

DIPLOMADO INTELIGENCIA ARTIFICIAL APLICADA

DGTIC UNAM
PROYECTO FINAL

Detección y transcripción a partitura de cadencia y tono en tarareos

Nombre:

Elí Jafet Velásquez Luna

https://github.com/JafetMoon/hum_transcription_Project_DIA.git

Resumen

En el ámbito de la composición musical, tanto académica como amateur, se han desarrollado recursos para facilitar el arreglo y composición de melodías. Programas de escritura como Sibelius o GuitarPro han permitido optimizar la edición de partituras; pero el proceso que lleva de la gestación de una idea a la codificación en partitura es un área muy diversa que puede beneficiarse de herramientas nuevas. En este proyecto se busca desarrollar un sistema que permita a los músicos, a partir de su tarareo, reconocer notas y transcribirlas a un formato adecuado para la comunicación digital entre instrumentos y programas de edición de partitura, como lo es el MIDI. Se analizó el conjunto de datos público HumTrans que consiste en audios y MIDI de 1000 segmentos ejecutadas por 10 músicos.

El sistema de reconocimiento consiste en dos etapas: reconocimiento de ejecuciones y reconocimiento de nota fundamental. El primero es un sistema Conv2D Encoder-Decoder para codificar audios de tarareos en cadenas MIDI con una distancia de Levenshtein promedio 74.8 % eficaz y precisión 81.4 %. La capa convolucional inicial permite agregar características y minimizar el ruido del audio, mientras que el sistema Encoder-Decoder permite traducir secuencias de audio en secuencias MIDI. El segundo sistema clusteriza el audio con K-medias para encontrar la nota fundamental, en todos las ejecuciones encontradas por el primer sistema.

Índice

3. Introducción	3
4. Problema	4
4.1. Contexto	4
4.2. Técnico	4
4.3. Sistema artificial	6
5. Planeación y diseño	7
5.1. Conceptos relevantes	7
5.2. Sistemas existentes	7
5.3. Herramienta de IA	8
5.3.1. Metodología	9
5.4. Alcance	10
6. Adquisición y exploración de datos	12
6.1. Origen	12
6.2. Descripción	12
6.3. Análisis exploratorio	13
6.3.1. MIDI	13
6.3.2. Audios	17
6.4. Transformación	18
6.4.1. MIDI	18
6.4.2. Audio	20
6.5. Datos preprocesados	21
7. Modelado	22
7.1. Arquitectura	22
7.1.1. Sistema detector de ejecución	22
7.2. Evaluación	24
7.2.1. Descripción	24
7.2.2. Resultados preliminares	25
8. Conclusiones	26

3. Introducción

El ámbito de la composición musical, tanto en entornos de música académica como en la música informal, es un proceso creativo no estandarizado que conlleva la expresión y organización de elementos musicales para formar piezas que reflejen las ideas del compositor. Este proceso puede estar compuesto por diversas fases desde la conceptualización de una idea semilla, hasta arreglos complejos de orquestación y armonización.

La composición, como es de esperarse de los procesos en áreas artísticas, no es un proceso con una metodología única al día de hoy, y depende de factores subjetivos como la intención y visión del compositor; así como de enfoques más teóricos y estructurados (propios de la música académica), o enfoques más experimentales basados en la inspiración (más informales).

Una de las características que comparten las fases de la composición musical es la de la transcripción de las melodías en partituras. Si bien, con una formación en música, un *copista* podría encargarse de llevar estas ideas al atril, la realidad es que el compositor principiante o aficionado tiene que enfrentarse a este proceso más de una vez en el proceso de composición, lo que puede resultar cansado, complicado o innecesariamente repetitivo.

El *tararear* melodías al concebir ideas, frases o motivos, es una práctica común entre los músicos al momento de componer. Los *tarareos* pueden llevarse al papel para realizar un análisis armónico más exhaustivo. Sin embargo, el proceso puede consumir tiempo y esfuerzo sin aportar sustancialmente al proceso creativo. Las alternativas actuales son los sistemas de *transcripción automáticos de música* (AMT por sus siglas en inglés) que permiten un rápido acceso a partituras.

La AMT ha evolucionado significativamente, desde técnicas de procesamiento de señales que utilizan autocorrelación y características espectrales para caracterizar la información de tono, hasta Modelos Ocultos de Markov para capturar la variabilidad de las notas musicales.

Las redes neuronales han permitido también desarrollar modelos que superan a los anteriores al capturar patrones complejos en datos musicales. Aunque se ha avanzado mucho en la transcripción de música instrumental, la transcripción de voces sigue siendo muy poco explorada [4, 3, 1, 5] debido a la inestabilidad de los tonos vocales y la dificultad de obtener datos precisos. Por esta razón, hay una necesidad de diseñar herramientas basadas en voz (como es el caso de los *tarareos*), para mejorar la precisión de los sistemas de transcripción vocal.

Este proyecto se centra en abordar estos desafíos específicos al desarrollar un sistema de transcripción de voz a partitura, con un enfoque en etapas separadas para la transcripción en tiempo y detección de tonos.

4. Problema

4.1. Contexto

Actualmente, el proceso de transcribir frases o motivos expresados mediante *tarareos* a papel, se realiza de forma tradicional; es decir, manual e intensivamente. Ya que este proceso aparece continuamente en las distintas fases del proceso de composición, las técnicas actuales suponen un problema, pues conllevan invertir considerablemente más tiempo en esta tarea mecánica que en el proceso creativo. Esto sucede mayormente en el ámbito de la música emergente, *amateur*, e informal; aunque también sucede en la música académica en menor medida.

El problema de la transcripción de motivos expresados mediante *tarareos* al papel se puede reducir al de interpretar características en señales de audio habladas para detectar regiones como:

- Ataques y decaimientos
- Formantes
- Ritmo, tiempo y cadencia
- Frecuencia fundamental

El sonido natural está compuesto de una combinación de múltiples alturas –que llamaremos tono por ahora– el cual puede tener origen en distintos factores: Muchas notas ejecutadas al mismo tiempo, una nota con distintos sobretonos o armónicos, o una combinación de éstos. Las técnicas básicas, como el análisis espectral, permiten reconocer elementos como la nota fundamental y reconocer características agregadas basadas en la predominancia –intensidad– de un tono o frecuencia; pero puede requerir un análisis más personalizado sobre cada señal de audio particular.

Por lo anterior, un sistema basado en Inteligencia Artificial puede ofrecer un enfoque *holístico* del sonido basándose en patrones y reconociendo categorías como las descritas en la lista anterior, superando problemas como las diferencias en el timbre de notas distintas ejecutadas por personas distintas.

4.2. Técnico

El problema consiste en analizar pistas de audio arbitrarias de duración T para encontrar patrones de *tarareos* y entregar su ubicación temporal y tono asociado.

Para ello se escoge una representación del objeto de estudio *pista de audio* como una secuencia de N intervalos regulares, que a su vez tienen M características que los describen.

Las N segmentaciones buscan discretizar la pista de audio para permitir su análisis como secuencias de características agregadas y poder estudiar los cambios entre segmentos contiguos. Este enfoque se puede observar en la Figura 1, donde la señal se divide en intervalos regulares que agrupan características tanto cíclicas, como singulares entre ellas. Por ejemplo, la sección intermedia entre trenes de ondas conserva la frecuencia fundamental, pero cambia aspectos de amplitud y timbre.

Se requiere, por tanto, encontrar el subconjunto dentro de las características M de los intervalos, que mejor describa los cambios generados en un *tarareo*. Entre el conjunto general de características se tiene:

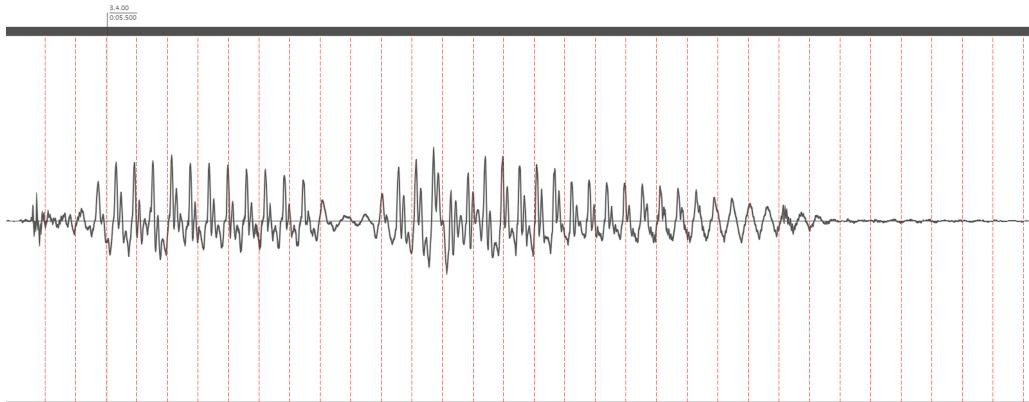


Figura 1: Visualización de la discretización en intervalos regulares de las señales de audio.

