dimensiones [1, 110250]. Además, para normalizar los datos durante el entrenamiento, utilizamos Batch Normalization, que ayuda a mantener los valores en rangos óptimos para el funcionamiento de las funciones de activación. Debido a que PyTorch requería que BatchNorm opere sobre tensores tridimensionales, aplicamos un segundo "unsqueeze", ajustando las dimensiones de entrada del modelo a [1, 1, 110250].
Fuimos achicando lentamente el espacio haciendole convoluciones al modelo. Finalmente llegamos a un espacio latente de [1, 16, 2299] como el más pequeño que conserva el ritmo de la música. Nuestro modelo quedó finalmente compuesto por un autoencoder que contiene 3 convoluciones en su decoder, 2 poolings (un average pool y un max pool) y 5 convoluciones transpuestas para regresar al audio a su tamaño original. Para que quede más claro adjuntamos los cálculos realizados:

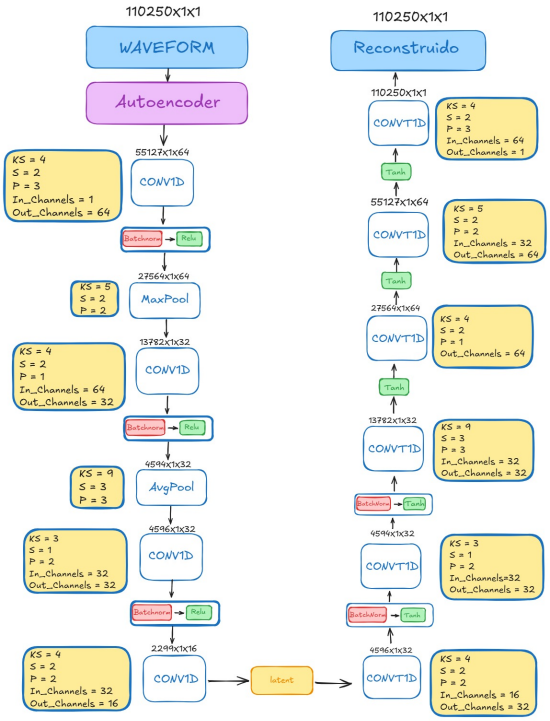


Figure 1: Autoencoder realizado

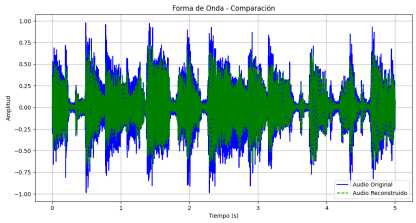
Decisiones del modelo: Para revertir las convoluciones utilizamos convoluciones transpuestas. Sin embargo, para deshacer los poolings descubrimos que teníamos varias opciones:

- **Max Unpooling**: Inicialmente, utilizamos max unpooling, que emplea los índices obtenidos durante el max pooling para reconstruir el tamaño original del tensor. Sin embargo, al escuchar el audio reconstruido, notamos que la calidad era bastante mala.

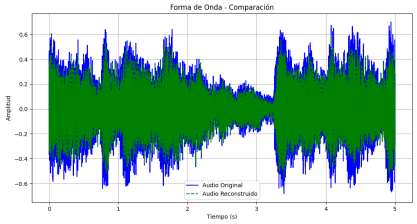
- **Upsampling Lineal**: Luego probamos con nn.Upsample usando el modo 'linear', que realiza una interpolación lineal para aumentar las dimensiones del tensor. Este enfoque es útil cuando se desea aumentar la resolución de las señales unidimensionales sin crear distorsiones significativas, ya que los valores de salida se calculan de manera continua entre los valores de entrada de acuerdo con una función lineal. Esta decisión mejoró significativamente la calidad del audio en comparación con el max unpooling, ya que suavizó la transición entre los valores y evitó grandes distorsiones. Como esta estrategia mejoró la calidad del audio significativamente en comparacion al unpool, decidimos sacar el mismo y dejar upsamplings.

- Finalmente, la estrategia con la que obtuvimos mejores resultados fue con convoluciones traspuestas. Creemos que este resultado se debe a que su uso ayuda al modelo a aprender un proceso de upsampleo en vez de basarse en una ecuación matemática fija, como en los otros casos.

