UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería de Telecomunicación

Identificación de patrones de señales musicales usando inteligencia artificial.

Trabajo Fin de Grado

Grado en Tecnología Digital y Multimedia

AUTOR/A: Soriano Gallego, Pedro

Tutor/a: Ferrer Contreras, Miguel

CURSO ACADÉMICO: 2023/2024

Resumen

La identificación de patrones de señales musicales es la base para identificar la correspondencia de un determinado extracto musical con alguno de los contenidos incluidos dentro de una base de datos. Conocidas aplicaciones como Shazam o Soundhound realizan esta operación para identificar temas musicales a partir de cortos fragmentos de los mismos.

Los patrones o huellas espectrales habitualmente usados se basan en la detección de puntos relevantes en el espectrograma (generalmente relacionados con máximos de energía) que contienen cierta singularidad que permite diferenciarlos del resto. Algunas de estas estrategias no son robustas cuando, más allá del problema de identificación, se pretende resolver un problema de emparejamiento de audio (identificación de los temas de audio que guardan alguna correspondencia con el fragmento en cuestión, independientemente de si no contiene exactamente la misma versión, ritmo, tono, o simplemente hay ruido en el ambiente o es una versión simplificada como puede ser un extracto silbado o tatareado).

En este trabajo se pretende explorar la capacidad del empleo de técnicas de aprendizaje automático basado en el manejo de datos (Machine Learning), como un ejemplo de la aplicación de las técnicas de inteligencia artificial para resolver el problema de identificación y emparejamiento de patrones de audio. En concreto, se analizará la aptitud de los clasificadores basados en redes neuronales para realizar esta tarea. Para ello se partirá de una base de datos reducida, que aplicando técnicas de aumento de datos dotará a la red robustez para realizar correctamente la tarea de emparejamiento además de la de identificación.

Palabras Clave:

Aprendizaje automático; emparejamiento de audio; clasificación; aumento de datos; extracción de melodías; sonificación; desplazamiento de pitch; modificación del tempo.

Abstract:

The identification of patterns of musical signals is the basis for identifying the matching of a certain musical excerpt with any of the contents included within a database. Well-known applications such as Shazam or Soundhound perform this operation to identify musical themes from short fragments of them.

The commonly used spectral patterns or fingerprints are based on the detection of relevant points in the spectrogram (generally related to energy maxims) that contain a certain singularity that allows them to be differentiated from the rest. Some of these strategies are not robust when, beyond the identification problem, the aim is to solve an audio matching problem (identification of audio file that have some correspondence with the fragment in question, regardless of whether they do not contain exactly the same version, rhythm, tone, or there is some noise in the environment or it is a simplified version such as a whistled or hummed extract).

This work aims to explore the capacity of using machine learning techniques, as an example of the application of artificial intelligence techniques to solve the problem of identifying and matching audio patterns. Specifically, the ability of classifiers based on neural networks to perform this task will be analyzed. To do this, a reduced database jointly with data augmentation techniques will be used, providing the network with robustness to correctly perform the matching task in addition to the identification task.

Keywords:

Machine learning; audio matching; data augmentation; classification; extraction of musical melodies; sonification; pitch shifting, time stretching.

RESUMEN EJECUTIVO

La memoria del TFG del Grado en Tecnología Digital y Multimedia debe desarrollar en el texto los siguientes conceptos, debidamente justificados y discutidos, centrados en el ámbito de las tecnologías digitales y multimedia.

CONCEPT (ABET)	**CONCEPTO (traducción)**	**¿Cumple? (S/N)**	**¿Dónde? (páginas)**
1. IDENTIFY:	1. IDENTIFICAR:		
1.1. Problem statement and opportunity	1.1. Planteamiento del problema y oportunidad	S	1
1.2. Constraints (standards, codes, needs, requirements & specifications)	1.2. Toma en consideración de los condicionantes (normas técnicas y regulación, necesidades, requisitos y especificaciones)	S	1
1.3. Setting of goals	1.3. Establecimiento de objetivos	S	2
2. FORMULATE:	2. FORMULAR:		
2.1. Creative solution generation (analysis)	2.1. Generación de soluciones creativas (análisis)	S	12-17
2.2. Evaluation of multiple solutions and decisionmaking (synthesis)	2.2. Evaluación de múltiples soluciones y toma de decisiones (síntesis)	S	21-32
3. SOLVE:	3. RESOLVER:		
3.1. Fulfilment of goals	3.1. Evaluación del cumplimiento de objetivos	S	32-37
3.2. Overall impact and significance (contributions and practical recommendations)	3.2. Evaluación del impacto global y alcance (contribuciones y recomendaciones prácticas)	S	37-38

Índice General

Índice de Figuras.

1. Introducción.

1.1. Justificación.

En la era digital actual, la música y el sonido son componentes fundamentales de la experiencia cotidiana de millones de personas. Con el avance de la tecnología, la capacidad de analizar, procesar e identificar señales de audio ha cobrado una importancia creciente. Herramientas como Shazam, que permiten identificar una canción a partir de un breve fragmento, han revolucionado la forma en que interactuamos con la música, brindando soluciones rápidas y precisas a un problema complejo: identificar patrones en señales de audio.

Shazam, y otras aplicaciones similares, basan su funcionamiento en la creación de huellas espectrales de las canciones, permitiendo la comparación de fragmentos de audio con una base de datos ya existente. Sin embargo, estos métodos, aunque efectivos, pueden enfrentar limitaciones cuando se trata de identificar versiones alteradas de una canción, tales como variaciones en el ritmo, tono o calidad de audio.

Frente a estas limitaciones, las técnicas de Machine Learning ofrecen un enfoque alternativo y potencialmente más robusto para la identificación y emparejamiento de señales de audio. Estas técnicas permiten a los sistemas aprender y mejorar a partir de datos.

Asimismo, en la selección y manipulación de los datos utilizados, se ha prestado especial atención al cumplimiento del Reglamento General de Protección de Datos (RGPD) de la Unión Europea. Este marco legal es fundamental para proteger la privacidad de los usuarios y garantizar que cualquier procesamiento de datos personales, como el análisis de fragmentos de audio que podrían incluir voces humanas, se realice de manera ética y conforme a la ley [20]. Por ejemplo, en este proyecto se ha asegurado que todos los datos utilizados provengan de fuentes públicas o se hayan anonimizado adecuadamente para evitar la identificación de personas.

Otra consideración importante ha sido la adhesión a las recomendaciones técnicas establecidas por la IEEE. Si bien la norma IEEE 1857.4 [21] está enfocada en la codificación eficiente de imágenes y no directamente en señales de audio, en este proyecto se han considerado principios y directrices adecuadas para el procesamiento de señales de audio. Estas guías son esenciales para el diseño de sistemas que deben manejar grandes volúmenes de datos de audio de manera eficiente, asegurando tanto la calidad del procesamiento como la interoperabilidad de los sistemas desarrollados.

El presente trabajo no solo busca avanzar en el campo de la identificación y emparejamiento de fragmentos de audio mediante técnicas de Machine Learning, sino que también contribuye a los Objetivos de Desarrollo Sostenible (ODS) de las Naciones Unidas [22]. Específicamente, este proyecto se alinea con el ODS 9, 'Industria, Innovación e Infraestructura', al fomentar la innovación en el procesamiento de señales de audio y la creación de tecnologías más accesibles y robustas. Además, el cumplimiento de normativas como el RGPD, que garantiza la privacidad

y seguridad de los datos, está en línea con el ODS 16, 'Paz, Justicia e Instituciones Sólidas', que promueve sociedades justas y transparentes.

1.2. Objetivo.

Este trabajo se centra en la exploración de las capacidades de algunas estrategias de Machine Learning, como las de las redes neuronales o algoritmo de vecinos cercanos para la identificación y emparejamiento de fragmentos de audio. A diferencia de los enfoques tradicionales que se basan en la creación de huellas digitales específicas, este estudio propone un modelo de aprendizaje profundo que puede generalizar mejor a variaciones en el audio, proporcionando una mayor precisión y flexibilidad en la identificación de canciones incluso en condiciones adversas.

Para llevarlo a cabo, se ha diseñado un experimento utilizando una base de datos reducida de canciones, la cual se ha segmentado en fragmentos. A partir de estos fragmentos, se han extraído características representativas mediante la utilización de coeficientes cepstrales en la frecuencia de Mel (MFCC). Estas características han sido utilizadas para entrenar el algoritmo de vecinos más cercanos (KNN). En fases posteriores del estudio, se pretende realizar una comparativa entre el uso de MFCC y espectrogramas de Mel para determinar cuál de estos enfoques es más efectivo para la tarea de identificación y emparejamiento de fragmentos de audio, al igual que comparar entre el modelo de red neuronal convolucional y el algoritmo de vecinos más cercanos.

2. Fundamentos Teóricos.

En esta sección se abordarán los conceptos teóricos esenciales que sustentan el desarrollo del proyecto, comenzando con una introducción al Machine Learning, sus aplicaciones y tipos, y luego profundizando en las arquitecturas de redes neuronales más relevantes para el problema de identificación y emparejamiento de patrones de audio. Además, se introducirá el uso de espectrogramas como una técnica alternativa a los MFCCs y se discutirá su aplicabilidad en este contexto.

2.1. Introducción al Machine Learning.

El Machine Learning o aprendizaje automático es un campo de la inteligencia artificial que se centra en el desarrollo de algoritmos y técnicas que permiten a las máquinas mejorar su rendimiento en tareas específicas a través de la experiencia. A diferencia de los sistemas tradicionales de programación, donde las reglas y las decisiones se codifican explícitamente, en el aprendizaje automático las máquinas utilizan datos para "aprender" patrones y tomar decisiones basadas en estos.

El Machine Learning ha revolucionado numerosos campos como el reconocimiento de voz, la visión por computadora, la medicina, las finanzas, y en el caso que nos ocupa, el procesamiento de señales de audio. En aplicaciones como el reconocimiento de canciones o la identificación de patrones de audio, el ML se emplea para reconocer estructuras y características en fragmentos de audio que permiten su clasificación o emparejamiento con otros fragmentos en una base de datos.

2.2. Tipos de Predictores.

En el campo del procesamiento de señales de audio, la selección de predictores o características adecuadas es fundamental para el rendimiento de los modelos de clasificación. Dos de los enfoques más comunes en la extracción de características de audio son los coeficientes cepstrales en la frecuencia de Mel (MFCC) y los espectrogramas. Este apartado aborda la teoría detrás de cada uno de estos métodos, su relevancia en el contexto del procesamiento de señales de audio, y la justificación de la elección de los espectrogramas como predictor principal en este trabajo.

2.2.1. Coeficientes Cepstrales en la Frecuencia de Mel (MFCC)

Los coeficientes cepstrales en la frecuencia de Mel (MFCC) son una representación frecuentemente utilizada en tareas de reconocimiento de voz y procesamiento de audio debido a su capacidad para captar las características perceptuales de los sonidos [1]. El cálculo de los MFCC se basa en la escala de Mel, que es una escala perceptual de frecuencias diseñada para aproximarse a la forma en que los humanos perciben las diferencias de tono.

El proceso para calcular los MFCCs implica varios pasos:

1. Preénfasis: Se aplica un filtro para acentuar las altas frecuencias, compensando la menor energía presente en estas bandas debido a la atenuación natural en la producción del sonido.

2. Segmentación en Ventanas: La señal de audio se divide en segmentos cortos o ventanas (típicamente de 20 a 40 ms) para capturar la estacionariedad local de la señal.

3. Transformada Discreta de Fourier (DFT): Se aplica la DFT a cada segmento (ventana) para convertir la señal del dominio del tiempo al dominio de la frecuencia.

4. Escalado en Mel: Se mapea la frecuencia lineal obtenida de cada DFT a la escala de Mel, utilizando un conjunto de filtros triangulares superpuestos.

5. Logaritmo de la Energía: Se toma el logaritmo de la energía de cada filtro, lo cual emula la percepción humana que es logarítmica en naturaleza.

6. Transformada Discreta del Coseno (DCT): Finalmente, se aplica la DCT para obtener los coeficientes cepstrales, los cuales representan la envolvente del espectro en la escala de Mel.

Los MFCCs son eficaces porque simplifican la información frecuencial del audio en un conjunto compacto de características que son robustas frente a variaciones en el tono y el ruido, lo cual es esencial en aplicaciones como el reconocimiento de voz y la identificación de patrones acústicos.