- **Amplitud (dB):** La magnitud máxima de la vibración de la onda sonora. Su rango de valores típicos va desde $-\infty$ hasta 0 dB.
- **Frecuencia Fundamental (Hz):** La frecuencia más baja de la onda, que corresponde al tono fundamental. Se puede obtener mediante análisispectral, como la transformada de Fourier.
- **Energía Espectral (dB):** La energía para diferentes bandas de frecuencia. También se obtiene mediante análisispectral, por lo que se debe determinar qué tan específico se realizará (o si es necesario) en vista de no sacrificar eficiencia.
- **Variabilidad Temporal:** La cantidad de cambio o variación en la señal de audio, medida como la desviación estándar o varianza de la amplitud.
- **Relación Señal-Ruido (dB):** La relación entre la potencia de la señal de interés y la potencia del ruido de fondo.
- **Duración de los Segmentos Silenciosos:** La duración de los intervalos de silencio en la señal de audio. Se puede calcular definiendo una precisión de amplitud en las ondas.

En la Figura 2 se observa un ejemplo de grabación de estudio (alta definición), en el que se etiquetó manualmente algunos *formantes* y *tarareos* realizados. En el se puede observar ciertos patrones y rasgos comunes entre las ocurrencias, como la existencia de pequeños silencios entre formantes y el posterior pico de energía al vocalizar el *tarareo*. Son estas características las que se busca encontrar automáticamente para pistas de audio no ideales o con ruido, de manera que no dependa explícitamente del timbre de voz o particularidades en el pitch de las voces, sino únicamente en la relación de cambio que existe entre los segmentos.

En cuanto al reconocimiento de la nota, se puede abordar de manera clásica (con análisispectral) o bien, identificando la similitud de los intervalos con el típico generado por una nota ideal (pitch único). Las categorías básicas para el tono son las dadas por la escala cromática ($C, C\#, D, D\#, E, F, F\#, G, G\#, A, A\#, B$) y un número natural n que corresponde a un identificador de la altura. En la Figura 3 se muestran dos trenes de onda para notas distintas, la decisión sobre si un intervalo es de algún tipo dependerá de los atributos del intervalo. Notar que en la figura también aparecen líneas horizontales como etiquetas, estas corresponden a una clasificación binaria sobre si el intervalo contiene un *tara* o no.

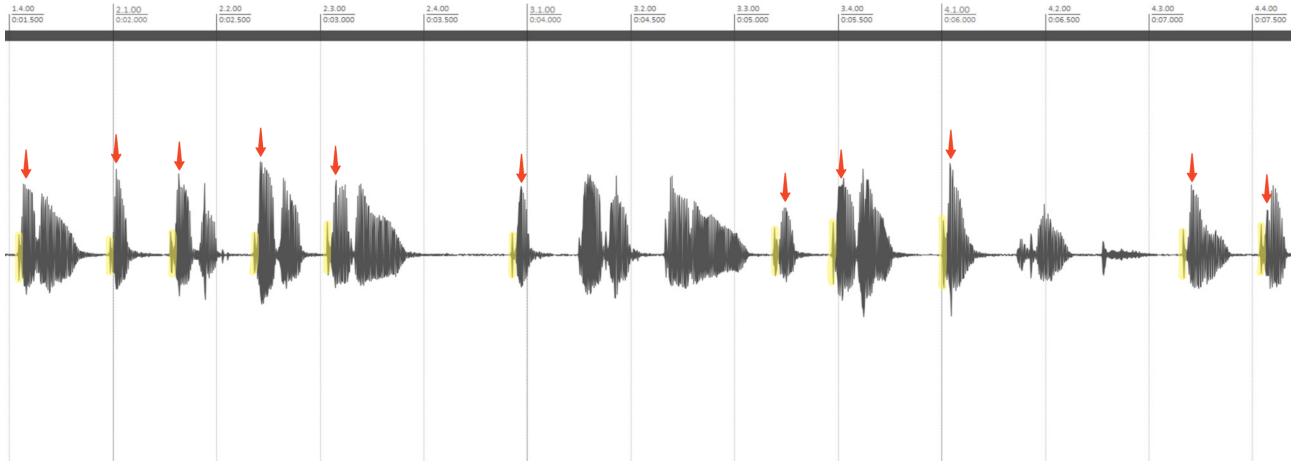


Figura 2: Señal de audio profesional que muestra la ubicación ideal de los *tarareos*. Las secciones amarillas muestran formantes del habla, y las flechas rojas el pico de amplitud al vocalizar un *ta*.

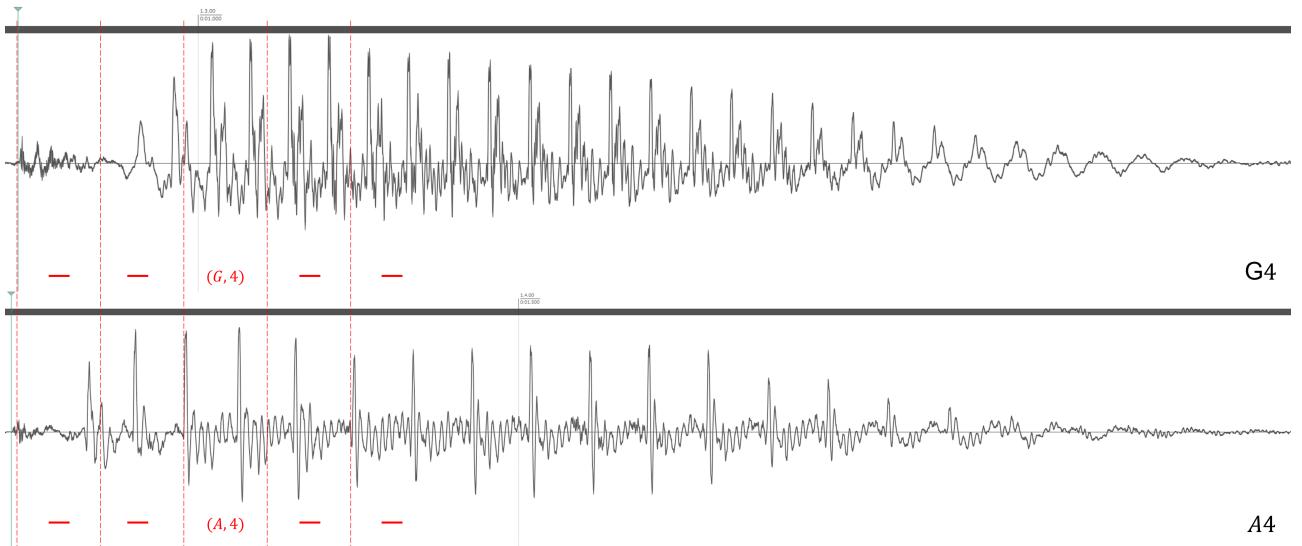


Figura 3: Comparación de la secuencia resultante de un tren de ondas para dos tipos de notas distintos: *A4* (La 4) y *G4* (Sol 4).

4.3. Sistema artificial

El *tarareo* es una actividad que el humano puede reconocer fácilmente, sin necesidad de hacer análisis de Fourier, o aplicar métodos complejos. Sin embargo –quizás de manera inconsciente– hay ciertas características que tomamos en cuenta al momento de analizar la estructura de un *tarareo*: El ataque de la sílaba *Ta*, la aspiración entre fonemas, la variación en el tono y la intensidad de la pronunciación. Estas son características que se pueden detectar en una secuencia de fonemas pronunciados, y que podrían tener propiedades similares que es posible identificar en un señal de audio.

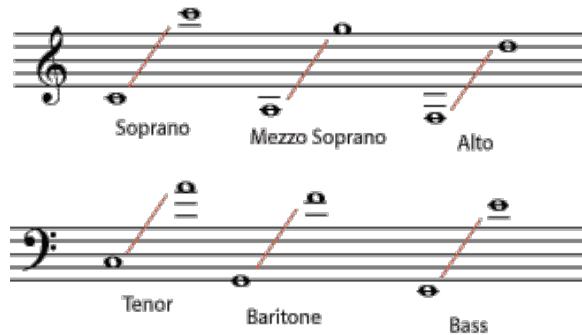
Para su análisis es posible discretizar la señal y posteriormente explorar el espacio de señales discretas, analizando las características de las secciones vecinas para identificar estos marcadores –ataques, variaciones de tono, intensidad– que sugieren la presencia del fonéma unitario del *tarareo*. También es posible crear "ventanas" de secciones de la señal para analizar a un nivel

de agregación más alto, para eliminar ruido de ser necesario.

5. Planeación y diseño

5.1. Conceptos relevantes

- **MIDI (Musical Instrument Digital Interface;** Es un protocolo de comunicación que permite que instrumentos musicales y computadores se comuniquen entre sí.
 - **Tono:** En el trabajo actual se define como el sobrenombre que representa al conjunto de notas de la misma altura relativa en la escala musical (Do, Re, Mi, . . . , Si).
- $$T = \{C, C\#, D, D\#, E, F, F\#, G, A, A\#, B\}$$
- **Nota:** Es una frecuencia particular del espectro disponible en el estándar MIDI. Se representa como un Tono y su altura (octava). Ejemplo: $C3, A\#5, G6$.
 - **Nota fundamental:** Nota más grave de un acorde o composición de notas. Es la que tiene la mayor energía.
 - **Timbre:** Característica que distingue a una voz de otra. Es una mezcla y configuración específica de frecuencias o tonos.
 - **Tesitura:** Intervalo de sonidos de frecuencia determinada en el que una voz humana se expresa más cómodamente.