Además de escuchar el audio decidimos graficar la waveform original y la final para compararlas. Estos fueron nuestros resultados:



(a) Audio original de una canción y audio codificado comparados.

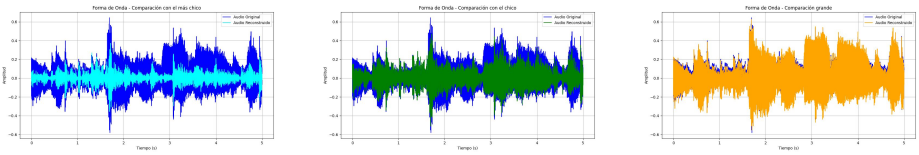


(b) Audio original de otra canción y audio codificado comparados.

Figure 2: Comparación de los audios originales y sus versiones codificadas.

2 Ejercicio 2:

En este ejercicio, creamos dos autoencoders adicionales al modelo del ejercicio 1. El primero de ellos, un modelo de mayor tamaño, consiste en tres capas convolucionales y una capa de max pooling, con un espacio latente de 4596x32x1. Este modelo incluye además cuatro capas de convoluciones transpuestas, que permiten que el audio sea reconstruido hasta su tamaño original.
El segundo autoencoder, diseñado para tener un espacio latente más pequeño, cuenta con seis capas convolucionales y dos capas de pooling, generando un vector latente de 575x16x1. Este modelo utiliza ocho capas de convoluciones transpuestas para reconstruir el audio a su tamaño original. Antes de comenzar con el análisis exploratorio de los vectores latentes, decidimos evaluar el rendimiento de cada autoencoder en relación con el tamaño de su espacio latente, utilizando las canciones del dataset proporcionado. Para ello, seleccionamos canciones aleatorias del conjunto de datos y escuchamos las reconstrucciones de cada modelo. Este paso nos permitió comprender cómo cada autoencoder procesa y reconstruye los datos originales.



(a) Tamaño pequeño **(b)** Tamaño mediano **(c)** Tamaño grande

Figure 3: Comparación de una misma canción procesada por los 3 autoencoders

Nos sorprendió bastante el desempeño de los modelos al escuchar los tres audios reconstruidos. Tanto el autoencoder de mayor tamaño como el de menor tamaño lograron mantener el ritmo de la canción y, en ambos casos, se podían distinguir claramente las voces y la melodía. Sin embargo, al escuchar el audio reconstruido por el autoencoder más pequeño, notamos que la canción casi no se distinguía, lo que inicialmente nos hizo pensar que el modelo no estaba funcionando tan bien. Sin embargo, al imprimir las waveforms, nos sorprendió ver que el autoencoder más chico estaba funcionando mucho mejor de lo que habíamos pensado. La forma de la waveform se mantenía bastante parecida a la original, lo que indicaba que, a pesar de la pérdida perceptible en la escucha, el modelo había logrado preservar la estructura fundamental de la señal.

2.1 Análisis exploratorio de los vectores latentes

2.1.1 Análisis de Componentes Principales (PCA):

Nuestro objetivo fue visualizar cómo se distribuyen los datos en un espacio reducido utilizando Análisis de Componentes Principales (PCA). Dado que los vectores latentes de los modelos están en alta dimensionalidad, realizamos unas transformaciones para visualizarlos en dos dimensiones. Aplanamos los vectores latentes: primero convertimos los tensores de cada modelo, originalmente en formato [B, C, L], a un formato plano [B, C $\times$ L]. Esto permitió tratarlos como matrices en NumPy. Finalmente aplicamos PCA (n_components=2), para poder observar la distribución de los datos en un espacio bidimensional.