2.2.2. Espectrogramas de Mel.

Un espectrograma de Mel es una representación visual que muestra cómo varía la energía de las distintas frecuencias de una señal a lo largo del tiempo, utilizando una escala logarítmica (Mel) que imita la percepción humana del tono [2]. A diferencia del espectrograma estándar, el espectrograma de Mel comprime las frecuencias en una escala que es más densa en las frecuencias bajas y más dispersa en las altas, reflejando mejor cómo los humanos perciben el sonido.
El proceso para generar un espectrograma de Mel incluye:

1. **Segmentación en Ventanas:** Al igual que con otros métodos de análisis espectral, la señal se divide en ventanas temporales.

2. **DFT** Se aplica la Transformada Discreta de Fourier (DFT) a cada ventana para obtener un espectro de frecuencias, calculando el módulo de esta transformación para cada ventana.

3. **Conversión a Escala Mel:** Las frecuencias obtenidas se convierten a la escala de Mel, ajustando la resolución frecuencial para alinearse con la percepción auditiva humana. La fórmula de conversión es $\text{Mel(f)} = 1127.01048 \cdot \mathbf{ln}(1 + \frac{f}{700})$

Los espectrogramas ofrecen una visión detallada tanto de la estructura temporal como frecuencial de la señal, lo que los hace especialmente útiles para analizar señales complejas o no estacionarias. En particular, capturan la evolución de las frecuencias a lo largo del tiempo, lo cual es crucial en

tareas donde la temporalidad de los eventos acústicos es significativa, como en el reconocimiento de música y patrones sonoros complejos.

2.2.3. Comparativa entre MFCCs y Espectrogramas de Mel.

Ambos métodos, MFCCs y espectrogramas, tienen sus propias ventajas y limitaciones, y la elección entre ellos depende en gran medida del tipo de tarea y las características de la señal de audio.

Ventajas de los MFCCs:

- Compactación de Información: Los MFCCs reducen la dimensionalidad de la señal de audio, lo que facilita el entrenamiento de modelos de clasificación al proporcionar un conjunto más pequeño y manejable de características.
- Modelado Perceptual: Dado que están diseñados para aproximar la percepción auditiva humana, los MFCCs son especialmente útiles en tareas que requieren la emulación de cómo los humanos perciben el sonido, como el reconocimiento de voz.
- Robustez al Ruido: Los MFCCs son relativamente robustos frente a variaciones en el entorno acústico, como ruido o cambios en el tono, lo que los hace ideales para aplicaciones en ambientes ruidosos.

Ventajas de los Espectrogramas:

- Riqueza de Información: A diferencia de los MFCCs, los espectrogramas conservan una mayor cantidad de detalles de la señal original, lo que es crucial para tareas que requieren una alta precisión en la identificación de patrones específicos de frecuencia y temporalidad.
- Análisis Temporal: Los espectrogramas permiten el análisis de cómo cambian las frecuencias a lo largo del tiempo, lo que es esencial en aplicaciones donde la dinámica temporal del sonido es relevante.
- Flexibilidad: Los espectrogramas pueden adaptarse mejor a tareas de identificación de audio en condiciones de gran variabilidad tonal o cuando se trabaja con versiones muy alteradas de una señal.

2.2.4. Justificación de la Elección de MFCCs.

En este proyecto, se ha optado por utilizar los **Coeficientes Cepstrales en la Frecuencia de Mel (MFCCs)** como el método principal de extracción de características, en combinación con el algoritmo **K-Nearest Neighbors (KNN)**, debido a varias razones clave. Los MFCCs son una representación compacta y robusta que simplifica la información frecuencial del audio, permitiendo capturar las características perceptuales más relevantes de la señal. Esto es particularmente útil en el contexto del reconocimiento de patrones sonoros.

El uso de **KNN** junto con **MFCCs** resulta especialmente ventajoso por los siguientes motivos:

- **Compactación y eficiencia**: Los MFCCs reducen la dimensionalidad de la señal, lo que facilita que KNN procese las características de forma eficiente, minimizando el impacto negativo de una alta dimensionalidad.
- **Similitud perceptual**: La combinación de KNN con MFCCs permite un análisis basado en la proximidad de las características perceptuales del audio, imitando cómo los humanos perciben el sonido, lo que es crucial en tareas como la clasificación de patrones acústicos.
- **Robustez frente al ruido**: Los MFCCs proporcionan una representación robusta, incluso en condiciones de ruido o variabilidad en el tono, lo que mejora la precisión de KNN al clasificar señales de audio en entornos acústicamente adversos.

La elección de esta combinación metodológica ha demostrado ser eficaz en la clasificación de patrones de audio, gracias a la capacidad de KNN para identificar patrones similares dentro del espacio de características proporcionado por los MFCCs.

2.3. Tipos de Aprendizaje Automático.

Existen varios enfoques dentro del aprendizaje automático, cada uno adecuado para diferentes tipos de problemas [3]. A continuación, se describen los principales tipos de aprendizaje automático:

2.3.1. Aprendizaje Supervisado.

El aprendizaje supervisado es el tipo más común de aprendizaje automático. En este enfoque, el modelo se entrena utilizando un conjunto de datos etiquetados, donde cada ejemplo de entrenamiento viene acompañado de una etiqueta o resultado deseado. El objetivo del modelo es aprender una función que, a partir de las características de entrada, prediga correctamente las etiquetas correspondientes. Este tipo de aprendizaje es especialmente útil en problemas de clasificación y regresión.

En el contexto del proyecto, el aprendizaje supervisado se utiliza para entrenar una red neuronal que, dada una representación de características de un fragmento de audio (como MFCCs o espectrogramas), prediga a qué canción pertenece dicho fragmento.

2.3.2. Aprendizaje No Supervisado.

En el aprendizaje no supervisado, el modelo se entrena con datos que no están etiquetados. El objetivo es descubrir estructuras ocultas o patrones en los datos. Los algoritmos de clustering

(agrupamiento) y reducción de dimensionalidad son ejemplos comunes de aprendizaje no supervisado.

Aunque este tipo de aprendizaje no se ha aplicado directamente en este proyecto, es importante mencionar que podría ser útil en futuras investigaciones, por ejemplo, para agrupar canciones similares sin necesidad de etiquetas o para reducir la complejidad de los datos de entrada mediante técnicas como el Análisis de Componentes Principales (PCA).

2.3.3. Aprendizaje Semi-Supervisado.

El aprendizaje semi-supervisado es un enfoque híbrido que utiliza una pequeña cantidad de datos etiquetados junto con una gran cantidad de datos no etiquetados. Este método es útil cuando la obtención de etiquetas es costosa o difícil, pero se dispone de muchos datos sin etiquetar.

En este proyecto, el aprendizaje semi-supervisado no se ha utilizado, pero es un área prometedora para mejorar la generalización de los modelos, especialmente en escenarios donde las canciones etiquetadas son escasas.

2.3.4. Aprendizaje por Refuerzo.

El aprendizaje por refuerzo se basa en un sistema de recompensas y castigos para que un agente aprenda a tomar decisiones en un entorno. A diferencia de los otros tipos de aprendizaje, en el aprendizaje por refuerzo, el modelo aprende mediante la interacción con el entorno, tomando acciones y recibiendo retroalimentación en forma de recompensas.

Este enfoque es común en problemas de control y optimización, pero no es directamente aplicable al problema de identificación de audio que se aborda en este proyecto. Sin embargo, el aprendizaje por refuerzo podría explorarse en contextos donde el sistema debe adaptarse dinámicamente a nuevas canciones o estilos musicales.

2.4. Redes Neuronales.

Las redes neuronales son una clase de modelos en el aprendizaje automático inspirados en la estructura y el funcionamiento del cerebro humano. Estas redes consisten en capas de nodos o "neuronas" interconectadas, donde cada conexión tiene un peso ajustable que se aprende durante el proceso de entrenamiento. Las redes neuronales son particularmente eficaces para detectar patrones complejos en los datos, lo que las hace adecuadas para tareas de clasificación, reconocimiento y predicción.

A continuación, se describen las arquitecturas de redes neuronales más relevantes para este proyecto:

2.4.1. Perceptrón y Perceptrón Multicapa (MLP).

El perceptrón es el modelo más simple de red neuronal y se basa en una única capa de neuronas. Cada neurona realiza una combinación lineal de las entradas y aplica una función de activación para producir una salida. Sin embargo, el perceptrón simple es limitado en su capacidad de resolver problemas complejos.

El perceptrón multicapa (MLP) mejora este modelo al añadir capas ocultas entre la entrada y la salida, permitiendo que la red aprenda relaciones no lineales complejas siempre que se usen funciones de activación no lineales. Las MLPs son efectivas para problemas de clasificación de datos estructurados, pero no son ideales para trabajar con datos que tienen una estructura espacial o temporal, como las señales de audio.

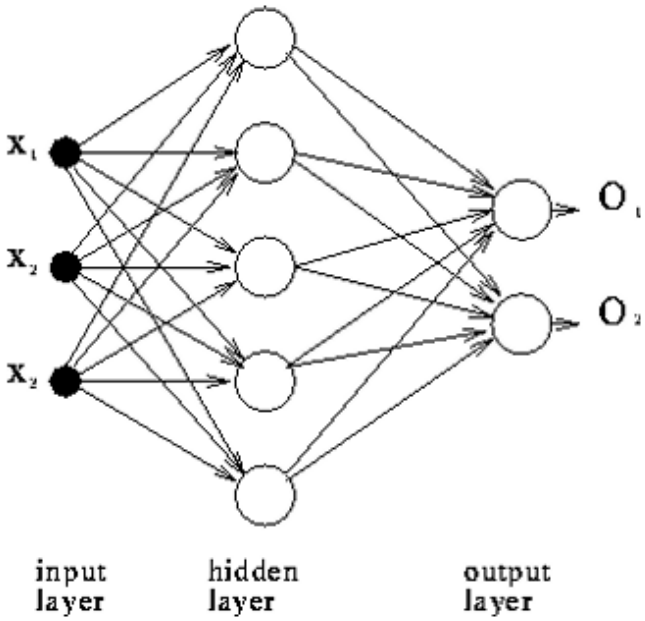

Ilustración 1. Red Multicapa [4]

2.4.2. Redes Neuronales Convolucionales (CNN)

Las redes neuronales convolucionales (CNN) son una arquitectura especialmente diseñada para procesar datos con una estructura en mallado, como imágenes y, en este caso, representaciones de audio como espectrogramas. Las CNNs utilizan capas convolucionales que aplican filtros o kernels para detectar características locales en los datos, como bordes en imágenes o patrones de frecuencia en señales de audio.

Las CNNs son particularmente adecuadas para la tarea de identificación de patrones en audio porque pueden capturar características espaciales en los espectrogramas o representaciones de MFCCs, permitiendo una clasificación más precisa.

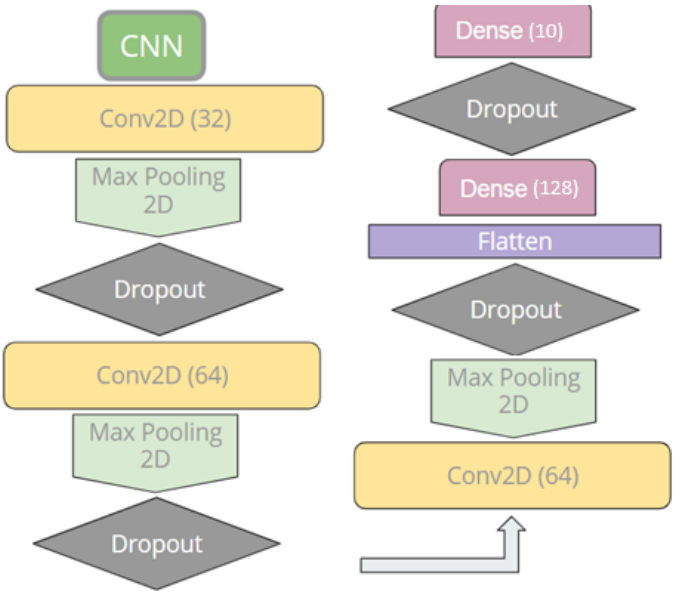

Ilustración 2. Red Neuronal Convolucional.