5.2. Sistemas existentes

- **Singing voice melody transcription using deep neural networks. [3]** Este sistema aborda el problema de la *Recuperación de información musical* enfocado a extraer la voz de música polifónica; es decir, señales de audio con distintos tonos y ejecuciones simultáneas.

Su acercamiento consiste en analizar el espacio de frecuencias vs. ventanas de tiempo (espectrograma), lo cual induce una serie de tiempo multicanal (tantos canales como ventanas de frecuencia). Esta secuencia es entonces alimentada a una arquitectura (DNN) que consiste en 2 capas LSTM Bidireccionales, cuya salida es la estimación de la señal; cada recursión de la red genera una estimación de la señal para el tiempo t . Además cuenta

con un sistema secundario similar al primero, pero cuyo set de datos fue transformado para realzar frecuencias que contengan la melodía en voz.

El sistema **no resuelve** precisamente el mismo problema, pero plantea un avance interesante sobre cómo reconstruir señales de audio de distintos tonos. Además propone analizar directamente el espectrograma de voz realzando las frecuencias relevantes para la voz.

- **Melody transcription via generative pre-training [1]** Este sistema busca resolver el mismo objetivo que el anterior, pero no limitaod a voz, sino a reconocer cualquier melodía en cualquier grabación de música (principalmente aquellas que fueron generadas con muchos instrumentos). La parte generativa del trabajo busca generar música con distintos instrumentos no explorados para extraer la melodía de ellos. Este acercamiento se realizó por la falta de datos de música ejecutada por distintos instrumentos.

El sistema utiliza el modelo generativo "Jukebox" para generar representaciones de música. Después toma la media en ventanas de un 16vo de negra (esto es mucho más pequeño que una semicorchea). El resultado alimenta a un transformer para detectar *onsets* (ataques de nota) en el espacio de ventanas temporales y el resultado es traducido a MIDI.

Este sistema contribuye con la idea de la agregación de características en ventanas de audio con duración equivalente a una fracción de nota. En ese sistema se *promediaron* las ventanas temporales, mientras que este proyecto propone una capa convolucional para extraer características relevantes. Es similar, sin embargo, en la traducción MIDI.

- **Transcription of the Singing Melody in Polyphonic Music [4]** El sistema propone la detección y transcripción de melodías cantadas sobre bases musicales de múltiples tonos (polifónicas).

Su acercamiento consiste en dos etapas:

- Modelo musicológico: Filtrado de rangos vocales, estimación de la nota musical y selección de la nota más probable. Utiliza u sistema de lenguaje para aproximar la nota más probable, el rango de notas de la melodía y probabilidades de transición.
- Modelo acústico: Utiliza cadenas ocultas de Markov y un modelo de Mezclas Gaus-sianas para encontrar los silencios entre notas.

Con ambas partes puede reconstruir una secuencia de notas (modelo musicológico) y silencios (modelo acústico).

Este sistema es similar en cuanto a la separación en etapas, principalmente en la etapa de reconocimiento de nota. Sin embargo, su acercamiento mediante el reconocimiento de silencios con cadenas ocultas de Markov y GMM es distinto al utilizado en este proyecto.

5.3. Herramienta de IA

El problema plantea el análisis de audio y transcripción a MIDI (partitura). Esto sugiere estudiar los siguientes elementos:

- Señal de audio
 - Notas / frecuencias
 - Figuras musicales / ubicación temporal

- Secuencia MIDI (etiquetas)

Es decir, los audios cuentan con dos dimensiones de interés: **frecuencia** y **tiempo**. Las secuencias MIDI no necesitan la dimensión de frecuencias para clasificar sus notas, pues ya están etiquetadas. Esto plantea la siguiente transformación:

$$Fr \times T \rightarrow Cr \times T,$$

donde Fr es el espacio de frecuencias, T el espacio de ventanas discretas de tiempo y Cr el espacio de notas MIDI (que va de 0 o C-1 hasta 127 o G9).

Esta transformación es similar a un *traductor automático* en el que una secuencia de tokens es transformada en otra secuencia de tokens en otro idioma (Fig. 4), con la diferencia de que, en este caso, la dimensión Fr toma valores continuos y el vocabulario en un traductor está limitado.

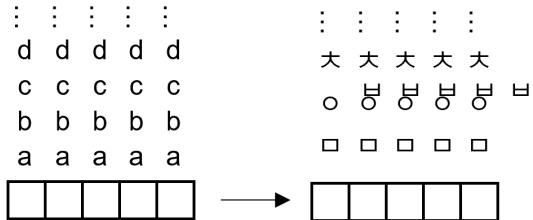


Figura 4: Sistema automático de traducción, esquema general.

El carácter de secuencia de los datos de entrada y los datos de salida sugiere una arquitectura tipo **Secuencia a secuencia**, donde la secuencia de entrada tiene distintos canales (espacio Fr).

Por otro lado, la voz es un fenómeno natural particular ya que varía de persona a persona. La particularidad de la voz es que, salvo en casos excepcionales, **no es polifónica**; es decir, sólo está compuesta de un sólo tono a la vez. Sin embargo, este tono se repite en octavas (armónico) con diferente porcentaje de contribución según el timbre de la persona. Esta variación natural, aunado al ruido de los audios, puede hacer que sea complicado homologar la contribución de los armónicos en una nota ejecutada para casos generales y, con ello, extraer la nota fundamental.

Por lo anterior, se propone separar la dimensión Fr y utilizar **algoritmos de clústering** aprovechando que las notas de la voz natural están separadas por octavas. Con una clusterización sencilla, es posible escoger aquel grupo con el centro en la frecuencia más baja (nota fundamental) para finalmente *redondear* a la nota más cercana en la escala cromática.

5.3.1. Metodología

Se seguirá una metodología **KDD (Knowledge Discovery in Databases)** en el proyecto. Aunque esta es una metodología propia de la ciencia de datos, propone iteraciones y ajustes en todos los pasos con el fin de optimizar la adquisición de conocimiento; en este caso, de transcribir correctamente los *tarareos*. Permite regresar incluso a la etapa de selección de muestras para refinar o restringir el sistema si es necesario. Los pasos se muestran en la Figura 5.

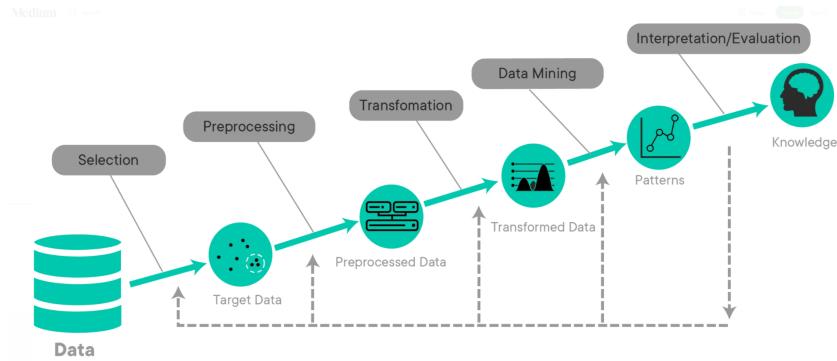


Figura 5: Esquema de metodología KDD. Tomada de [https://medium.com/@aj.ramirez23/3-most-popular-data-science-methods-e61f6600b83f].

El proyecto se compone de distintas partes (Figura 6) para un **análisis** complementario de los datos MIDI y tarareos (audio); es decir, que los resultados de uno influyan en el otro, pudiéndose repetir de forma recursiva hasta codificar los datos de manera óptima. Posteriormente en la fase de **transformación** se implementan los parámetros y salvaguardas obtenidas en el análisis para generar los datos para alimentar a los sistemas que, finalmente, se compondrán por dos:

- **Sistema detector de ejecución**: analiza la intensidad de onda en la dimensión temporal T .
- **Sistema detector de notas**: busca la nota fundamental en espacios de frecuencia restringidos a ventanas $t_0 \in T$, es decir en $\text{Fr} \times T|_{t_0}$.