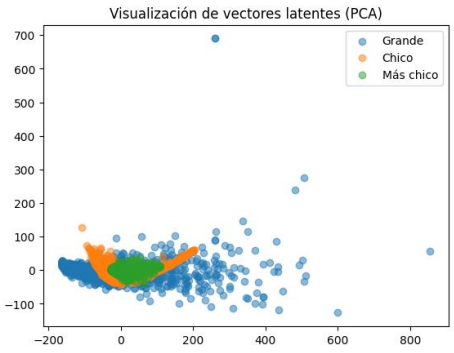


Figure 4: Análisis de Componentes Principales en espacio bidimensional

En la visualización obtenida podemos observar que:

- **Vectores del modelo grande:** Se encuentran representados en color azul. Su distribución ocupa un rango más amplio, lo que podría estar relacionado con su mayor dimensionalidad inicial.
- **Vectores del modelo chico:** Representados en color naranja, parecen tener una dispersión más moderada.
- **Vectores del modelo más chico:** Representados en color verde, muestran una distribución más compacta en comparación con los otros dos modelos.

2.1.2 Clustering

Utilizamos el algoritmo K-Means para explorar la estructura de los vectores latentes generados por los tres autoencoders. El objetivo fue identificar patrones en los datos comprimidos y evaluar cómo cada arquitectura afecta la distribución y agrupamiento de los datos. Para determinar el número óptimo de clusters, aplicamos el método del codo. Nos quedó que los 3 modelos necesitaban 3 clusters. Usamos PCA para reducir las dimensiones de los vectores latentes y visualizamos los clusters generados por K-Means:

- **Modelo Grande:** Tiene una distribución mas dispersa,con clusters bien definidos pero con solapamiento entre algunos puntos. La arquitectura captura una mayor variedad de patrones, lo que podría reflejar una mayor capacidad representacional.
- **Modelo chico:** Los clusters son más compactos en comparación con el modelo grande, pero mantienen buena separación. La estructura es ligeramente menos dispersa.
- **Modelo más chico:** Los clusters son los más compactos y tienen la menor dispersión. Esto refleja una compresión más agresiva del espacio latente, que podría estar sacrificando detalles en favor de la simplicidad.

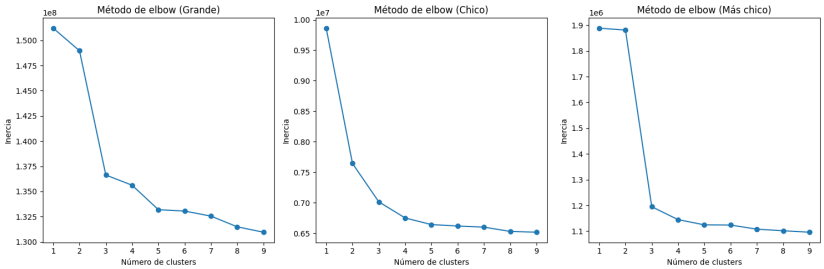


Figure 5: Codo clusters

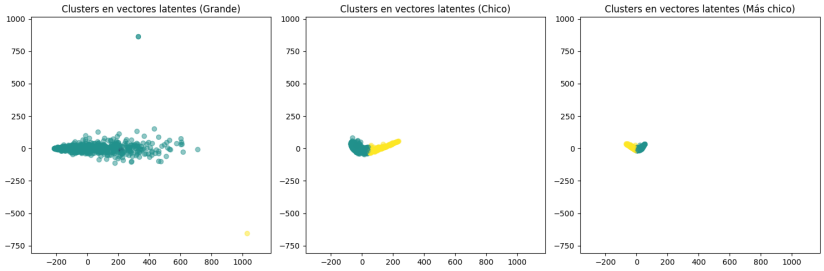


Figure 6: Distribución de clusters

Como conclusión de este análisis vemos que el modelo grande genera clusters más dispersos, posiblemente porque conserva más detalles en sus representaciones latentes. Los modelos chico y más chico producen clusters más compactos, lo que podría facilitar tareas como clasificación, pero podría perder sutilezas (quizás importantes) en los datos.