2.4.3. Redes Neuronales Recurrentes (RNN)

Las redes neuronales recurrentes (RNN) están diseñadas para manejar datos secuenciales, como series temporales o señales de audio. A diferencia de las CNNs, las RNNs tienen conexiones recurrentes que les permiten mantener un "estado" o memoria de las entradas anteriores, lo que es útil para procesar datos donde el orden importa.

Aunque las RNNs pueden ser útiles para el análisis de audio en tiempo real o para tareas que requieren un modelado secuencial, su aplicación directa en este proyecto fue limitada debido a la naturaleza de la tarea, que se enfocó más en la clasificación de patrones espaciales en los MFCCs y espectrogramas que en la secuencialidad de las señales.

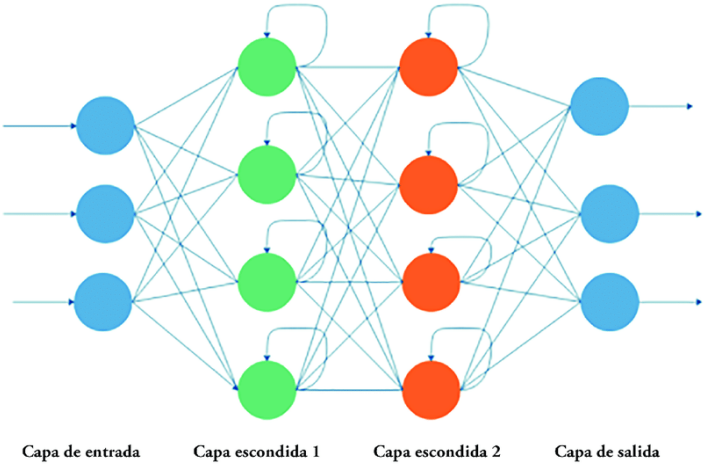

Ilustración 3. Red Neuronal Recurrente [5].

2.4.4. Algoritmo de Vecinos Más Cercanos KNN.

El modelo de K-Nearest Neighbors (KNN) es un algoritmo de clasificación que asigna una clase a un nuevo punto de datos basándose en las clases de sus k vecinos más cercanos en el espacio de características. A diferencia de las redes neuronales, KNN no requiere un proceso de entrenamiento previo; en su lugar, almacena todo el conjunto de datos de entrenamiento y calcula las distancias en tiempo real para clasificar nuevos datos.

KNN es especialmente útil en el procesamiento de audio cuando se trabaja con características como los Coeficientes Cepstrales en la Frecuencia de Mel (MFCC). Su simplicidad lo convierte en una opción efectiva para tareas de clasificación donde no se dispone de grandes cantidades de datos, aunque su rendimiento puede verse afectado por la elección del parámetro k y la presencia de ruido en los datos.

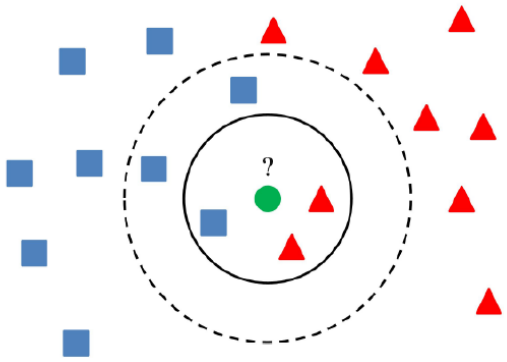

Ilustración 4. Algoritmo KNN [6].

Como se puede ver en la Ilustración 4. Algoritmo KNN [6]., lo que hace el algoritmo KNN es identificar los k puntos más cercanos. El objetivo es clasificar el punto verde. Este método funciona calculando la distancia entre el punto verde y los *k* puntos más cercanos en el espacio de características. Por ejemplo, al seleccionar un valor de *k* igual a 3, los tres objetos más cercanos son los incluidos dentro del círculo de trazo continuo. En este caso, dado que hay dos puntos rojos y uno azul, el punto verde se clasificaría como perteneciente a la clase roja, que es la predominante. Si, en cambio, se elige un valor de *k* igual a 5, se tomarían en cuenta los puntos dentro del círculo de trazo discontinuo, y la clase seleccionada sería la azul, al ser mayoría.

En general, si las características son seleccionadas adecuadamente, las clases estarán bien diferenciadas en el espacio de características, lo que permite que un valor de *k* alto o medio contribuya a que la decisión sea más robusta frente al ruido, es decir, objetos que pertenecen a una clase, pero cuyas características son muy similares a las de otra clase.

2.4.5. Comparativa entre Distintas Arquitecturas.

Para este proyecto, se decidió explorar tanto las Redes Neuronales Convolucionales (CNN) como los modelos de Vecinos Más Cercanos (KNN) debido a sus respectivas capacidades para procesar y analizar representaciones de características como los MFCCs y espectrogramas.

Sin embargo, la elección final se inclinó hacia KNN debido a su simplicidad y efectividad cuando se trabaja con representaciones como los MFCCs. A diferencia de las CNNs, que requieren un entrenamiento extenso y son adecuadas para analizar datos con alta complejidad espacial y temporal, el KNN se basa en la proximidad de las características extraídas en el espacio de datos. Este enfoque es particularmente útil cuando los datos son relativamente homogéneos y bien estructurados, como los MFCCs, que ya representan una simplificación robusta de la señal de audio.

El KNN resultó ser una opción efectiva para este proyecto, ya que permite clasificar nuevas muestras de manera directa basándose en los vecinos más cercanos en el espacio de características. Además, KNN no requiere un proceso de entrenamiento largo, lo que lo convierte en una solución más eficiente en términos de tiempo y recursos computacionales. También es útil en escenarios donde se cuenta con conjuntos de datos más pequeños o en los que se necesita una implementación rápida.

Aunque las redes neuronales CNN son poderosas en la extracción de características jerárquicas a partir de datos complejos como los espectrogramas de Mel, el uso de MFCCs ya proporciona una representación suficientemente informativa para el análisis con KNN. Dado que los MFCCs condensan la información esencial de la señal de audio en un conjunto compacto de características, el KNN fue capaz de ofrecer un buen rendimiento sin la necesidad de una arquitectura más compleja como una CNN.

En resumen, aunque las CNNs tienen su lugar en problemas de mayor complejidad espacial y temporal, la elección de KNN fue más adecuada para este proyecto debido a la naturaleza de los datos y la simplicidad del modelo. KNN demostró ser particularmente eficaz al trabajar con MFCCs, lo que permitió abordar la tarea de clasificación de audio con un enfoque más directo y menos costoso en términos computacionales.

3. Metodología.

En esta sección se detalla la metodología adoptada para llevar a cabo el presente estudio, el cual abarca desde la construcción y organización de la base de datos hasta el entrenamiento y evaluación de los modelos de Machine Learning. Se describirán los pasos seguidos para el aumento de datos, el procesamiento de las señales, el diseño de la arquitectura de la red, así como las estrategias implementadas para el entrenamiento y la evaluación del modelo.

3.1. Frameworks de desarrollo.

Tomando como referencia la siguiente gráfica:

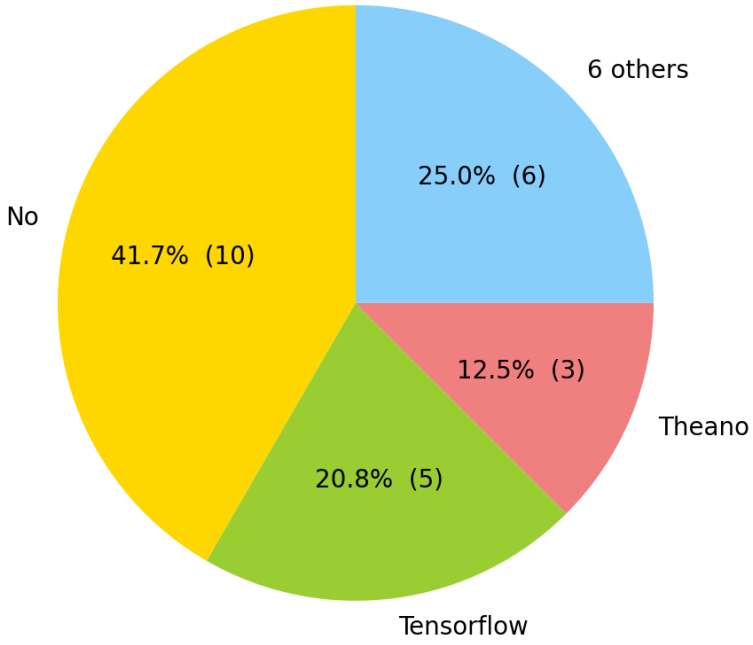

Ilustración 5. Distribución de frameworks en proyectos de machine learning aplicados a música [7].

Se puede observar que la opción predominante es no emplear ningún framework específico, optando en su lugar por programar manualmente todas las funcionalidades de la red. Dado que mi diseño de red es de carácter general y sigue una arquitectura estándar, resulta más eficiente utilizar librerías como TensorFlow, que facilitan y aceleran el proceso de construcción de la red. A continuación, analizaré diferentes frameworks para determinar cuál es el más adecuado:

3.1.1. TensorFlow

TensorFlow es una de las librerías más ampliamente utilizadas en el ámbito del aprendizaje automático. Desarrollada por Google y de código abierto, es compatible con varios lenguajes de

programación, siendo Python el más común. Originalmente, esta librería fue creada para los proyectos internos de Google, pero su uso se ha extendido globalmente en investigación y desarrollo. El nombre TensorFlow proviene de los tensores, que son objetos matemáticos utilizados por las redes neuronales para realizar operaciones sobre conjuntos de datos [8].

A pesar de su amplia adopción, TensorFlow puede resultar difícil de aprender debido a sus términos y conceptos, como las sesiones y los tensores, que no son comunes en otros contextos. Sin embargo, su popularidad ha dado lugar a una extensa comunidad de usuarios, lo que facilita encontrar recursos y soluciones a los problemas que puedan surgir.

TensorFlow está diseñado para ejecutarse en una amplia variedad de dispositivos y arquitecturas, incluyendo CPUs, GPUs y clústeres de servidores.

3.1.2. Theano

Theano es una librería de Python que proporciona al programador herramientas para manipular y evaluar expresiones matemáticas, especialmente aquellas relacionadas con matrices. Al igual que TensorFlow, Theano está diseñado para ejecutarse tanto en CPUs como en GPUs [9].

Aunque su enfoque principal es la realización de cálculos matemáticos, Theano también puede utilizarse para construir sistemas de machine learning, aunque su nivel de abstracción es más bajo.

3.1.3. PyTorch

PyTorch es una librería de machine learning de código abierto desarrollada por Facebook. Su diseño se centra en resolver problemas de procesamiento de lenguaje natural y tareas de visión artificial, lo que la hace menos adecuada para la clasificación musical por géneros, que es el enfoque de este proyecto.

Al igual que TensorFlow, PyTorch utiliza tensores para realizar sus operaciones. PyTorch está construido sobre Torch, una librería basada en Lua que ofrece una gran variedad de algoritmos de machine learning. PyTorch simplifica las complejas expresiones de Torch, haciéndolas más accesibles y fáciles de utilizar [10].

3.1.4. Keras

Keras es una librería para Python que se ejecuta sobre otras librerías de aprendizaje automático como TensorFlow y Theano, proporcionando una interfaz simplificada para la construcción rápida de redes neuronales [11].

Keras permite crear capas básicas, ya sean simples, convolucionales o recurrentes, aunque no permite una experimentación profunda con funciones de activación personalizadas o conexiones neuronales no convencionales.

Para este estudio, la arquitectura de la red ya está establecida, por lo que no se requiere experimentar con nuevas técnicas, funciones de activación o estructuras de red. TensorFlow ofrece una base sólida para los cálculos de machine learning, y Keras facilita la creación de prototipos de redes con el nivel de personalización necesario sin añadir complejidad innecesaria. Por lo tanto, he decidido utilizar Python como lenguaje de programación y Keras sobre TensorFlow para los cálculos de machine learning.

Además de Keras y TensorFlow, también utilizo scikit-learn, una biblioteca ampliamente usada en Python para tareas de clasificación, regresión y agrupamiento. scikit-learn es ideal para la implementación de algoritmos de machine learning clásicos, como la normalización de datos, la selección de características y el entrenamiento de modelos de clasificación y regresión, los cuales son cruciales para complementar el trabajo con redes neuronales en este proyecto.