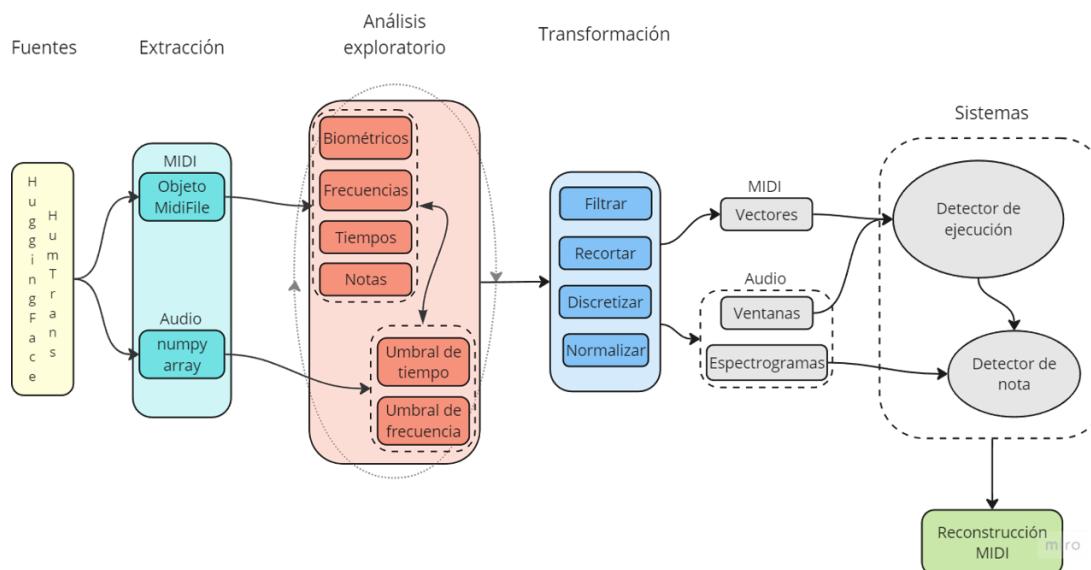


Figura 6: Pipeline del proyecto de datos y sistemas de Inteligencia Artificial.

Finalmente los sistemas combinan sus resultados para obtener una secuencia MIDI con notas, su duración y silencios.

5.4. Alcance

A la luz del estado del arte en sistemas de transcripción de melodías a partitura y, en específico, a las arquitecturas complejas propuestas para el análisis de audios en etapas separadas

[1], el alcance del proyecto es puramente personal.

Ryynänen y Klapuri proponen un sistema sofisticado basado en silencios para la detección de ejecución de notas, compuesta por dos componentes: cadenas de markov y mezclas Gaussianas. Este proyecto propone una red neuronal profunda de secuencia a secuencia. El alcance en esta etapa es poder extraer características significativas de los audios sin necesidad de alimentar al sistema con información de ejecución extra. No se estima que tenga una mayor precisión, pero sí favorable.

Ryynänen y Klapuri también proponen una etapa para la detección de notas que incluye asignación de probabilidades en la dimensión temporal. En este proyecto se propone una clusterización seguida de un promedio, prescindiendo de la parte temporal debido a que se analizan sólo las partes temporales señaladas por la Etapa 1. Al clusterizar sólo la parte de frecuencias, se espera que se beneficie la velocidad y precisión de la extracción de notas.

6. Adquisición y exploración de datos

6.1. Origen

- **Fuente:** HumTrans: A Novel Open-Source Dataset for Humming Melody Transcription and Beyond [2]
- **Autores:** Shansong Liu, Xu Li, Dian Li, Ying Shan
- **Fecha:** Septiembre, 2023
- **Información:** 14,610 patrones musicales tarareados por 10 músicos, entre hombres y mujeres. El conjunto contiene también repeticiones de los mismos patrones. Está compuesto por archivos .wav (tarareos) y archivos MIDI (etiquetas). La duración total es >57 horas y su peso es >15gb.

6.2. Descripción

Las características de los datos en Python y su forma de importación es la siguiente:

	Tarareos	Etiquetas MIDI
Formato	.wav	.mid
Librería	librosa	mido
Tipo de dato en Python	numpy.ndarray	mido.midifiles.MidiFile

▪ Tarareo:

Array de dimensiones $(n,)$ donde n es el número de muestras totales del audio. También incluye la frecuencia de muestreo en Hz (sr) que expresa el número de muestras por segundo.

Ex. Un audio con $sr = 22050$ y longitud $n = 44100$ durará 2 segundos.

La amplitud del audio se codifica en los valores del array. Toma valores en \mathbb{R} donde las amplitudes crecen según la distancia al cero.

Ex. Una amplitud de -1 es igual de tenue que una de 1.

▪ MIDI:

Un archivo MIDI se compone de mensajes secuenciadas en unidades internas que dependen de parámetros generales de tiempo, afinación y estructura musical.

Los parámetros de interés para el análisis son:

- **tempo:** $[\mu s/beat]$ donde el beat es la unidad de tiempo del compás; por ejemplo, en un compás de 4/4 la unidad será el *cuarto* o negra.
- **ticks:** Es una unidad interna que representa subdivisiones del beat.
- **ticks_per_beat:** $[ticks/beat]$ cuántas subdivisiones tiene un beat internamente (precisión del MIDI).

El MIDI se compone de pistas (tracks), mensajes y metamensajes. Los MIDI en el dataset se componen de un único track y la información de las notas se almacena como mensajes con atributos:

- **type**: 'note_on' / 'note_of'
- **velocity**: describe la fuerza del ataque de la nota, un *velocity* = 0 implica que no se ejecuta nada.
- **note**: describe el código de nota del 1 127 que representan Do-1 a Sol9 (C-1 a G9).
- **time**: tiempo en ticks después del cual sucede el mensaje.

6.3. Análisis exploratorio

El análisis exploratorio se realizó en dos conjuntos de datos simultáneos: MIDI y Audios. Se comenzó explorando el conjunto **MIDI** y, según sus resultados, se analizó lo necesario del conjunto **Audios**. Este proceso se realizó múltiples veces para extraer las características adecuadas para el proyecto. El proceso general fue el siguiente:

MIDI

- Importación de archivos MIDI
- Extracción de información
- Visualizar y sistetizar datos
- Caracterizar y filtrar datos

Audio

- Importación de archivos de Audio
- Explorar espectrogramas
- Analizar límites extraídos del MIDI

6.3.1. MIDI

Los archivos MIDI se importaron usando la biblioteca **mido** en Python, lo cual genera un objeto "MidiFile" que es un iterador de "tracks"(pistas) y mensajes. Luce como el siguiente:

```

1 MidiFile(type=1, ticks_per_beat=1024, tracks=[  

2     MidiTrack([  

3         MetaMessage('set_tempo', tempo=600000, time=0),  

4         MetaMessage('key_signature', key='C', time=0),  

5         MetaMessage('time_signature', numerator=3, denominator=4,  

6         clocks_per_click=24, notated_32nd_notes_per_beat=8, time=0),  

7         MetaMessage('end_of_track', time=1024)]),  

8     MidiTrack([  

9         MetaMessage('track_name', name='é\x92fc\x90', Piano', time=0),  

10        Message('program_change', channel=0, program=0, time=0),  

11        Message('pitchwheel', channel=0, pitch=0, time=0),  

12        Message('program_change', channel=0, program=0, time=0),  

13        Message('program_change', channel=0, program=0, time=0),  

14        Message('note_on', channel=0, note=52, velocity=81, time=2560),  

15        Message('note_off', channel=0, note=52, velocity=0, time=512),  

16        Message('note_on', channel=0, note=57, velocity=101, time=0),  

17        Message('note_off', channel=0, note=57, velocity=0, time=1536),  

18        Message('note_on', channel=0, note=57, velocity=81, time=0),  

19        Message('note_off', channel=0, note=57, velocity=0, time=512),

```

```
20     Message('note_on', channel=0, note=59, velocity=81, time=0),
21     Message('note_off', channel=0, note=59, velocity=0, time=768),
```

Lo principal fue analizar la consistencia de la estructura de datos del MIDI. Esto es, que todos los documentos cumplan con la forma **estándar** del MIDI, el cual cumple:

- ‘note_on’ para marcar la ejecución de la nota ($velocity > 0$).
- ‘note_off’ marca la finalización de la nota ($velocity = 0$) y expresa la duración de la nota ($time > 0$)
- Si ‘note_on’ tiene un $time > 0$ indica silencio antes de ejecutarse.

Los MIDI no estándar comienzan y terminan las notas con ‘note_on’, además seccionan la duración de las notas en distintos mensajes. Se encontró características particulares de estos dos tipos de MIDI:

	count	ticks
Estándar	10053	1024
No estándar	4561	480

Los MIDI estándar y no estándar tienen una cantidad fija de ticks de precisión. Debido a que el tipo **estándar** es el que mejor representa la intuición de *ejecuta* y *silenciar* notas, entonces se decide trabajar con el subconjunto estándar. Habiéndo dicho esto, también se programaron rutinas para estandarizar el subconjunto restante, sin embargo, debido a que sus MIDI tienen el problema de seccionar las notas, es imposible reconstruir el MIDI original sin información extra sobre la figura musical mínima (negra, corchea, semicorchea, etc.), lo que lleva a un problema cuya solución está fuera del objetivo de este proyecto.

A partir del dataset filtrado, se escribieron rutinas para extraer información relevante: [ubicación: utils/preprocess.py]

- Biométricos: Género, IDs (persona, melodía, segmento, repetición).
- Tiempo: Duración total, mensaje más corto, tempo mínimo
- Musicalidad: Figura mínima (corchea, negra, etc.), mínimo y máximo cifrado (C, C#, D, D#, etc.), número de notas por tono.
- Frecuencias: Frecuencia mínima y máxima, y su equivalente nota mínima y máxima.

Para lograr un sistema que capture todas las variaciones de timbres, diferencias entre tonos y sus octavas para cubrir todos los distintos espectros de la voz humana posible, se analizó el origen de las muestras.