Métrica extra

Para poder evaluar la calidad de los clusters que se estaban generando investigamos otras posibles métricas. Encontramos la **Silhouette Score** que permite evaluar la calidad de un clustering considerando dos factores clave: cohesión (qué tan cerca están los puntos de un cluster con respecto a su centroide) y separación (qué tan lejos están los puntos de un cluster de los centroides de los demás clusters). Aplicándola sobre los resultados de los 3 autoencoders obtuvimos que:

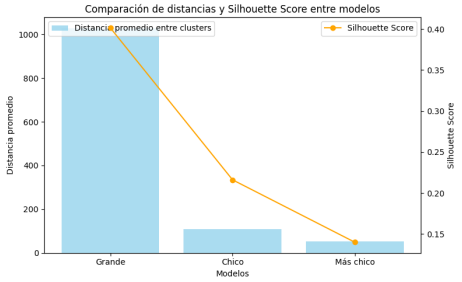


Figure 7: Métrica extra: Silhoutte Score

- **Modelo Grande:** Tiene la mayor distancia promedio entre clusters (1000). Esto indica que los centros de los clusters están muy separados entre sí. El Silhouette Score (0.40) es también el más alto, lo que sugiere una buena calidad de los clusters.
- **Modelo Chico:** La distancia promedio entre clusters disminuye significativamente en comparación con el modelo "Grande", lo que indica clusters

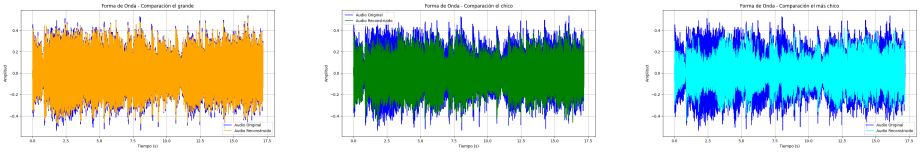
más cercanos entre sí. El Silhouette Score también baja (0.25), lo que refleja una pérdida de calidad en los clusters.

- **Modelo Más Chico:** La distancia promedio entre clusters es la más baja (200), indicando que los clusters están muy cerca entre sí. El Silhouette Score (0.15) es el más bajo, lo que sugiere que los clusters se solapan o no están bien definidos.

Concluimos que el Modelo Grande muestra la mejor separación entre clusters, aunque su cohesión puede mejorar. El Modelo más Chico presenta la peor calidad de clustering, probablemente debido a una representación latente limitada. Por otro lado, el Modelo Chico funciona un poco mejor, presentándose como una buena alternativa intermedia.

3 Ejercicio 3: Arquitectura CNN

Seleccionamos algunas canciones que nos gustaban: como "Arde la Ciudad", "La Bestia Pop", "Vámonos de Viaje" y "Emotihadas", para evaluar el desempeño de los autoencoders que habíamos implementado previamente. Al procesar estas canciones, observamos que los resultados eran aún mejores que aquellos obtenidos con el dataset proporcionado. Además, realizamos pruebas utilizando autoencoders de diferentes tamaños del ejercicio 2, comprobando su capacidad para capturar y reconstruir las características esenciales del audio con gran efectividad hasta con el modelo con el espacio latente más pequeño. Ploteamos los waveforms reconstruidos puestos encima de los originales para poder analizarlos y compararlos con mayor claridad.



(a) Autoencoder grande **(b)** Autoencoder original **(c)** Autoencoder chico

Figure 8: Comparación de reconstrucciones con diferentes tamaños de autoencoder.

Observamos que claramente hasta el más pequeño representa bien la cancion original.

4 Ejercicio 4: Generación de música

En este ejercicio exploramos cómo el modelo de VAE puede ser utilizado para la generación y modificación de música, enfocándonos en la manipulación del espacio latente. Realizamos tres experimentos clave: interpolación entre canciones,

modificación de características del espacio latente y generación de música a partir de un vector aleatorio. Los resultados de cada experimento se presentan a continuación.

Interpolación de vectores latentes

En este ejercicio, se realiza una interpolación lineal entre dos vectores latentes generados por un autoencoder variacional (VAE). La interpolación se define de la siguiente forma:

$$\mathbf{z_{interp}}(\alpha) = (1 - \alpha) \cdot \mathbf{z_1} + \alpha \cdot \mathbf{z_2}, \quad \alpha \in [0, 1]$$

donde $\mathbf{z_1}$ y $\mathbf{z_2}$ son los dos vectores latentes originales, y α es un parámetro de interpolación que varía entre 0 y 1. Cuando $\alpha = 0$, el vector latente interpolado es igual a $\mathbf{z_1}$, y cuando $\alpha = 1$, el vector latente interpolado es igual a $\mathbf{z_2}$.