Finalmente, después de seleccionar el lenguaje de programación, es necesario decidir qué librerías utilizar para trabajar con archivos de audio. Las librerías más potentes y utilizadas en Python son PyDub y LibRosa. Para este proyecto, utilizaré LibRosa para el procesamiento de audio, como la extracción de características.

3.2. Recopilación y preparación de datos.

Lo primero fue elegir una base de datos reducida que tuviera las suficientes canciones como para mostrar unos resultados eficaces sin la necesidad de sacrificar mucho tiempo cada vez que había que ejecutar el modelado de la red neuronal. En mi caso 10 canciones me permitía poder tener resultados fiables sin sacrificar mucho uso computacional, ya que se me generaban variables que almacenaban una cantidad enorme de datos.

3.3. Data Augmentation.

El proceso de data augmentation, o aumento de datos, es una técnica esencial en el ámbito del aprendizaje automático y la ciencia de datos, utilizada para mejorar el rendimiento y la generalización de los modelos. En el contexto del procesamiento de señales de audio, esta técnica es particularmente valiosa debido a la variabilidad inherente y a la complejidad de las señales de audio en aplicaciones del mundo real [12]. A continuación, se detalla el proceso de data augmentation para señales de audio y su implementación práctica en el proyecto.

3.3.1. Objetivos del Data Augmentation

- **Incrementar la Diversidad de Datos**: Generar variaciones realistas de los datos existentes para simular diferentes condiciones de grabación y escenarios de uso.
- **Prevenir el Sobreajuste**: Al aumentar el número de ejemplos de entrenamiento, el modelo puede aprender características más generalizables y no memorizar los datos.

- **Mejorar la Robustez del Modelo**: Hacer que el modelo sea capaz de manejar variaciones naturales en las señales de audio, como cambios en la tonalidad, velocidad y ruido de fondo.

3.3.2. Técnicas de Data Augmentation para Audio

La aplicación de data augmentation en audio involucra varias técnicas que modifican las señales de audio originales de maneras que son comunes en el mundo real, sin alterar su interpretación o significado. Las principales técnicas utilizadas incluyen:

1. Cambio de Tonalidad (Pitch Shifting)

- Descripción: Esta técnica varía la tonalidad de la señal de audio sin cambiar su velocidad. El cambio se realiza modificando la frecuencia de las muestras de audio.
- Implementación: Utilizar una librería como librosa que permite cambiar la tonalidad de una señal de audio mediante una función específica, como librosa.effects.pitch_shift(audio, sr, n_steps), donde n_steps indica la cantidad de semitonos que la señal debe ser desplazada hacia arriba o hacia abajo.

2. Cambio de Velocidad (Time Stretching)

- Descripción: Altera la velocidad de reproducción del audio sin afectar su tonalidad. Este método estira o comprime la señal de audio en el tiempo.
- Implementación: librosa.effects.time_stretch(audio, rate) permite estirar o comprimir una señal de audio, donde rate es el factor por el cual la duración del audio es cambiada. Un valor mayor que 1 hace que el audio sea más rápido, mientras que un valor menor que 1 lo hace más lento.

3. Adición de Ruido

- Descripción: Consiste en añadir ruido blanco u otro tipo de ruido de fondo al audio para simular condiciones de grabación menos ideales y aumentar la robustez del modelo frente a perturbaciones.
- Implementación: Generar un vector de ruido con la misma longitud que la señal de audio y sumarlo a la señal original. La intensidad del ruido puede ser controlada para asegurar que no eclipse las características importantes del audio.

4. Eco

- Descripción: Simular la presencia de eco o reverberación como la que se experimentaría en diferentes entornos acústicos, desde habitaciones pequeñas hasta grandes salas.
- Implementación: Generar una señal introduciendo un retraso en la señal original para simular el eco, ajustando la intensidad de la señal retardada mediante un factor de atenuación. La señal resultante se normaliza para evitar distorsiones en el rango dinámico.

3.3.3. Implementación Práctica en el Estudio

Para implementar data augmentation en este proyecto, se diseñó una serie de funciones de procesamiento de audio donde cada archivo de audio es sujeto a todas las técnicas de augmentación descritas anteriormente como se puede ver en la Ilustración 6. Funciones de data augmentation. e Ilustración 7. Función para aplicar el data augmentation.. Este proceso se lleva a cabo antes de la extracción de características, asegurando que el modelo de red neuronal tenga acceso a un conjunto de datos enriquecido y diverso durante el entrenamiento.

```python
def ruido(signal, noise_percentage_factor):
    noise = np.random.normal(0, signal.std(), signal.size)
    augmented_signal = signal + noise * noise_percentage_factor
    return augmented_signal


def time_stretch(signal, time_stretch_rate):
    return librosa.effects.time_stretch(signal, rate=time_stretch_rate)


def pitch_scale(signal, sr, num_semitones):
    return librosa.effects.pitch_shift(signal, sr=sr, n_steps=num_semitones)

def echo(signal, sr, delay, attenuation):
    delay_samples = int(delay * sr)

    echo_signal = np.zeros(len(signal) + delay_samples)
    echo_signal[:len(signal)] += signal
    echo_signal[delay_samples:] += signal * attenuation

    # Normalizar para evitar distorsiones
    echo_signal = np.clip(echo_signal, -1.0, 1.0)
    return echo_signal
```

Ilustración 6. Funciones de data augmentation.

Al principio implementé el funcionamiento de la data augmentation de forma que almacenaba las señales de audio en ficheros .mp3, lo cual me di cuenta de que no era la forma óptima de manejar los datos, ya que hacía mucho uso de memoria y tiempo. Finalmente adopté las funciones de tal manera que solo manejaba las señales de audio almacenando en una variable, sin necesidad de tener que guardarlas y volverlas a abrir para extraer las señales de audio.

```python
def data_augmentation(signal, sr, song):
    augmented_signals=[]

    for factor_ruido in [0.1, 0.05]:
        noise_audio=ruido(signal, factor_ruido)
        augmented_signals.append(noise_audio)
        print(f"{song}_noise_{factor_ruido}")

    for factor_velocidad in [0.5, 0.75, 1.5, 2]:
        time_audio=time_stretch(signal, factor_velocidad)
        augmented_signals.append(time_audio)
        print(f"{song}_speed_{factor_velocidad}")

    for semitono in [-6,-4,-2, 2, 4, 6]:
        pitch_audio=pitch_scale(signal, sr, semitono)
        augmented_signals.append(pitch_audio)
        print(f"{song}_pitch_{semitono}")

    for delay in [0.1, 0.2, 0.3]:
        for attenuation in [0.1, 0.15]:
            echo_audio=echo(signal, sr, delay, attenuation)
            augmented_signals.append(echo_audio)
            print(f"{song}_echo_{delay}_{attenuation}")

    return augmented_signals
```

Ilustración 7. Función para aplicar el data augmentation.

3.4. Separación Vocal e Instrumental usando Demucs.

En el ámbito del procesamiento de señales de audio, una de las tareas más desafiantes y útiles es la separación de componentes vocales e instrumentales en pistas musicales. Este proceso es fundamental para diversas aplicaciones, como la producción de música, el entrenamiento de modelos de reconocimiento automático de voz y la mejora de sistemas de recomendación musical, entre otros. Utilizar la biblioteca Demucs para esta tarea proporciona una solución robusta y eficiente debido a su enfoque basado en redes neuronales para la separación de fuentes de audio [13].

3.4.1. Descripción de Demucs.

Demucs, que se basa en un modelo de red neuronal profunda, utiliza una combinación de convoluciones y operaciones de desconvolución para aprender a desentrañar las distintas fuentes de audio de una mezcla. Originado como parte de la investigación de Facebook AI, Demucs ha demostrado ser efectivo para separar voces, baterías, bajos y otros instrumentos de grabaciones de audio complejas [14].

3.4.2. Proceso de Separación Vocal e Instrumental

La aplicación de Demucs en nuestro proyecto permite extraer por separado las pistas vocales y los acompañamientos instrumentales de las canciones. Este proceso se lleva a cabo siguiendo estos pasos:

1. **Carga del Audio Original**:

- Se carga la pista de audio original utilizando herramientas estándar como librosa.

2. **Aplicación de Demucs**:

- Demucs se aplica a la pista de audio completa para separar las componentes vocales e instrumentales. Esto se realiza mediante la invocación del modelo preentrenado de Demucs, el cual genera dos pistas separadas: una conteniendo únicamente las vocales y otra con la instrumentación.

3. **Post-Procesamiento**:

- Las pistas separadas se pueden procesar adicionalmente para ajustes de calidad o correcciones menores si es necesario.

3.4.3. Integración con Data Augmentation

Una vez que las pistas vocales e instrumentales están separadas, aplicamos las mismas técnicas de data augmentation descritas anteriormente para enriquecer aún más el conjunto de datos de entrenamiento. Cada pista separada (vocales e instrumentales) se somete a los procesos de cambio de tonalidad, cambio de velocidad, adición de ruido y aplicación de eco ya explicados anteriormente. Esto no solo aumenta la cantidad de datos disponibles para el entrenamiento, sino que también ayuda a la red neuronal a aprender a manejar una variedad más amplia de características y condiciones acústicas.

```
def demucs(cancion):
    import demucs.separate
    demucs.separate.main(["--mp3", "--two-stems", "vocals", "-n", "mdx_extra", 'canciones/'+cancion])
```

Ilustración 8. Función para implementar separación de voz e instrumentación.

3.5. Preprocesamiento de señales de audio.

El preprocesamiento de señales de audio es un paso crucial en el análisis de patrones de audio y en la preparación de datos para su posterior uso en modelos de aprendizaje automático. Este proceso implica la transformación de señales de audio crudas en representaciones que puedan ser procesadas por redes neuronales o algoritmos, mejorando así la eficacia y eficiencia del modelo en tareas de clasificación y emparejamiento de audio.

En este apartado, se detalla el procedimiento implementado para preprocesar las señales de audio utilizadas en este proyecto, el cual se basa en la segmentación de señales, la generación de MFCCs, la normalización de datos, y el uso de técnicas avanzadas como la separación de componentes vocales e instrumentales.

3.5.1. Carga y Segmentación de Datos

El primer paso en el preprocesamiento consiste en cargar las señales de audio desde la base de datos y segmentarlas en fragmentos uniformes. Para asegurar que todos los fragmentos de **MFCCs** tengan la misma forma y puedan ser procesados de manera eficiente por el algoritmo **KNN**, se decidió dividir las señales en fragmentos de una duración fija, en este caso, de 10 segundos. Esta uniformidad en la longitud de los fragmentos es crucial para que el **KNN** pueda comparar correctamente las características de las señales y garantizar que todas las entradas tengan la misma estructura dimensional.

El código encargado de realizar este proceso se encapsula en la función load_data, que toma como parámetros la ruta de los datos (data_path), la duración de los fragmentos (fragment_duration), y la frecuencia de muestreo (sample_rate).

```
def load_data(data_path, fragment_duration, sample_rate, num_mfcc):
    X, y = [], []
    label_to_song = {label: song for label, song in enumerate(os.listdir(data_path))}
    num_classes = len(label_to_song)

    for label, song in enumerate(os.listdir(data_path)):
        song_path = os.path.join(data_path, song)
        signal, sr = librosa.load(song_path, sr=sample_rate)
        signals = []
        signals.append(signal)  # Añadir señal original
```

Ilustración 9. Función load_data

Como se puede observar en la Ilustración 9. Función load_data, la función load_data primero carga cada archivo de audio y lo procesa para segmentarlo en fragmentos. Estos fragmentos se almacenan en listas junto con sus etiquetas correspondientes, las cuales indican a qué clase (o canción) pertenece cada fragmento. Además, se aplica una técnica de **aumento de datos** (data augmentation) para generar variaciones de los fragmentos originales, lo que incrementa la cantidad de datos disponibles para el entrenamiento del modelo y mejora su capacidad de generalización.