Se encontró que la distribución de sexo es homogénea (Figura 7) lo que permite analizar espectros graves y agudos de la voz humana. La distribución de voz masculina y femenina se refleja en la distribución de notas y sus octavas (Figura 8).

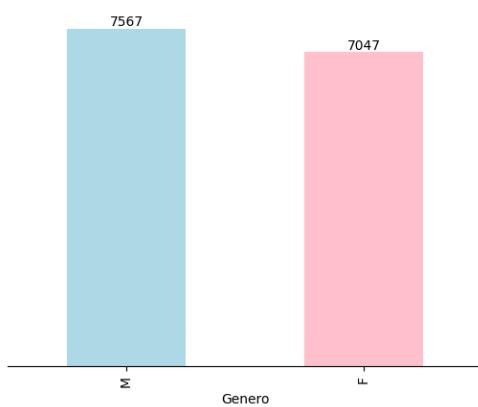


Figura 7: Distribución de sexo de las personas que generaron las muestras.

	-1	0	1	2	3	octava	4	5	6	7	8	9
nota	A	0	0	154	3632	24768	29064	918	0	0	0	0
D#	A#	0	0	43	624	4800	5490	143	0	0	0	0
B	B	0	0	121	1992	13866	11083	150	0	0	0	0
C	C	0	0	0	431	4449	35880	14830	104	0	0	0
C#	C#	0	0	0	83	791	6388	2290	35	0	0	0
D	D	0	0	17	850	5559	41853	10139	23	0	0	0
D#	D#	0	0	0	220	905	7650	1055	0	0	0	0
E	E	0	0	29	1314	7731	44711	6818	5	0	0	0
F	F	0	0	0	836	3836	18452	1543	0	0	0	0
F#	F#	0	0	37	520	2488	8962	787	0	0	0	0
G	G	0	0	81	2498	14854	35963	2255	0	0	0	0
G#	G#	0	0	5	377	2404	4445	244	0	0	0	0

Figura 8: Distribución de las notas de la escala cromática y sus octavas.

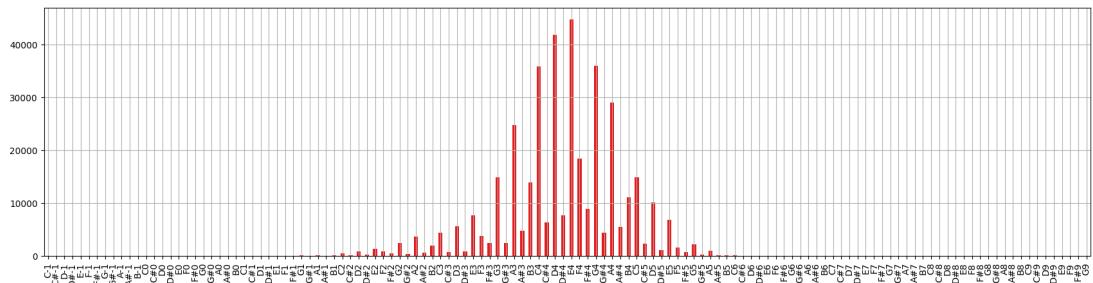


Figura 9: Distribución normal de los tonos disponibles en el dataset.

El mapa de calor sugiere una distribución normal de las voces que es exactamente lo que se esperaría de la población en general; una mayor concentración en el rango de $C4$ a $A4$ y no encontrando ninguna voz tan grave como la altura $< B1$ ni tan aguda como $> D6$.

La distribución de tonos (Figura 9) se puede separar en la tesitura, si se desea reducir el número de datos para ajustarse a un tipo de voz específico. En este caso, la tesitura *alto* mostró tener la distirbución menos sesgada (Figura 10)

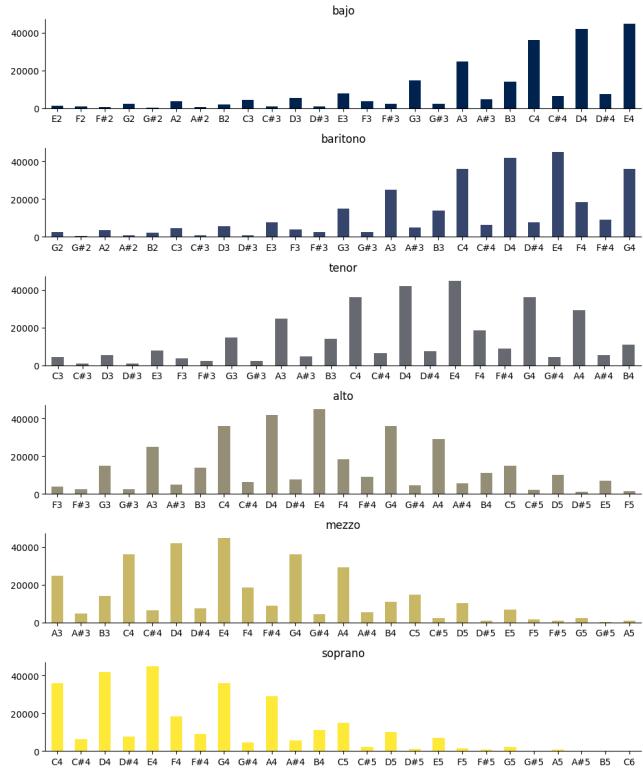


Figura 10: Distribución de tesituras de las notas disponibles en el dataset.

Respecto al tiempo de duración de las muestras, se analizaron sus estadísticas para determinar un tiempo fijo para el entrenamiento del sistema. Esta necesidad surgió de que la frecuencia de muestreo de origen de los audios es de 44100Hz , lo que resulta en casi 1 millón de muestras para audios de 20 segundos. Independientemente de la longitud de las ventanas con las que se discretizará el audio, este número de datos permanecería constante. Por eso se buscó la necesidad de reducir a un tiempo prudente, el cual resultó ser de **10 segundos** ya que es el número cerrado más cercano a la media de duración, y el que mostró mejores resultados del análisis de audios (el cual se muestra en la siguiente sección).

count	mean	std	min	25 %	50 %	75 %	max
14614	14.3149	4.5898	4.9375	10.6250	13.2857	17	30.4687

Finalmente, ya que se busca discretizar a las muestras en el tiempo, se propuso un tiempo **2 veces más rápido que la nota más corta**. Así que, a partir del subconjunto de MIDIs extraído, se encontró la duración de la nota más corta como:

$$\text{lowest_time} = 0.125s,$$

por lo que la ventana temporal es función de este tiempo, como $L(n) = \text{lowest_time}/n$.

6.3.2. Audios

Para el análisis de audios, primero se visualizó el audio para dar una idea de la forma del array de datos importado por librosa (Figura 11)

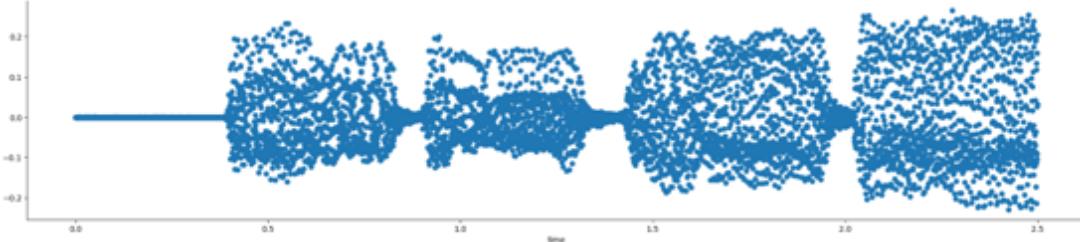


Figura 11: Primeros 2.5 segundos de una muestra de tarareo.

Lo anterior muestra que, si bien librosa representa el array del audio como una *onda*, en realidad es una matriz con datos dispersas y cuya ubicación en la matriz y dispersión es relevante. Ese *percepción* ayudará a definir el tipo de arquitectura que se requiere.

Respecto al análisis en cada dimensión del audio (frecuencia y tiempo), se realizó después de analizar las cotas y características que el MIDI reproduce de los tarareos. Por lo que el análisis se enfoca en reducir la dimensionalidad a partir de explotar las limitaciones de la voz (no es necesario analizar frecuencias extremas).

NOTA: Para la conclusión de este análisis se requieren resultados de la transformación de MIDIs. En particular, la mínima y máxima frecuencia de la voz humana: $\text{Hz}_{\min}, \text{Hz}_{\max} = (18, 1167)$.

- **Frecuencia:** Se representó el audio con su espectrograma (dimensiones (Fr, T)). Ya que esta dimensión se utilizará en el **Sistema detector de notas**, entonces es conveniente priorizar la resolución de los datos en esa dimensión al mismo tiempo que se reduce la dimensión. Por esa razón, se analizaron distintos métodos de representación:

	Detalles	sr	freq dim	tiempo dim
Muestra original	Señal de origen como comparación	44100	1025	1723
Resampleado	2 veces la máxima frecuencia Hz_{\max}	2334	1025	92
Slicing	Recortar el espectrograma original	44100	53	1723

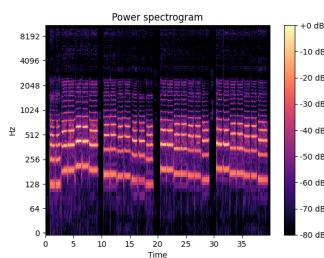


Figura 12: Espectrograma de "muestra original".