4.1 Generación de interpolaciones

Para generar interpolaciones entre los vectores latentes, primero se seleccionan dos canciones. Los vectores latentes correspondientes a estas canciones se interpolan usando la fórmula anterior, variando α en un rango de 0 a 1. El número de pasos en la interpolación depende de cuántas variantes queramos generar. En este caso, se realizaron 5 variantes de la interpolación.

4.1.1 Resultados y análisis

Al decodificar los vectores latentes interpolados, se observa una transición gradual entre las dos canciones originales, lo que demuestra que el modelo puede combinar las características de ambos audios. Este tipo de interpolación es útil para generar mezclas o "remixes" de dos canciones, lo que resulta en la creación de música nueva y única.
El inicio y el final del gráfico representa los audios originales. Podemos ver cómo en la interpolación se genera una waveform que contiene una mezcla entre ambas canciones.

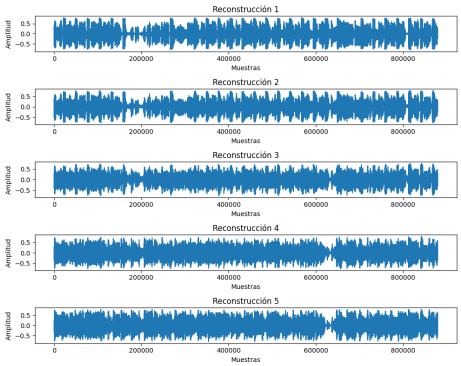


Figure 9: Resultado de las interpolaciones

4.2 Modificación de Características del Espacio Latente

En este experimento, tomamos el espacio latente de una canción específica y modificamos algunas de sus dimensiones asignándoles valores aleatorios entre -0.75 y 0.75. La reconstrucción del modelo resultante presentaba la canción original con un ruido de fondo superpuesto.
Esto sugiere que, aunque el modelo conserva la estructura general de la canción, las modificaciones en el espacio latente afectan el resultado de forma audible. Si bien el resultado no puede considerarse música completamente nueva, demuestra que pequeñas perturbaciones en el espacio latente pueden generar variantes de un audio original.

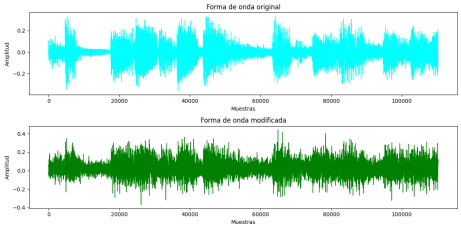


Figure 10: Enter Caption

Conclusión: Este proceso puede ser utilizado para introducir modificaciones creativas en una canción sin perder su esencia original.

4.3 Generación de Música con un Espacio Latente Aleatorio

En este experimento, generamos un vector latente aleatorio con valores entre -1 y 1, que fue decodificado directamente por el modelo. El resultado fue un audio que consistía en ruido, sin patrones musicales reconocibles. Esto sugiere que,

para producir música, el espacio latente necesita estar estructurado de acuerdo con la distribución aprendida por el modelo durante el entrenamiento.
Conclusión: Un vector aleatorio fuera de la distribución aprendida no genera música, pero evidencia que el modelo sigue reconstruyendo entradas desconocidas basándose en lo que aprendió, resultando en ruido estructurado.

4.4 Conclusión General

Todos los experimentos demuestran que el espacio latente del modelo tiene un gran potencial para la manipulación creativa de audio: la interpolación entre vectores latentes genera transiciones coherentes y mezclas interesantes entre canciones, logrando una especie de remix.
Las modificaciones controladas del espacio latente pueden alterar una canción original de manera predecible, permitiendo introducir efectos creativos.
La generación de audio a partir de un vector aleatorio, aunque no produce música propiamente dicha, resalta la importancia de una distribución latente bien estructurada. En conjunto, estos resultados muestran que un modelo VAE bien entrenado no solo puede reconstruir canciones existentes, sino también generar contenido nuevo y creativo basado en la manipulación de su espacio latente. Esto abre las puertas a nuevas aplicaciones en la generación de música y audio.