3.5.2. Separación de Componentes Vocálicos.

Un aspecto innovador de este trabajo es la separación de las señales en sus componentes vocales e instrumentales utilizando la librería **Demucs**. Esta separación permite al modelo aprender de manera más efectiva las características de cada componente por separado, aumentando la precisión en la identificación de patrones de audio complejos.

```
signal, sr = librosa.load('separated/mdx_extra/'+song.split('.mp3')[0]+'/vocals.mp3', sr=sample_rate)
signal_vocals = data_augmentation(signal, sr, song+'_vocals')
augmented_signals += signal_vocals
augmented_signals.append(signal)

# Extraer la frecuencia fundamental (melodía) de la señal original
signal, sr = librosa.load('separated/mdx_extra/'+song.split('.mp3')[0]+'/vocals.mp3', sr=sample_rate)
f0, voiced_flag, voiced_probs = librosa.pyin(signal,
                                    sr=sr,
                                    fmin=librosa.note_to_hz('C2'),
                                    fmax=librosa.note_to_hz('C7'),
                                    fill_na=None)

times = librosa.times_like(f0)
vneg = (-1)**(~voiced_flag)
melody_signal = mir_eval.sonify.pitch_contour(times, f0 * vneg, sr)

augmented_signals.append(melody_signal)  # Añadir la señal de la melodía

melody_augmented=data_augmentation(melody_signal, sr, song)
augmented_signals+=melody_augmented
```

Ilustración 10. Separación vocálica y frecuencia fundamental dentro de load_data.

Como se puede ver en la Ilustración 10. Separación vocálica y frecuencia fundamental dentro de load_data., este código realiza varios pasos relacionados con el procesamiento de una señal de audio vocal, la aplicación de técnicas de aumento de datos, y la extracción de la melodía (frecuencia fundamental) de la señal [15].

Primero, se carga la señal de audio del audio vocálico de la canción que estamos realizando el procesado de datos.

A continuación, se aplica la técnica de data augmentation a la señal cargada como ya se ha explicado en el apartado anterior.

Después extraemos la frecuencia fundamental (o melodía) de la señala para poder aportar más información al algoritmo. La función librosa.pyin() [16] realiza esta tarea, calculando la frecuencia fundamental a partir de la señal vocal. Este proceso utiliza un rango de frecuencias predefinido que abarca desde la nota C2 (baja) hasta C7 (alta), lo cual se establece con las funciones librosa.note_to_hz(). Una vez tenemos la frecuencia fundamental se procede a sonificarla para también poderle aplicar el proceso de data augmentation.

En este contexto, he utilizado la función pitch_contour de la librería mir_eval para evaluar y extraer el contorno de la melodía. Esta función permite obtener una representación precisa de la frecuencia fundamental a lo largo del tiempo, facilitando su análisis y comparación con el contorno de la melodía original. El uso de pitch_contour garantiza una correcta evaluación del contorno de tono.

3.5.3. Generación de MFCCs.

Cada fragmento de audio, ya sea el original o una versión aumentada, se convierte en un conjunto de **Coeficientes Cepstrales en la Frecuencia de Mel (MFCCs)**, que representan de manera

compacta y eficaz las características perceptuales del contenido frecuencial del audio. Este tipo de representación es especialmente útil en tareas de clasificación de audio, ya que simplifica la señal en un conjunto de características que capturan tanto la información relevante de las frecuencias como de la envolvente temporal, facilitando la comparación entre fragmentos.

```
step_duration = 6  # Duración del paso en segundos

for augmented_signal in augmented_signals:
    for i in range(0, len(augmented_signal) - fragment_duration * sample_rate + 1, step_duration * sample_rate):
        fragment = augmented_signal[i:i + fragment_duration * sample_rate]
        if len(fragment) == fragment_duration * sample_rate:
            rms = np.sqrt(np.mean(fragment**2))

            # Verificar si el RMS está por debajo del umbral
            if rms >= 0.04:  # Filtrar fragmentos con RMS bajo
                mfccs = librosa.feature.mfcc(y=fragment, sr=sr, n_mfcc=num_mfcc)
                X.append(mfccs)
                y.append(label)
```

Ilustración 11. Generación de MFCCs dentro de load_data.

En este proceso, cada fragmento de audio se transforma en un conjunto de **MFCCs** con un número definido de coeficientes, en nuestro caso 13. Luego, los **MFCCs** se normalizan para asegurar que las características se encuentren en una escala comparable, y se ajustan para su uso en el algoritmo de **K-Nearest Neighbors (KNN)**, garantizando que todas las muestras tengan una estructura uniforme y sean adecuadas para el proceso de clasificación.

3.6. Diseño del Modelo de Clasificación con KNN.

El diseño del modelo de clasificación es un componente fundamental en el proceso de aprendizaje automático, ya que determina cómo se procesan y aprenden los patrones presentes en los datos. En este proyecto, se ha implementado el **algoritmo de Vecinos Más Cercanos (K-Nearest Neighbors, KNN)** debido a su simplicidad y eficacia en la clasificación de datos cuando las características están bien estructuradas, como los **MFCCs** generados a partir de las señales de audio.

3.6.1. Arquitectura del Modelo

El modelo **KNN** no utiliza capas o unidades de procesamiento internas como en una red neuronal, sino que se basa en la comparación directa entre las características de las muestras de audio. El modelo busca los **k vecinos más cercanos** en el espacio de características de los **MFCCs** y clasifica una nueva muestra según las etiquetas de esos vecinos.

```python
# Normalizar los datos
scaler = StandardScaler()

# Guardar la forma original del conjunto de prueba
n_samples_pyin, n_mfcc_pyin, n_frames_pyin = X_pyin.shape
# Aplanar las dos últimas dimensiones
X_pyin_flat = X_pyin.reshape(n_samples_pyin, n_mfcc_pyin * n_frames_pyin)
# Normalizar los datos
X_pyin_scaled = scaler.fit_transform(X_pyin_flat)


# Crear y entrenar el modelo KNN
knn = KNeighborsClassifier(n_neighbors=7)
knn.fit(X_pyin_scaled, y_pyin)


# Guardar la forma original
n_samples, n_mfcc, n_frames = X.shape
# Aplanar las dos últimas dimensiones
X_flat = X.reshape(n_samples, n_mfcc * n_frames)
# Normalizar los datos
X_scaled = scaler.transform(X_flat)


y_pred = knn.predict(X_scaled)

# Calcula la precisión usando las etiquetas correctas
accuracy_train = accuracy_score(y_pyin, knn.predict(X_pyin_scaled))
accuracy_test = accuracy_score(y, y_pred)

print(f'Train accuracy: {accuracy_train}')
print(f'Test accuracy: {accuracy_test}')
```

Ilustración 12. Código del algoritmo KNN.

En este proyecto, se ha configurado el algoritmo para que utilice **7 vecinos (k=7)**, ya que este valor ha demostrado ofrecer un buen equilibrio entre precisión y robustez en la clasificación. El valor de **k** determina cuántas muestras cercanas al punto de prueba se toman en cuenta para asignar la clase. Un valor demasiado pequeño de **k** puede hacer que el modelo sea sensible al ruido, mientras que un valor demasiado grande puede diluir la influencia de las clases más cercanas.

3.6.2. Preprocesamiento y Normalización de los Datos.

Para que el algoritmo **KNN** funcione de manera óptima, es crucial transformar y **normalizar** los datos de entrada. Los datos de audio, representados como **MFCCs**, son matrices multidimensionales, donde cada fragmento de audio está representado por un número determinado de coeficientes **MFCC** (en este caso, `n_mfcc`) y la cantidad de tramas temporales (`n_frames`).

Antes de aplicar el modelo, los **MFCCs** de cada fragmento se redimensionan en vectores unidimensionales para que el algoritmo **KNN** pueda operar en un espacio de características simplificado. Esto se hace **redimensionando las dos últimas dimensiones** (coeficientes y marcos) de los datos de entrada, obteniendo vectores de la forma `(n_samples, n_mfcc * n_frames)`.

3.6.3. Entrenamiento y Predicción.

Una vez que los datos han sido transformados y normalizados, se procede a la creación y entrenamiento del modelo **KNN**. El modelo se entrena utilizando los datos normalizados correspondientes a los fragmentos de audio. El objetivo es que el modelo aprenda a asociar cada vector de características **MFCC** con su canción correspondiente.

Después de entrenar el modelo con los datos del conjunto de entrenamiento, se realizan predicciones sobre el conjunto de prueba utilizando las características normalizadas de dicho conjunto. El modelo **KNN** clasifica cada fragmento de audio según la clase predominante entre los **7 vecinos más cercanos**.

3.6.4. Evaluación del Modelo.

El rendimiento del modelo se evalúa mediante la métrica de **precisión (accuracy)**, que mide la proporción de predicciones correctas realizadas por el modelo tanto en el conjunto de entrenamiento como en el de prueba. En este caso, la precisión se calcula utilizando las etiquetas reales de los fragmentos de audio y las predicciones realizadas por el modelo. El uso de dos conjuntos distintos (entrenamiento y prueba) permite evaluar el grado de generalización del modelo, es decir, cómo se desempeña en datos que no ha visto durante el entrenamiento.

```
Train accuracy: 0.813268489631125
Test accuracy: 0.8468137254901961
```

Ilustración 13. Resultados de la Evaluación del Modelo KNN.

Como se observa en la Ilustración 13. Resultados de la Evaluación del Modelo KNN., la diferencia entre los valores de train accuracy (0.81) y test accuracy (0.84) no es inusual ni problemática, y puede deberse a factores como la naturaleza de los datos y la robustez del algoritmo KNN. El hecho de que las predicciones se cumplan indica que el modelo está funcionando correctamente y generalizando bien a los datos nuevos, lo cual es el objetivo principal en problemas de clasificación.

3.6.5. Predicciones.

Para realizar las predicciones sobre nuevas canciones, se utiliza un proceso basado en el modelo KNN entrenado previamente. El método empleado carga la señal de audio de la canción, la segmenta en fragmentos con la misma duración utilizada anteriormente. Esto se debe a que también deben de tener la misma forma que los fragmentos con los que se ha entrenado el algoritmo. A partir de estos fragmentos, extrae las características de MFCCs necesarias para realizar la predicción. Cada fragmento de audio es procesado de manera independiente, donde primero se calcula el nivel de energía (RMS) para asegurar que no se incluyan fragmentos demasiado silenciosos.

```
def predict_song(song_path):
    audio, sr = librosa.load(song_path, sr=sample_rate)
    fragment_predictions = []

    step_duration = 3  # Duración del paso en segundos

    for i in range(0, len(audio) - fragment_duration * sample_rate + 1, step_duration * sample_rate):
        fragment = audio[i:i + fragment_duration * sample_rate]
        if len(fragment) == fragment_duration * sample_rate:
            rms = np.sqrt(np.mean(fragment**2))
            #if rms >= 0.04:  # Filtrar fragmentos con RMS bajo
            # Extraer los MFCCs
            mfccs = librosa.feature.mfcc(y=fragment, sr=sample_rate, n_mfcc=num_mfcc)
            mfccs = mfccs.flatten()  # Aplanar los MFCCs para que sean compatibles con KNN

            # Normalizar los MFCCs con el mismo scaler usado en el entrenamiento
            mfccs = scaler.transform([mfccs])

            # Realizar la predicción usando KNN
            prediction = knn.predict(mfccs)
            fragment_predictions.append(prediction[0])


    if fragment_predictions:
        # Determinar la clase más frecuente en las predicciones de los fragmentos
        predicted_class = max(set(fragment_predictions), key=fragment_predictions.count)

        # Calcular la probabilidad de cada clase
        class_counts = {label_to_song[label]: fragment_predictions.count(label) for label in set(fragment_predictions)}

        for class_name, count in class_counts.items():
            probability = count / len(fragment_predictions)
            print('La probabilidad de ser {} es: {}'.format(class_name, probability))

        return label_to_song[predicted_class]
    else:
        return "No prediction could be made."
```

Ilustración 14. Código para realizar las predicciones.

Los MFCCs extraídos de cada fragmento son redimensionados para convertirlos en un vector unidimensional y luego se normalizan usando la misma normalización extraída y aplicada durante el entrenamiento del modelo. Este paso asegura que las características de los fragmentos de audio nuevos estén en el mismo rango que las características utilizadas en el entrenamiento, permitiendo que el modelo KNN pueda hacer predicciones coherentes.

Cada fragmento es clasificado por el modelo KNN, y las predicciones de todos los fragmentos se recopilan. Una vez se han procesado todos los fragmentos de la canción, se determina la clase más frecuente entre ellos, lo que representa la predicción final para la canción completa. Adicionalmente, se calcula la probabilidad de cada clase en función de cuántos fragmentos fueron asignados a cada una, proporcionando una idea de la confianza en la predicción.

Este enfoque de predicción basada en fragmentos permite al modelo ser robusto frente a posibles ruidos o variaciones en diferentes partes de la señal de audio, asegurando que la predicción final esté basada en la mayor parte de la información de la canción.