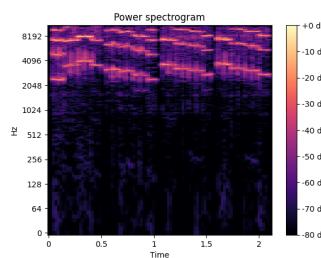


Figura 13: Espectrograma de Resampleado".

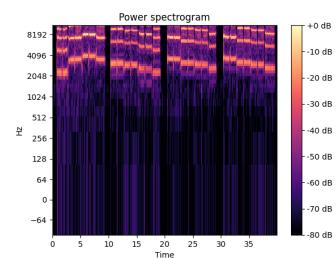


Figura 14: Espectrograma de "Slicing".

La Figura 14 que corresponde a recortar el array de frecuencia de meustreo original, se ve mejor y reduce ampliamente la dimensión de frecuencias ($1025 \rightarrow 53$), así que se escogió ese método para reducir la dimensión de frecuencias.

- **Tiempo:** El análisis en este caso es sobre los audios raw, con el propósito de transformarlos a frames de la forma (muestras, ventanas) con dimensiones adecuadas en el tiempo.

En el análisis MIDI se concluyó una ventana de tiempo de longitud $L = 0.125s/2$ por lo que se probó para distintos recortes de los audios a tiempos D :

- 5 segundos: Resulta en una representación de (2756, 80) dimensiones.
- 10 segundos: Resulta en una representación de (2756, 160) dimensiones.
- 15 segundos: Resulta en una representación de (2756, 240) dimensiones.

Se concluyó $D = 10$ segundos, tal como el análisis MIDI.

6.4. Transformación

Recordando el objetivo de la **sección 5.3**, se desea trabajar con 2 sistemas por separado. Un sistema detector de ejecución que analiza la dimensión **Temporal** del audio y resuelve etiquetas MIDI, y un sistema detector de notas que analiza la dimensión de **Frecuencias** para las ventanas temporales del sistema anterior. Por lo tanto, el audio necesita dos salidas:

- Espectrograma: Enfocada a las frecuencias.
- Frames temporales: Señal natural de muestras e intensidad, discretizada en ventanas.

Mientras que el MIDI "sólo" debe ser transformado a una secuencia/vector. En ambos casos y con base en el análisis, se determinaron las siguientes condiciones:

1. Recortar silencios iniciales
2. Recortar a $D = 10$ segundos.
3. Eliminar notas sonando al segundo $t = 10$.
4. Capturar el tiempo t_f en la que termina la última nota ($t_f \leq 10$).
5. Normalizar intensidades y utilizar la escala de intensidad natural.

6.4.1. MIDI

Para el tratamiento de los MIDI se realizaron rutinas de lectura y transcripción a vector [ubicación: `utils/midi.py`]. Que se dividen en dos: Reducción de la dimensión del tiempo (para cumplir las condiciones) y transformar a vector.

- **Reducción de dimensión de tiempo:** Se evaluó si el mensaje tenía silencios al inicio (`time > 0` en el primer '`note_on`') y se eliminó recursivamente en el archivo `MidiFile`:

```

1     Message('note_on', channel=0, note=52, velocity=81, time=2560),
2     Message('note_off', channel=0, note=52, velocity=0, time=512),...

```

Después se generó una función para restringir el tiempo del MIDI a $D = 10$ segundos eliminando la última nota truncada. Para ello se siguió el algoritmo:

Algorithm 1 Limitar MIDI a D segundos

```

1: Transformar  $D = 10s$  a ticks (unidad interna)
2: total_ticks  $\leftarrow 0$ 
3: new_track  $\leftarrow$  MidiTrack()
4: for mensaje  $\in$  archivo MIDI do
5:   if tipo del mensaje = "note_on"  $\vee$  "note_off" then
6:     total_ticks += ticksmensaje
7:     if ticksmensaje  $\leq D_{ticks}$  then
8:       añadir mensaje a new_track
9:     end if
10:   end if
11: end for
12: if el último mensaje es "note_on" then
13:   eliminar el último mensaje
14: end if

```

- **MIDI 2 Vector:** Los MIDI resultantes del paso anterior fueron alimentados a una rutina de transformación a lista. Ya que la dimensión temporal busca encontrar momentos donde se ejecuta una nota, se reconocieron 3 eventos escenciales:

Código	Descripción
0	Silencio
1	Ejecución de nota
2	Nota sostenida

Por lo que el vector busca codificar los mensajes MIDI a 0, 1 y 2's.

Dada la longitud de ventana $L = \text{lowest_time}/2$, el MIDI almacenará su significado temporal en el orden de las celdas y los eventos como el contenido de las celdas (Figura 15). Se realiza un ajuste *hacia atrás* tal que la duración de nota culmine al final de celda.

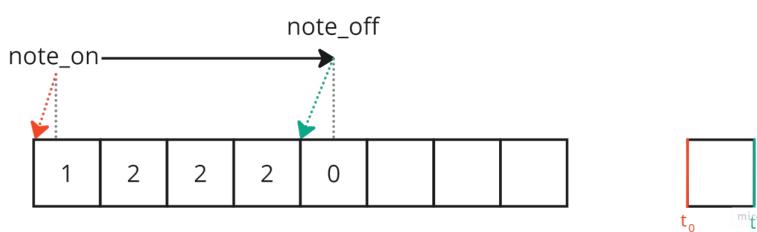


Figura 15: Diagrama de la asignación de mensajes MIDI a vector.

Para lograr esta codificación, se realizó un algoritmo que tratara todos los casos posibles de manera distinta: cuando se ejecuta una nota, cuando hay silencios antes de que se ejecute una nota, y cuando la nota está sonando. El algoritmo se resume en calcular cuántas unidades de L permite el mensaje MIDI y entonces asignar un código a las casillas respectivas:

Algorithm 2 Rutina MIDI 2 Vec

```

1: vector: lista
2:  $L$ : longitud de ventana
3: for mensaje  $\in$  archivo MIDI do
4:   if mensaje = 'note_on' & velocity > 0 & time= 0 then            $\triangleright$  Notas inmediatas
5:     vector  $\leftarrow$  1
6:   end if
7:   if mensaje = 'note_on' & velocity > 0 & time> 0 then            $\triangleright$  Notas con silencio inicial
8:     silencios =  $\lfloor time/L \rfloor$ 
9:     vector  $\leftarrow 0 \times$ silencios
10:    vector  $\leftarrow$  1
11:   end if
12:   if mensaje = 'note_off' & velocity = 0 & time> 0 then            $\triangleright$  Notas sonando
13:     tiempo restante =  $time - L$ 
14:     sonando = tiempo restante / $L$ 
15:     vector  $\leftarrow 2 \times$ sonando
16:   end if
17: end for
  
```

6.4.2. Audio

En el caso del audio se realizó un proceso similar pero sin necesidad de una codificación compleja [ubicación: utils/tarareos.py]:

1. **Recorte de tiempo:** Se recortaron los silencios iniciales y finales usando **librosa.trim** con un umbral de 55 decibeles (media de intensidad en silencios). Después se recortó los audios a $D = t_f$ donde t_f es el tiempo final del MIDI recortado que sigue las condiciones de transformación.
2. **Normalización:** Se determinó usar la intensidad cruda del audio, ya que su distribución tiene sesgos significativos cuando una nota se ejecuta; a diferencia de escalas en *decibeles* (logarítmica) que suavizan los cambios grandes. Luego se normalizó para que la intensidad estuviera en un rango de $[0, 1]$.
3. **Frames temporales:** Se dividió el audio de M muestras, en N ventanas de L' muestras, resultando en una matriz de (N, L') (de ahora en adelante se usará solo L en el entendido de que se refiere a la longitud de ventana, sea en segundos, muestras o ticks).
4. **Espectrogramas:** Se realizaron con la STFT (short-time Fourier transform) con un kernel de longitud L y stride L . Recortando las frecuencias al rango (18, 1167) como se encontró en el análisis.

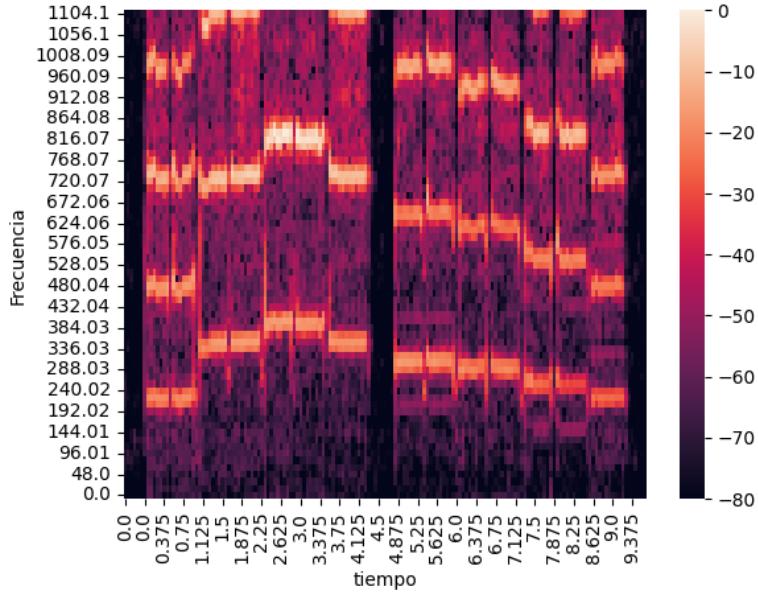


Figura 16: Espectrograma procesado con los lineamientos.