5 Ejercicio 5: Implementación de VAE

Para este ejercicio decidimos crear un VAE entrenandolo con las canciones del ejercicio 1. Desafortunadamente no pudimos conseguir un buen resultado dado que si bien hicimos la fórmula de la loss de VAE mostrada a continuación, la misma llegó a los 6.000 y no logramos hacerla disminuir por lo que no pudimos solucionarlo.

5.1 VAE Loss

La función de pérdida (*VAE Loss*) está compuesta por dos términos principales: el error de reconstrucción y la divergencia Kullback-Leibler (KL). Se define de la siguiente manera:

$$\mathcal{L} = \underbrace{MSE(x, \hat{x})}_{Errordereconstrucción} + \underbrace{-\frac{1}{2}\sum\left(1 + \log(\sigma^2) - \mu^2 - \sigma^2\right)}_{DivergenciaKL}$$

Donde:

- x: El dato original.
- $\hat{x}$: El dato reconstruido por el modelo.
- μ: El vector de medias del espacio latente.

- $\sigma^2 = \exp(logvar)$: La varianza del espacio latente, obtenida a partir del logaritmo de la varianza (`logvar`).

Detalles:

1. **Error de reconstrucción:**

$$MSE(x, \hat{x}) = \sum (x - \hat{x})^2$$

Representa cuánto difieren los datos reconstruidos de los datos originales.

2. **Divergencia KL:**

$$KL = -\frac{1}{2} \sum \left(1 + \log(\sigma^2) - \mu^2 - \sigma^2\right)$$

Penaliza la diferencia entre la distribución latente generada por el modelo y la distribución gaussiana estándar $\mathcal{N}(0, I)$.
Por lo tanto, la pérdida total es la suma de estos dos términos:

$$\mathcal{L} = Reconstrucción + KL$$

La loss durante todo el entrenamiento nos dio bastante alta y el resultado del audio final fue ruido, no música con algun tipo de ritmo, pero en comparación al experimento de ponerle un input random si que mejora, es un ruido más suave.

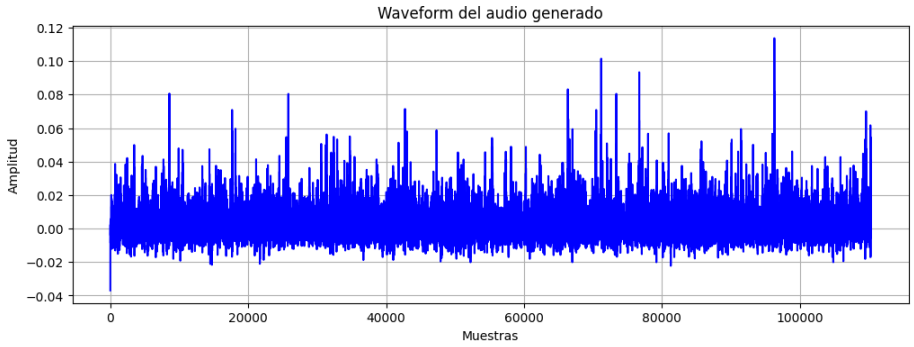


Figure 11: Audio generado

Conclusión general del trabajo

Este trabajo fue una experiencia muy interesante que nos permitió entender cómo los autoencoders, en general, son herramientas poderosas para capturar las características más importantes de los datos. Experimentamos de manera práctica cómo estas arquitecturas son capaces de reducir datos complejos a representaciones compactas y eficientes en un espacio latente, sin perder su esencia.

Poner en práctica los conceptos teóricos nos ayudó a comprender cómo los autoencoders pueden preservar información relevante incluso cuando trabajamos con dimensiones latentes reducidas. Además, exploramos cómo esta representación comprimida facilita tareas como la generación de datos, la interpolación entre ejemplos y la modificación de características específicas de los datos originales.

En comparación con arquitecturas tradicionales como las *fully connected*, los autoencoders destacan por su capacidad de generalizar y conservar patrones significativos, incluso con recursos limitados. Esto los convierte en herramientas muy útiles para una amplia variedad de aplicaciones en campos como la generación de contenido, la compresión de datos y la detección de anomalías.

En conclusión, este proyecto nos permitió no solo afianzar conocimientos teóricos, sino también valorar la versatilidad y el potencial de los autoencoders para resolver problemas complejos en el procesamiento de datos.