4. Alternativa con Espectrogramas y CNN.

A medida que se trabaja con bases de datos más grandes y se dispone de mayor capacidad computacional, el uso de redes neuronales convolucionales (CNN) se convierte en una solución adecuada para la tarea de identificación y emparejamiento de audio. Las CNN son especialmente eficaces cuando pueden aprovechar grandes cantidades de datos, ya que tienen la capacidad de aprender características jerárquicas y complejas de las representaciones de audio, como los espectrogramas de Mel. Cuanta más información tenga el modelo, mejor será su capacidad para extraer patrones relevantes y generalizar a nuevos datos, lo que las hace idóneas para bases de datos extensas y con variabilidad en las señales.

No obstante, en el caso de este proyecto, la cantidad de datos reales disponibles es limitada, y aunque se utilizan técnicas de data augmentation para aumentar la variabilidad, estas no pueden sustituir completamente la riqueza de un conjunto de datos más diverso y grande. El data augmentation permite generar variaciones artificiales de las señales de audio, modificando la tonalidad, el tempo o añadiendo ruido, lo cual es útil para aumentar la robustez del modelo frente a variaciones comunes en las señales. Sin embargo, estas técnicas tienen sus limitaciones, ya que, en esencia, los datos originales siguen siendo los mismos, y las variaciones generadas solo pueden simular escenarios hasta cierto punto.

Las CNN tienden a mostrar un rendimiento superior en problemas donde hay una gran cantidad de datos y variedad en las muestras de entrenamiento. Esto se debe a que, al entrenar con más datos, la red puede captar más detalles específicos y, a la vez, aprender características generales que son aplicables a una variedad de situaciones, como diferentes versiones de una misma canción, ruido de fondo, o cambios en el ritmo. Cuando el modelo se ve limitado por la cantidad de datos, como en este caso donde el data augmentation es la única fuente adicional de variabilidad, la capacidad de la CNN para generalizar puede verse restringida.

En este contexto, aunque la CNN puede ofrecer una gran mejora en la identificación de patrones de audio cuando se dispone de un conjunto de datos más amplio y variado, su eficacia se ve limitada por la falta de diversidad en los datos originales. Si bien el uso de técnicas como el aprendizaje transferido o la adquisición de datos más diversos podría mejorar el rendimiento, el data augmentation solo puede simular variaciones limitadas, lo que restringe el potencial completo de la CNN.

4.1. Diseño del Clasificador mediante Red Neuronal CNN.

4.1.1. Procesamiento de Datos.

Como se ha comentado anteriormente, la red neuronal convolucional funciona mejor con características complejas como pueden ser los espectrogramas, por lo que el procesamiento de datos en los audios varía.

```
for augmented_signal in augmented_signals:
    for i in range(0, len(augmented_signal), fragment_duration * sample_rate):
        fragment = augmented_signal[i:i + fragment_duration * sample_rate]
        if len(fragment) == fragment_duration * sample_rate:
            rms = np.sqrt(np.mean(fragment**2))

            # Verificar si el RMS está por debajo del umbral
            if rms >= 0.04:  # Filtrar fragmentos con RMS bajo
                # Convertir fragmento a espectrograma de Mel
                spectrogram = librosa.feature.melspectrogram(y=fragment, sr=sr, n_mels=128)
                X.append(spectrogram)
                y.append(label)
```

Ilustración 15. Procesamiento de datos en CNN.

Como se puede observar en la Ilustración 15. Procesamiento de datos en CNN., la diferencia en el procesado está en que usando KNN almacenábamos los datos del MFCC y en este caso estamos almacenando los espectrogramas de MEL.

4.1.2. Preparación de Datos para el Modelo.

También debemos de normalizar los datos de forma distinta.

```
X, y, label_to_song, num_classes = load_data(DATA_PATH, fragment_duration, sample_rate)

# Normalización de los espectrogramas para mejorar la estabilidad en el entrenamiento
X = X[..., np.newaxis]  # añadir una dimensión para el canal

y = to_categorical(y, num_classes)
```

Ilustración 16. Normalización de datos para CNN.

Como se observa en la Ilustración 16. Normalización de datos para CNN., a la variable de los espectrogramas debemos de agregarle una dimensión ya que la red CNN se entrena con datos como si fueran imágenes, de forma que debemos introducirle datos en forma de (alto, ancho, canales).

En nuestro caso, X tiene forma de (número de fragmentos, número de “mels”, número de tramas). La primera dimensión corresponde al total de fragmentos de audio que se han procesado y convertido en espectrogramas. La segunda dimensión es el número de mels elegido para los espectrogramas, el cual se puede observar en la Ilustración 15. Procesamiento de datos en CNN.. La tercera dimensión corresponde al número de tramas de tiempo.

Los números de tramas de los fragmentos deben ser todos iguales para poder introducirlos al modelo CNN, por ello se ha llevado a cabo la división por fragmentos de los audios.

En el caso de la variable “y” se utiliza la función to_categorical de modo que si por ejemplo tenemos este array: [0, 1, 3, 5, …], donde cada número es la canción a la que pertenece ese fragmento, convierte ese array a este otro:

[[1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

[0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]

[0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]

[0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]

4.1.3. Arquitectura del Modelo.

La red neuronal diseñada sigue una arquitectura secuencial típica de una CNN, que consta de capas convolucionales seguidas de capas de pooling, capas de redimensionado y capas densas. A continuación, se describen los componentes principales de esta arquitectura:

```
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(X_train.shape[1], X_train.shape[2], 1)),
    MaxPooling2D((2, 2)),
    Dropout(0.25),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Dropout(0.25),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Dropout(0.25),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(num_classes, activation='softmax')
])
```

Ilustración 17. Modelo de la red neuronal.

4.1.3.1. Capas Convolucionales (Conv2D)

Las capas convolucionales (Conv2D) son el núcleo de la red CNN. Estas capas aplican filtros (o kernels) sobre la entrada para detectar características locales, como bordes o texturas en imágenes, que en el caso de los espectrogramas corresponden a patrones de frecuencias específicas en ciertos momentos.

- **Primera capa convolucional:** Se aplica un filtro de 32 kernels de tamaño 3x3, lo que permite detectar patrones básicos en los espectrogramas.
- **Segunda y tercera capa convolucional:** Aumentan la profundidad de la red con 64 y 128 filtros respectivamente, permitiendo que el modelo capture características más complejas a medida que la señal pasa a través de las capas.

Todas las capas convolucionales utilizan la función de activación **ReLU (Rectified Linear Unit)**, que introduce no linealidades en el modelo, lo que es crucial para aprender patrones complejos.

4.1.3.2. Capas de MaxPooling (MaxPooling2D)

Después de cada capa convolucional, se incluye una capa de **MaxPooling** con una ventana de 2x2. Estas capas reducen la dimensionalidad de los mapas de características generados por las capas convolucionales, extrayendo los valores máximos de cada sub-región del mapa. Esto no solo reduce el número de parámetros y el costo computacional, sino que también ayuda a generalizar el modelo al enfocarse en las características más destacadas.

4.1.3.3. Capas de Dropout.

El **Dropout** es una técnica de regularización que se emplea para prevenir el sobreajuste durante el entrenamiento del modelo. Consiste en "apagar" aleatoriamente una fracción de las neuronas en la red durante cada paso de entrenamiento, lo que obliga al modelo a aprender características más robustas y generalizables.

- En las capas convolucionales se aplica un dropout del 25%, mientras que en la capa densa final se aplica un dropout más agresivo del 50%, debido a la mayor cantidad de parámetros en esta capa.

4.1.3.4. Capas Densas (Dense).

La capa Flatten convierte la matriz bidimensional resultante del proceso convolucional en un vector unidimensional, que se conecta a una capa densa (Dense). Esta capa totalmente conectada tiene 128 neuronas con una función de activación ReLU y es seguida por otra capa densa con un número de neuronas igual al número de clases en el problema, utilizando una función de activación softmax para la clasificación multiclase.

La activación softmax transforma las salidas en probabilidades, permitiendo al modelo predecir la probabilidad de que un fragmento de audio pertenezca a cada clase posible.

4.1.4. Compilación y Entrenamiento del Modelo.

Una vez definida la arquitectura del modelo, se procede a su compilación y entrenamiento. La compilación del modelo especifica el optimizador, la función de pérdida y las métricas de evaluación.

```
# Compilar y entrenar el modelo
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(X_train, y_train, epochs=20, validation_split=0.2)
```

Ilustración 18. Compilación y Entrenamiento.

- **Optimizador Adam:** Se utiliza el optimizador **Adam**, una versión avanzada del descenso de gradiente que ajusta las tasas de aprendizaje de manera adaptativa para cada parámetro. Esto lo hace especialmente eficaz y popular en la práctica [17].

- **Pérdida Categórica Cruzada:** La función de pérdida seleccionada es **categorical_crossentropy**, adecuada para problemas de clasificación multiclase, ya que mide la disimilitud entre la distribución de las predicciones y la distribución real de las etiquetas.
- **Métrica de Precisión:** La métrica principal para evaluar el rendimiento del modelo durante el entrenamiento es la **precisión** (accuracy), que indica la proporción de predicciones correctas.

El modelo se entrena durante 20 épocas, utilizando un 20% del conjunto de entrenamiento como validación. Durante este proceso, el modelo ajusta sus pesos para minimizar la función de pérdida en el conjunto de entrenamiento, mientras monitorea su rendimiento en el conjunto de validación para evitar el sobreajuste.

4.1.5. Evaluación del Modelo.

Finalmente, el modelo se evalúa en el conjunto de prueba para medir su capacidad de generalización a nuevos datos.

```
# Evaluar el modelo
loss, accuracy = model.evaluate(X_test, y_test)
print(f'Test accuracy: {accuracy}')
```

Ilustración 19. Evaluación del modelo.

Esta evaluación devuelve la pérdida y la precisión en el conjunto de prueba, proporcionando una medida de qué tan bien el modelo puede predecir correctamente las clases de fragmentos de audio no vistos durante el entrenamiento.

El diseño y la implementación de este modelo CNN permiten abordar el problema de identificación y emparejamiento de fragmentos de audio con un alto grado de precisión, aprovechando tanto las características temporales como frecuenciales capturadas en los espectrogramas de Mel.

4.1.6. Predicciones.

Para poder llevar a cabo las predicciones se creó esta función, siguiendo el mismo funcionamiento que load_data:

```python
def predict_song(song_path):
    audio, sr = librosa.load(song_path, sr=sample_rate)
    fragment_probabilities = []

    for i in range(0, len(audio), fragment_duration * sample_rate):
        fragment = audio[i:i + fragment_duration * sample_rate]
        if len(fragment) == fragment_duration * sample_rate:  # asegurar fragmentos completos
            rms = np.sqrt(np.mean(fragment**2))
            if rms >= 0.04:  # Filtrar fragmentos con RMS bajo
                # Convertir el fragmento a espectrograma de Mel (sin escala logarítmica)
                spectrogram = librosa.feature.melspectrogram(y=fragment, sr=sr, n_mels=128)

                # Añadir dimensiones de batch y canal
                spectrogram = spectrogram[np.newaxis, ..., np.newaxis]

                # Hacer la predicción
                prediction = model.predict(spectrogram, verbose=0)
                fragment_probabilities.append(prediction[0])

    if fragment_probabilities:
        # Promediar las probabilidades de todas las predicciones
        avg_probabilities = np.mean(fragment_probabilities, axis=0)
        predicted_class = np.argmax(avg_probabilities)
        class_probabilities = {class_name: probability for class_name, probability in zip(label_to_song.values(), avg_probabilities)}

        # Mostrar las probabilidades de cada clase
        for class_name, probability in class_probabilities.items():
            print(f'La probabilidad de ser {class_name} es: {probability:.4f}')

        # Retornar la clase predicha
        return label_to_song[predicted_class]
    else:
        return "No prediction could be made."
```

Ilustración 20. Función de predicción para CNN.

Como se puede observar, se vuelven a crear los fragmentos para tener las mismas dimensiones que el modelo entrenado y se generan sus espectrogramas. A cada espectrograma se le hace la predicción y se almacena.

Luego se promedian para obtener la predicción final de la clase de la canción. Este enfoque de promediar las probabilidades de los fragmentos es clave para la robustez del modelo, permitiendo que la decisión final refleje de manera más precisa la totalidad del archivo de audio en lugar de depender de un solo fragmento.