6.5. Datos preprocesados

Los datos finales después de procesar son los siguientes:

	Vectores MIDI	Frames Audio	Espectrogramas
Dimensiones	K (variable)	(L,N) N: variable L: 1378	(M,N) N: variable M: 70

K es la longitud del vector MIDI transformado, L es el número de muestras por cada ventana N de tiempo, y M son las ventanas de frecuencia filtradas.

7. Modelado

7.1. Arquitectura

7.1.1. Sistema detector de ejecución

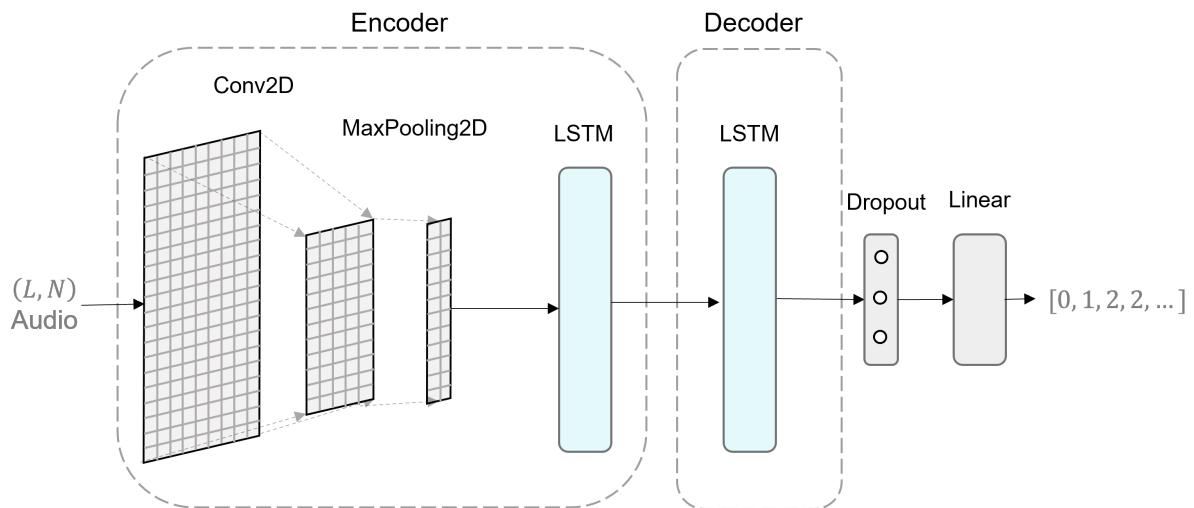


Figura 17: Diagrama de la arquitectura propuesta para construir un detector del tiempo de ejecución de tarareos.

Tal como lo menciona el planteamiento, el problema es análogo a un traductor automático con la diferencia de la dimensión de frecuencias y amplitudes. Es por ello que se propone la arquitectura de la Figura 17.

Pérdida	Parámetros	Dropout	Teacher forcing
CrossEntropy	15,807	0.25	0.5

■ Conv2D:

- **Justificación:** Los frames de audio definen un espacio de dos dimensiones. Este espacio tiene una familia de muestras por ventana de tiempo N . Como se vió en el análisis, los valores de intensidad de tiempos contiguos son dispersos. En realidad, es esta dispersión la que caracteriza a un *tarareo*; cuando no existe dispersión, es porque no hay notas sonando.
Una capa de convolución permite extraer características agregadas de ese *bonche* de valores.
- **Input dimension:** (batch_size, 1, L, N)
- **Canales:** 1
Proponemos un sólo canal para conservar la forma de entrada y optimizar los tiempos de entrenamiento.
- **Kernel:** (86,4)
Se escogió $L//16 = 86$ por que sólo interesan los valores muy dispersos, no es necesario conservar información a un nivel más bajo de agregación. Se escogió 4 en la

segunda entrada ya que las ventanas de tiempo N son 2 veces más pequeñas que la nota mínima; un valor de 4 permite analizar sólo 2 posibles unidades mínimas de audio codificadas.

- **Stride:** (1,1)

Requerimos agregar la información de las muestras **por ventana**, es decir, de la dimensión 1. Un stride de 1 permite al kernel avanzar en las (L) muestras.

- **MaxPooling2D:**

- **Justificación:** Permite concluir la agregación de características de la capa Conv2D. Se utiliza el Max y no un Avg, por que es la máxima amplitud la que diferencia un tarareo de un silencio.

- **Input dimension:** (batch_size,1,L',N')

L' y N' son el resultado de la capa Conv2D. En este caso $L' = 1293$ y N' es de longitud variable.

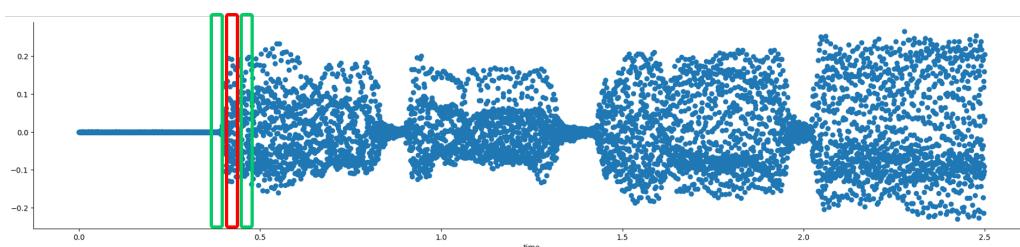
- **Kernel:** (21,2)

En este caso se uso $86//4$ en la dimensión 0 para concluir en 4 características luego de un stride del mismo tamaño del kernel. El 2 en la dimensión siguiente sigue buscando no perder información en esa dimensión.

- **stride:** Igual que el Kernel.

- **Encoder LSTM:**

- **Justificación:** Para un tiempo t es imposible saber sobre si una nota está comenzando, avanzando o por terminar. Se requiere información de los tiempos adyacentes. Una LSTM procesa las secuencias de las ventanas, manteniendo la memoria a lo largo del tiempo para capturar patrones en la secuencia de audio.



- **Input dimension:** (batch_size,N",L")

En este caso $L" = 61$ es la salida del MaxPooling2D. Se tiene que eliminar el canal de apoyo usado en las convolucionales e invertir el orden, de modo que $L"$ sea el número de características en la LSTM.

- **Hidden dim:** 30

Captura solo parte de la representación de muestras.

- **Decoder LSTM:**

- **Justificación:** Ya que necesitamos un sistema de secuencia a secuencia, una LSTM para generar el vector MIDI es lo más adecuado.

- **Input dimension:** (batch_size,N”,encoder_hidden_dim)

En este caso $L'' = 61$ es la salida del MaxPooling2D. Se tiene que eliminar el canal de apoyo usado en las convolucionales e invertir el orden, de modo que L'' sea el número de características en la LSTM.

- **Hidden dim:** encoder_hidden_dim

Debe ser el mismo para poder realizar la recursión.

- **Densa Linear:**

- **Justificación:** Se requiere una capa para convertir la salida del LSTM decodificador en probabilidades paralas etiquetas (0,1,2)

- **Input dimension:** (batch_size,K,encoder_hidden_dim)

7.2. Evaluación

7.2.1. Descripción

A lo largo del entrenamiento, se calculó la pérdida utilizando **CrossEntropy Loss** (Figura 18) además de la perplejidad (PPL) (Figura 19), sobre el conjunto de validación. Se realizó un entrenamiento con las siguientes características:

	n
Épocas	35
Batch size	32
Muestras / batch	220

Para llevar un control de la pérdida, se realizó el entrenamiento en bloques de 20 épocas, guardando el estado del modelo al final de cada época si la pérdida mejoraba respecto al mejor desempeño en memoria.

Luego del entrenamiento y el monitoreo del Loss de entrenamiento y validación, se calcularon las siguientes métricas en el conjunto de prueba:

- Accuracy
- Precision
- Recall
- F1
- Distancia de Levenshtein

Donde F1 se considera la más adecuada debido al desbalance de las clases por ventana (se espera una gran cantidad de 0-silencios y 2-sostenidos con respecto a las 1-ejecuciones). Todas las métricas de clasificación fueron ejecutadas a nivel ventana, es decir, sobre cada entrada de los vectores predichos.

La distancia de Levenshtein es adecuada para comparar dos cadenas de texto y determinar cuántas operaciones básicas (inserciones, eliminaciones o sustituciones de un solo carácter) son necesarias para recuperar una cadena de la otra.