5. Resultados y Conclusiones.

En este apartado se presentan los resultados obtenidos tras la implementación y evaluación de dos enfoques diferentes para la tarea de identificación y emparejamiento de fragmentos de audio: la Red Neuronal Convolucional (CNN) utilizando espectrogramas de Mel y el modelo de Vecinos Más Cercanos (KNN) empleando Coeficientes Cepstrales en la Frecuencia de Mel (MFCCs).

5.1. Evaluación del Modelo KNN.

El uso del algoritmo KNN en este proyecto ha demostrado ser una solución efectiva para la clasificación de fragmentos de audio representados por MFCCs. La simplicidad del algoritmo, junto con la normalización de los datos y la elección adecuada del valor de k, ha permitido obtener buenos resultados en términos de precisión tanto en el conjunto de entrenamiento como en el de prueba. Además, KNN es particularmente útil en situaciones como esta, donde la base de datos es relativamente reducida, ya que su coste computacional es bajo en comparación con otros modelos más complejos, como las redes neuronales convolucionales (CNN).

```
La probabilidad de ser 3_pecados_despues.mp3 es: 0.02666666666666667
La probabilidad de ser If_We_Being_Real.mp3 es: 0.013333333333333334
La probabilidad de ser lovely.mp3 es: 0.7733333333333333
La probabilidad de ser Lucid_Dreams.mp3 es: 0.013333333333333334
La probabilidad de ser revenge.mp3 es: 0.02666666666666667
La probabilidad de ser yebbas-heartbreak.mp3 es: 0.14666666666666667
Cancion: tests/lovely_cover.mp3 Predicted class: lovely.mp3

La probabilidad de ser yebbas-heartbreak.mp3 es: 0.1076923076923077
La probabilidad de ser If_We_Being_Real.mp3 es: 0.07692307692307693
La probabilidad de ser lovely.mp3 es: 0.7692307692307693
La probabilidad de ser fix_you.mp3 es: 0.046153846153846156
Cancion: tests/lovely_cover2.mp3 Predicted class: lovely.mp3

La probabilidad de ser yebbas-heartbreak.mp3 es: 1.0
Cancion: tests/yebba_cover.mp3 Predicted class: yebbas-heartbreak.mp3

La probabilidad de ser yebbas-heartbreak.mp3 es: 0.8125
La probabilidad de ser If_We_Being_Real.mp3 es: 0.03125
La probabilidad de ser lovely.mp3 es: 0.09375
La probabilidad de ser revenge.mp3 es: 0.0625
Cancion: tests/yebba_cover2.mp3 Predicted class: yebbas-heartbreak.mp3

La probabilidad de ser yebbas-heartbreak.mp3 es: 0.8947368421052632
La probabilidad de ser If_We_Being_Real.mp3 es: 0.02631578947368421
La probabilidad de ser lovely.mp3 es: 0.05263157894736842
La probabilidad de ser fix_you.mp3 es: 0.02631578947368421
Cancion: tests/yebba_cover3.mp3 Predicted class: yebbas-heartbreak.mp3
```

Ilustración 21. Ejemplo de predicciones.

Dado que KNN no requiere un proceso de entrenamiento intensivo, su implementación es rápida y eficaz cuando el conjunto de datos es pequeño. En este caso, la naturaleza del algoritmo permitió aprovechar al máximo los datos disponibles sin incurrir en altos costes computacionales, lo que lo convierte en una opción ideal para problemas de clasificación en los que se dispone de una cantidad limitada de datos.

5.2. Evaluación del Modelo CNN.

La Red Neuronal Convolucional (CNN) fue entrenada utilizando espectrogramas de Mel, lo que permitió capturar tanto la estructura temporal como frecuencial de las señales de audio. El modelo mostró un rendimiento sobresaliente, alcanzando una precisión del 96%.

```
Epoch 1/10
211/211 ━━━━━━━━━━━━━━━━━━━━ 291s 1s/step - accuracy: 0.3347 - loss: 16.4954 - val_accuracy: 0.8213 - val_loss: 0.5826
Epoch 2/10
211/211 ━━━━━━━━━━━━━━━━━━━━ 294s 1s/step - accuracy: 0.7860 - loss: 0.7036 - val_accuracy: 0.8913 - val_loss: 0.3843
Epoch 3/10
211/211 ━━━━━━━━━━━━━━━━━━━━ 290s 1s/step - accuracy: 0.8440 - loss: 0.5092 - val_accuracy: 0.9192 - val_loss: 0.2491
Epoch 4/10
211/211 ━━━━━━━━━━━━━━━━━━━━ 290s 1s/step - accuracy: 0.8986 - loss: 0.3341 - val_accuracy: 0.9276 - val_loss: 0.2274
Epoch 5/10
211/211 ━━━━━━━━━━━━━━━━━━━━ 288s 1s/step - accuracy: 0.9116 - loss: 0.2700 - val_accuracy: 0.9436 - val_loss: 0.1875
Epoch 6/10
211/211 ━━━━━━━━━━━━━━━━━━━━ 283s 1s/step - accuracy: 0.9310 - loss: 0.2404 - val_accuracy: 0.9448 - val_loss: 0.1812
Epoch 7/10
211/211 ━━━━━━━━━━━━━━━━━━━━ 282s 1s/step - accuracy: 0.9477 - loss: 0.1695 - val_accuracy: 0.9513 - val_loss: 0.1744
Epoch 8/10
211/211 ━━━━━━━━━━━━━━━━━━━━ 281s 1s/step - accuracy: 0.9496 - loss: 0.1737 - val_accuracy: 0.9495 - val_loss: 0.1742
Epoch 9/10
211/211 ━━━━━━━━━━━━━━━━━━━━ 279s 1s/step - accuracy: 0.9555 - loss: 0.1435 - val_accuracy: 0.9650 - val_loss: 0.1474
Epoch 10/10
211/211 ━━━━━━━━━━━━━━━━━━━━ 286s 1s/step - accuracy: 0.9577 - loss: 0.1439 - val_accuracy: 0.9632 - val_loss: 0.1440
30/30 ━━━━━━━━━━━━━━━━━━━━ 7s 245ms/step - accuracy: 0.9660 - loss: 0.1160
Test accuracy: 0.9647436141967773
```

Ilustración 22. Precisión del Modelo CNN.

Esta alta precisión se debe en gran parte a la capacidad del modelo para identificar características locales específicas en los espectrogramas, y a la robustez que ofrecieron las técnicas de regularización utilizadas, como el dropout, que previnieron el sobreajuste.

Sin embargo, es importante señalar que el entrenamiento de la CNN requirió una cantidad considerable de recursos computacionales, tanto en términos de tiempo como de capacidad de procesamiento.

```
La probabilidad de ser 3_pecados_despues.mp3 es: 0.0613
La probabilidad de ser fix_you.mp3 es: 0.1867
La probabilidad de ser If_We_Being_Real.mp3 es: 0.1353
La probabilidad de ser jocelyn_flores.mp3 es: 0.0037
La probabilidad de ser lovely.mp3 es: 0.4136
La probabilidad de ser Lucid_Dreams.mp3 es: 0.0218
La probabilidad de ser Promotion.mp3 es: 0.0507
La probabilidad de ser revenge.mp3 es: 0.0862
La probabilidad de ser shape_of_you.mp3 es: 0.0116
La probabilidad de ser yebbas-heartbreak.mp3 es: 0.0291
Cancion: tests/lovely_cover2.mp3 Predicted class: lovely.mp3

La probabilidad de ser 3_pecados_despues.mp3 es: 0.0007
La probabilidad de ser fix_you.mp3 es: 0.0645
La probabilidad de ser If_We_Being_Real.mp3 es: 0.0008
La probabilidad de ser jocelyn_flores.mp3 es: 0.0005
La probabilidad de ser lovely.mp3 es: 0.0670
La probabilidad de ser Lucid_Dreams.mp3 es: 0.1166
La probabilidad de ser Promotion.mp3 es: 0.0015
La probabilidad de ser revenge.mp3 es: 0.1314
La probabilidad de ser shape_of_you.mp3 es: 0.0658
La probabilidad de ser yebbas-heartbreak.mp3 es: 0.5511
Cancion: tests/yebba_cover.mp3 Predicted class: yebbas-heartbreak.mp3

La probabilidad de ser 3_pecados_despues.mp3 es: 0.0005
La probabilidad de ser fix_you.mp3 es: 0.0010
La probabilidad de ser If_We_Being_Real.mp3 es: 0.0054
La probabilidad de ser jocelyn_flores.mp3 es: 0.0000
La probabilidad de ser lovely.mp3 es: 0.0747
La probabilidad de ser Lucid_Dreams.mp3 es: 0.0067
La probabilidad de ser Promotion.mp3 es: 0.0545
La probabilidad de ser revenge.mp3 es: 0.0006
La probabilidad de ser shape_of_you.mp3 es: 0.0127
La probabilidad de ser yebbas-heartbreak.mp3 es: 0.8439
Cancion: tests/yebba_cover3.mp3 Predicted class: yebbas-heartbreak.mp3
```

```
La probabilidad de ser 3_pecados_despues.mp3 es: 0.0061
La probabilidad de ser fix_you.mp3 es: 0.0005
La probabilidad de ser If_We_Being_Real.mp3 es: 0.0003
La probabilidad de ser jocelyn_flores.mp3 es: 0.0047
La probabilidad de ser lovely.mp3 es: 0.1147
La probabilidad de ser Lucid_Dreams.mp3 es: 0.0146
La probabilidad de ser Promotion.mp3 es: 0.0060
La probabilidad de ser revenge.mp3 es: 0.0114
La probabilidad de ser shape_of_you.mp3 es: 0.0018
La probabilidad de ser yebbas-heartbreak.mp3 es: 0.8399
Cancion: tests/yebba_cover_corto.mp3 Predicted class: yebbas-heartbreak.mp3

La probabilidad de ser 3_pecados_despues.mp3 es: 0.0008
La probabilidad de ser fix_you.mp3 es: 0.0597
La probabilidad de ser If_We_Being_Real.mp3 es: 0.0003
La probabilidad de ser jocelyn_flores.mp3 es: 0.0016
La probabilidad de ser lovely.mp3 es: 0.0483
La probabilidad de ser Lucid_Dreams.mp3 es: 0.0005
La probabilidad de ser Promotion.mp3 es: 0.0011
La probabilidad de ser revenge.mp3 es: 0.0017
La probabilidad de ser shape_of_you.mp3 es: 0.0549
La probabilidad de ser yebbas-heartbreak.mp3 es: 0.8312
Cancion: tests/yebba_corto.mp3 Predicted class: yebbas-heartbreak.mp3

La probabilidad de ser 3_pecados_despues.mp3 es: 0.1293
La probabilidad de ser fix_you.mp3 es: 0.1433
La probabilidad de ser If_We_Being_Real.mp3 es: 0.3491
La probabilidad de ser jocelyn_flores.mp3 es: 0.0003
La probabilidad de ser lovely.mp3 es: 0.0510
La probabilidad de ser Lucid_Dreams.mp3 es: 0.2687
La probabilidad de ser Promotion.mp3 es: 0.0001
La probabilidad de ser revenge.mp3 es: 0.0000
La probabilidad de ser shape_of_you.mp3 es: 0.0579
La probabilidad de ser yebbas-heartbreak.mp3 es: 0.0004
Cancion: tests/real_slowed.mp3 Predicted class: If_We_Being_Real.mp3
```

Ilustración 23. Resultados de las predicciones usando CNN.

5.3. Limitaciones.

Durante el desarrollo y evaluación de este proyecto se identificaron varias limitaciones que es importante considerar para futuras investigaciones y aplicaciones:

1. **Requerimientos Computacionales:** La CNN, aunque potente, requiere un alto poder computacional para su entrenamiento e inferencia, lo que puede ser un obstáculo en sistemas con recursos limitados.
2. **Tamaño de los Datos:** Tanto la CNN como el KNN mostraron limitaciones al trabajar con un conjunto de datos reducido. Aunque se aplicaron técnicas de aumento de datos, un mayor volumen de datos reales podría mejorar significativamente la capacidad de la red CNN para generalizar.
3. **Sensibilidad a la Configuración:** El rendimiento del KNN es altamente dependiente de la selección del número de vecinos y de la normalización de los datos. Además, la CNN puede verse afectada por la sintonización subóptima de sus hiperparámetros.

5.4. Conclusiones.

Como se puede observar en los tests realizados durante este proyecto, el KNN mostró un rendimiento superior en ciertos escenarios. Este mejor rendimiento puede explicarse por varios factores:

- **Tamaño y Características del Conjunto de Datos:** El conjunto de datos utilizado era relativamente pequeño y homogéneo. En tales condiciones, el KNN puede aprovechar al

máximo cada punto de datos individual, mientras que la CNN podría no haber explotado completamente su potencial debido a la falta de complejidad en los datos.

- **Configuración de Hiperparámetros:** En el caso del KNN, el único hiperparámetro relevante que se utilizó fue el número de vecinos (k). Como no se emplearon valores extremos de k (ni muy pequeños ni muy grandes), los resultados no fueron particularmente sensibles a este valor. Esto permitió al KNN mantener un rendimiento estable, a diferencia de la CNN, que pudo haber sido más sensible a la configuración de sus múltiples hiperparámetros.
- **Simplicidad y Robustez del KNN:** La simplicidad del KNN, al no requerir un entrenamiento complejo, le permitió manejar mejor los datos en ciertos contextos, especialmente en un entorno de baja variabilidad, donde las diferencias entre clases son más claras.

Como se observa en la Ilustración 13. Resultados de la Evaluación del Modelo KNN., el KNN logró una precisión que osciló entre el 75% y el 85%, superando en algunas predicciones a la CNN, lo que sugiere que, en escenarios específicos, la simplicidad del KNN puede ofrecer ventajas sobre modelos más complejos.

Las Redes Neuronales Convolucionales son generalmente más adecuadas para la identificación y emparejamiento de fragmentos de audio en entornos con alta variabilidad [18]. Sin embargo, los resultados obtenidos en este proyecto subrayan que, bajo ciertas condiciones, como un conjunto de datos reducido y homogéneo, el modelo KNN puede ofrecer un rendimiento superior debido a su simplicidad y menor dependencia de una configuración óptima.

Las mejoras en la precisión y robustez del modelo CNN demuestran que estas técnicas pueden superar los desafíos presentes en el reconocimiento de audio bajo condiciones adversas, tales como ruido de fondo o variaciones en el tono y ritmo. Esta capacidad es crucial para aplicaciones como Shazam o SoundHound, que podrían beneficiarse de un sistema más robusto que no solo identifique canciones con alta precisión, sino que también lo haga en escenarios donde las versiones de las canciones estén alteradas, como versiones en vivo, covers, o incluso fragmentos silbados.

Además, el modelo basado en KNN, aunque más simple, mostró ser extremadamente eficiente en términos de recursos computacionales, lo que lo hace ideal para aplicaciones en dispositivos móviles o en entornos con limitaciones de hardware. Esto sugiere que las técnicas desarrolladas aquí no solo tienen aplicaciones en entornos de alta capacidad, sino que también pueden ser implementadas en soluciones más accesibles y portátiles, permitiendo una mayor democratización del acceso a tecnologías avanzadas de reconocimiento de audio.

En términos de impacto global, este trabajo abre la puerta a nuevas aplicaciones en áreas como la accesibilidad. Por ejemplo, personas con discapacidades auditivas podrían beneficiarse de herramientas que identifiquen patrones sonoros relevantes en tiempo real, permitiéndoles una mejor interacción con su entorno. Asimismo, la capacidad de estos modelos para adaptarse a variaciones en el audio sugiere que podrían ser utilizados en sistemas de recomendación musical más sofisticados, que no solo se basen en los gustos musicales previos del usuario, sino que

también reconozcan fragmentos de música grabados en condiciones no ideales, como en una fiesta o un concierto.

En conclusión, este proyecto no solo contribuye a la mejora técnica en el procesamiento de señales de audio, sino que también tiene aplicaciones prácticas de amplio alcance, que podrían transformar la manera en que interactuamos con la música y el sonido en nuestra vida cotidiana. El impacto de este trabajo podría extenderse a diversas industrias, desde el entretenimiento hasta la accesibilidad, demostrando la versatilidad y la relevancia de las técnicas desarrolladas.

6. Líneas Futuras.

El presente estudio ha demostrado la viabilidad y eficacia de utilizar modelos de aprendizaje automático, como las Redes Neuronales Convolucionales (CNN) y el algoritmo de Vecinos Más Cercanos (KNN), para la identificación y emparejamiento de fragmentos de audio. Sin embargo, existen varias áreas en las que se puede continuar desarrollando y mejorando este trabajo. A continuación, se presentan algunas líneas futuras de investigación y desarrollo que podrían explorarse para avanzar en este campo.

6.1. Ampliación y Diversificación del Conjunto de Datos.

Uno de los desafíos clave identificados en este estudio es la limitación impuesta por el tamaño reducido del conjunto de datos utilizado. Para mejorar la capacidad de generalización de los modelos, sería beneficioso trabajar con conjuntos de datos más grandes y diversificados que incluyan una mayor variedad de géneros musicales, diferentes estilos de interpretación y condiciones acústicas variadas. Además, incorporar datos de audio en diferentes idiomas y culturas podría hacer que los modelos sean más inclusivos y aplicables en un contexto global.

6.2. Optimización de Hiperparámetros y Modelos.

El rendimiento de los modelos CNN y KNN podría mejorarse mediante la optimización más avanzada de sus hiperparámetros. En el caso de la CNN, se podrían explorar diferentes arquitecturas, como redes más profundas o el uso de técnicas.

6.3. Exploración de Enfoques Híbridos.

Dado que el estudio ha revelado que tanto la CNN como el KNN tienen sus propias ventajas y limitaciones, una línea prometedora de investigación es el desarrollo de enfoques híbridos que combinen lo mejor de ambos mundos. Por ejemplo, se podría utilizar la CNN para extraer características complejas y luego emplear un KNN para la fase de clasificación final, aprovechando la robustez del primero y la simplicidad del segundo. Otra posibilidad sería la combinación de diferentes técnicas de machine learning para mejorar la precisión y la adaptabilidad de los modelos a diferentes tipos de datos de audio.

6.4. Implementación de Técnicas Avanzadas de Data Augmentation.

Aunque este estudio ya utilizó técnicas de aumento de datos, existen enfoques más avanzados que podrían ser explorados. Por ejemplo, se podrían aplicar técnicas de síntesis de audio, generación de música mediante redes generativas adversarias (GANs), o el uso de transformaciones más sofisticadas que simulen escenarios acústicos aún más variados. Estas técnicas podrían ayudar a crear conjuntos de datos más robustos y mejorar la capacidad de los modelos para manejar condiciones de audio extremadamente diversas.

6.5. Aplicaciones en Tiempo Real y Entornos Móviles.

Otra dirección importante para futuras investigaciones es la implementación de estos modelos en aplicaciones de tiempo real y en dispositivos móviles [19]. Esto requeriría optimizar los modelos para que sean más ligeros y eficientes en términos de recursos computacionales, asegurando que puedan operar en dispositivos con limitaciones de hardware y en situaciones donde la latencia es crítica. El desarrollo de aplicaciones móviles que utilicen estos modelos podría tener un impacto significativo en la forma en que las personas interactúan con la música y el audio en su vida cotidiana.

6.6. Integración con Tecnologías Emergentes.

Finalmente, una línea futura interesante es la integración de estos modelos con tecnologías emergentes como la realidad aumentada (AR), la realidad virtual (VR), y los asistentes de voz. Por ejemplo, se podrían desarrollar sistemas que identifiquen música en entornos de realidad aumentada, proporcionando información contextual en tiempo real, o mejorar la interacción con asistentes de voz mediante la identificación precisa de canciones y sonidos en diversos entornos acústicos.

6.7. Investigaciones Futuras alineadas con los ODS.

Una posible dirección para futuras investigaciones podría centrarse en la integración de las técnicas desarrolladas en este proyecto en sistemas de educación accesible, contribuyendo así al ODS 4, 'Educación de Calidad'. Por ejemplo, la identificación precisa de patrones de audio podría aplicarse en herramientas educativas diseñadas para personas con discapacidades auditivas, promoviendo una mayor inclusión y accesibilidad en la educación. Asimismo, la investigación en torno a la minimización del impacto medioambiental de los procesos de cómputo masivo podría alinearse con el ODS 13, 'Acción por el Clima', explorando formas de optimizar el uso de recursos energéticos en aplicaciones de Machine Learning.

7. Bibliografía.

[1] Steven B. Davies. *Comparison of Parametric Representations for Monosyllabic Word Recognition in Continuously Spoken Sentences.*

https://courses.grainger.illinois.edu/ece417/fa2017/davis80.pdf

[2] S. S. Stevens. *A Scale for the Measurement of the Psychological Magnitude Pitch.*

https://github.com/embatbr/graduation-project/blob/master/docs/references/A%20Scale%20for%20the%20Measurement%20of%20the%20Psychological%20Magnitude%20Pitch.pdf

[3] M. Jordan, J. Kleinberg, B. Scholkopf. *Pattern Recognition and Machine Learning.*

https://www.microsoft.com/en-us/research/uploads/prod/2006/01/Bishop-Pattern-Recognition-and-Machine-Learning-2006.pdf

[4] James Le. *The 10 Neural Network Architectures Machine Learning Researchers*

Need To Learn.

https://medium.com/cracking-the-data-science-interview/a-gentle-introduction-to-neural-networks-for-machine-learning-d5f3f8987786

[5] Edian F. Franco. *Aprendizaje de máquina y aprendizaje profundo en biotecnología: aplicaciones, impactos y desafíos.*

https://www.researchgate.net/publication/338026011_Aprendizaje_de_maquina_y_aprendizaje_profundo_en_biotecnologia_aplicaciones_impactos_y_desafios

[6] Juan Carlos Tovar. *Algoritmo K-Nearest Neighbors.*

https://forum.huawei.com/enterprise/es/algoritmo-k-nearest-neighbors/thread/667228973744144385-667212895009779712

[7] GitHub. *Awesome Deep Learning Music.*

https://github.com/ybayle/awesome-deep-learning-music/blob/master/README.md

[8] Incentro. *Tensor Flow.*

https://www.incentro.com/es-ES/blog/que-es-tensorflow

[9] Jesús Iris González. *Procesamiento de Audio con Técnicas de Inteligencia Artificial.*

https://e-archivo.uc3m.es/rest/api/core/bitstreams/8ef89078-c4d9-49e0-a29c-e959e217ff77/content

[10] IBM. *¿Qué es PyTorch?*

https://www.ibm.com/es-es/topics/pytorch

[11] DataScientest. Keras: *todo sobre la API de Deep Learning.*

https://datascientest.com/es/keras-la-api-de-deep-learning

[12] Zenodo. *Audio Data Augmentation with respect to Musical Instrument Recognition.*

https://zenodo.org/records/1066137

[13] PyTorch. *Music Source Separation with Hybrid Demucs*

https://pytorch.org/audio/main/tutorials/hybrid_demucs_tutorial.html

[14] GitHub. *Demucs.*

https://github.com/facebookresearch/demucs/tree/main

[15] AudioLabs. *Melody Extraction and Separation*

https://www.audiolabs-erlangen.de/resources/MIR/FMP/C8/C8S2_MelodyExtractSep.html

[16] Librosa. *Librosa.pyin().*

https://librosa.org/doc/latest/generated/librosa.pyin.html

[17] Medium. *CNNs for Audio Classification.*

https://towardsdatascience.com/cnns-for-audio-classification-6244954665ab

[18] Jon Nordby. *Environmental Sound Classification on Microcontrollers using Convolutional Neural Networks.*

https://github.com/jonnor/ESC-CNN-microcontroller/releases/download/print1/report-print1.pdf

[19] Sander. *Recommending music on Spotify with deep learning.*

https://sander.ai/2014/08/05/spotify-cnns.html

[20] Unión Europea. *Reglamento General de Protección de Datos.*

https://www.boe.es/doue/2016/119/L00001-00088.pdf

[21] IEEE SA. *IEEE 1857.4-2018.*

https://standards.ieee.org/ieee/1857.4/5817/

[22] Naciones Unidas. Objetivos de Desarrollo Sostenible.

https://www.pactomundial.org/que-puedes-hacer-tu/ods/?gad_source=1&gclid=Cj0KCQjwiuC2BhDSARIsALOVfBJWG_NMzqLBej5j8a2Hk6CmIyNBa1_jVdUSyOMRb2rVO62yZTJizKIaAll-EALw_wcB