7.2.2. Resultados preliminares

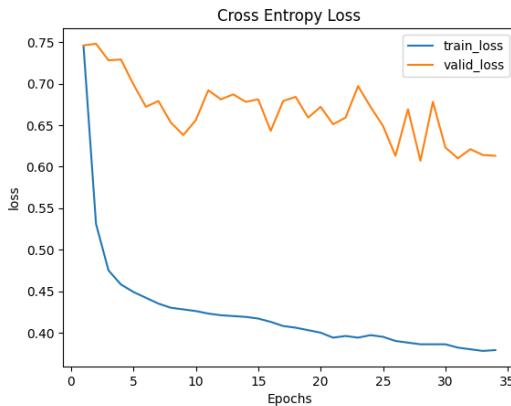


Figura 18: Evolución de la pérdida Cross Entropy.

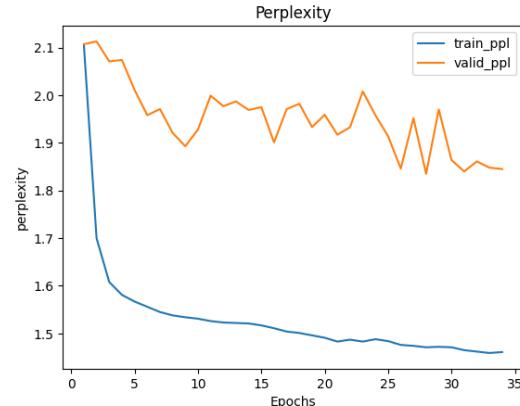


Figura 19: Evolución de la perplejidad.

Los resultados del entrenamiento, muestran una tendencia ligera a la baja de la función de pérdida en el **conjunto de validación**, mientras un rápido descenso en el **conjunto de prueba**. Esto indica un posible *sobreajuste* a los datos de entrenamiento. La red ConvSeq2Seq utilizada fue provista de un **Dropout** de 25 % y un **teacher forcing** de 50 %; incrementar el dropout podría ayudar a que el modelo no se sesgue tan rápido al conjunto de prueba, y permita generalizar en el conjunto de validación. También se podría cambiar el tamaño del batch a uno más pequeño.

Finalmente, esta tendencia baja de la pérdida indica que, a pesar de que esté sobreajustando, aún mejora la predicción de datos no vistos. Esto supone un panorama prometedor para el modelo respecto a la codificación de datos escogidos.

La evaluación en el **conjunto de prueba** arrojó:

Métrica	avg	0	1	2
Loss	0.653	-	-	-
Levenshtein	40.316	-	-	-
Accuracy	0.750	-	-	-
Precision	0.814	1	1	0.75
Recall	0.750	0.05	0.05	1
F1 Score	0.655	0.09	0.09	0.86

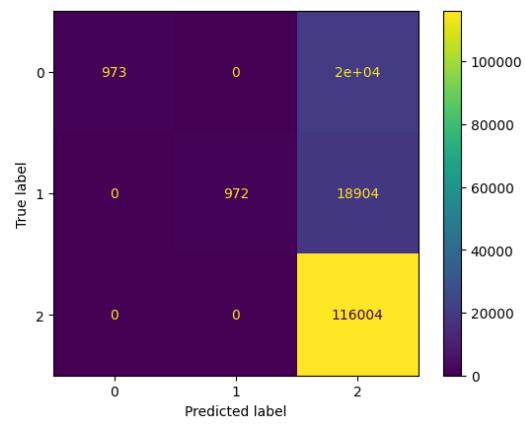
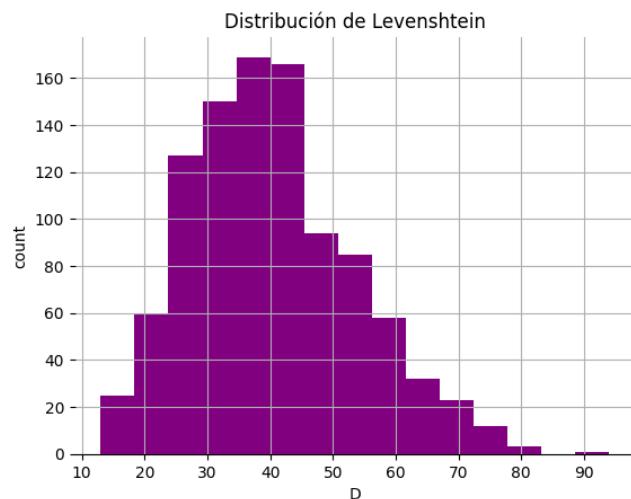


Figura 20: Matriz de confusión de clasificación

A pesar de que el Loss es igual de alto que el encontrado en la validación, las métricas de clasificación muestran un desempeño *aceptable*. Sin embargo, la matriz de confusión muestra un alto número de falsos negativos tanto para las clases 1 (ejecuciones) como 0 (silencios), y un alto sesgo hacia la clase 2 (sostenidos). Es de esperarse ya las melodías tienen en su mayoría

tarareos, pocos silencios, y mucho menos ejecuciones; una forma distinta de codificación en la que el ataque (que es un concepto) no ocupe el mismo nivel que el silencio y sostenido (que son características físicas).

- **Precisión:** Indica que el 81.4% de las ventanas clasifica correctamente las ejecuciones. Es importante ya que las ejecuciones son pocas, y falsos positivos crearía más notas de las que se requieren.
- **Accuracy:** Una muestra general de desempeño, un 75 %.
- **Recall:** Un 75 % de los casos que no son ejecuciones, eran realmente silencios o sostenidos.
- **F1-Score:** En este caso, la métrica de equilibrio sí penaliza el desempeño, mostrando una efectividad del 65.5 %.
- **Avg. Levenshtein:** El promedio de operaciones básicas para pasar de la cadena predicha a la cadena real es de 40.316. Ya que el promedio de longitud de los vectores MIDI es de 160, este es un número considerablemente bajo.



La distribución de las distancias de Levenshtein muestra que muy pocas cadenas tuvieron un valor alto de 90. Este valor es incluso bajo comparado con la longitud de cadena media (160).

8. Conclusiones

- Los archivos MIDI y de audio pueden requerir codificación muy especializada para adecuarlos a los propósitos del proyecto. Los resultados mostraron un alto sesgo a la clase de notas sostenidas, es probable que el tratamiento de datos fallara en colocar a las ejecuciones (1) en una codificación con la misma relevancia que las otras clases.
 - Uno de los problemas más importantes es el de la No estandarización de MIDIs.
 - Es un problema no trivial debido a que subdivide el tiempo de las notas sin información al respecto.

- La transformación a vectores podría cambiarse en una codificación basada sólo en mensajes y sus propiedades.
- El sistema que propone Matti Ryyränen y Anssi Klapuri por etapas sugiere la detección de silencios. El preprocesamiento de datos de este proyecto podría beneficiarse de ese enfoque.
- Con base en los trabajos previos, la arquitectura con redes recurrentes con etapa convolucional es una buena forma de agregar información con ruido.

Finalmente, Xiaochun Yin et al. [6] utiliza un enfoque parecido (CNN + BiLSTM) para tratar series de tiempo con ruido con muchos canales. Ese enfoque se puede recuperar en este trabajo, definiendo los canales como las muestras por ventana. Los datos del proyecto ya son compatibles con la arquitectura CNN+BiLSTM paralela.

El siguiente paso es continuar con la Etapa 2: Detección de notas, y compilar el pipeline realizado (extracción, transformación, limpieza y alimentación del modelo) para generar un sistema único de transcripción. Desarrollar el decodificador de los vectores MIDI y, finalmente, utilizar PyGuitarPro para generar las partituras deseadas.

Referencias

- [1] Chris Donahue, John Thickstun, and Percy Liang. Melody transcription via generative pre-training, 2022. URL: <https://arxiv.org/abs/2212.01884>, [arXiv:2212.01884](https://arxiv.org/abs/2212.01884).
- [2] Shansong Liu, Xu Li, Dian Li, and Ying Shan. Humtrans: A novel open-source dataset for humming melody transcription and beyond, 2023. URL: <https://arxiv.org/abs/2309.09623>, [arXiv:2309.09623](https://arxiv.org/abs/2309.09623).
- [3] François Rigaud and Mathieu Radenac. Singing voice melody transcription using deep neural networks. In *International Society for Music Information Retrieval Conference*, 2016. URL: <https://api.semanticscholar.org/CorpusID:18004775>.
- [4] Matti Ryyränen and Anssi Klapuri. Transcription of the Singing Melody in Polyphonic Music. In *Proceedings of the 7th International Conference on Music Information Retrieval*, pages 222–227. ISMIR, September 2018. [doi:10.5281/zenodo.1418291](https://doi.org/10.5281/zenodo.1418291).
- [5] Xianke Wang, Bowen Tian, Weiming Yang, Wei Xu, and Wenqing Cheng. Musicyolo: A vision-based framework for automatic singing transcription. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 31:229–241, 2023. [doi:10.1109/TASLP.2022.3221005](https://doi.org/10.1109/TASLP.2022.3221005).
- [6] Xiaochun Yin, Zengguang Liu, Deyong Liu, and Xiaojun Ren. A novel cnn-based bi-lstm parallel model with attention mechanism for human activity recognition with noisy data. *Scientific Reports*, 12, 05 2022. [doi:10.1038/s41598-022-11880-8](https://doi.org/10.1038/s41598-022-11880-8).