



Diplomado Inteligencia Artificial Aplicada DGTIC – UNAM

Detección y transcripción a partitura de cadencia y tono en tarareos

Proyecto Final

Nombre:

Elí Jafet Velásquez Luna



Introducción

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).

$$T = \{C, C\#, D, D\#, E, F, F\#, G, G\#, A, A\#, B\}$$

- Nota

Es una frecuencia particular del espectro disponible en el estándar MIDI. Se representa como un Tono y su altura (octava)

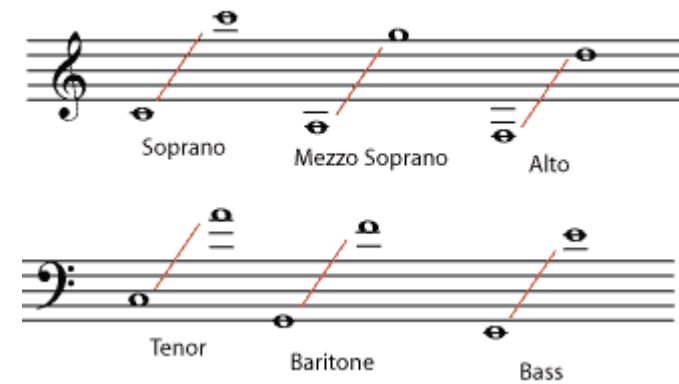
Ejm: C3, A#5, G6

- 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.



Contexto

El ámbito de la **composición musical**, tanto en música académica como amateur, es un proceso creativo de múltiples pasos. Uno de ellos:

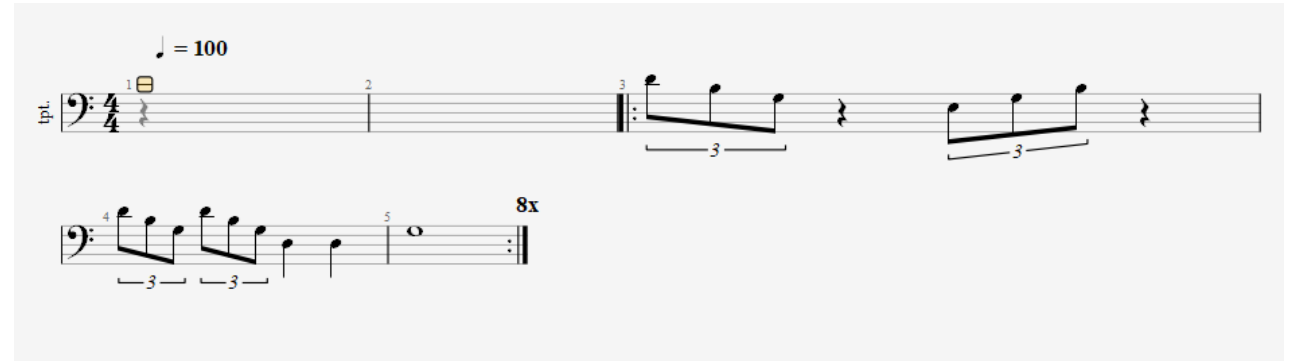
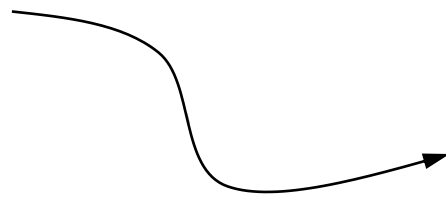
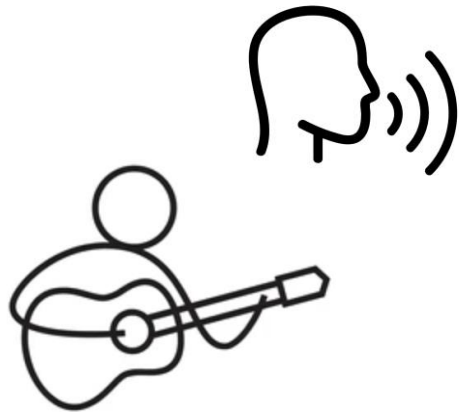
- **Transcripción de música:**

Tarea manual y larga que requiere un cierto conocimiento académico en música.

El *tararear* melodías al concebir una idea es una práctica común entre profesionales y cualquier aficionado a la música.

Estas ideas pueden **perderse** si no se someten a un proceso de escritura; el cual puede involucrar etapas:

- Repetición de la melodía múltiples veces
- Comparación con afinadores
- Corrección de figuras musicales



Contexto

Transcripción Automática de Música (AMT)

Es un área de estudio que ataca este problema con distintas soluciones:

- Autocorrelación
- Análisis espectral
- Modelos ocultos de Markov

Con el auge de las redes neuronales, han aparecido modelos más sofisticados.

Transcription of the Singing Melody in Polyphonic Music

Matti Ryynänen and Anssi Klapuri

Institute of Signal Processing, Tampere University Of Technology

P.O.Box 553, FI-33101 Tampere, Finland

{matti.ryynanen, anssi.klapuri}@tut.fi

Abstract

This paper proposes a method for the automatic transcription of singing melodies in polyphonic music. The method is based on multiple-F0 estimation followed by acoustic and musicological modeling. The acoustic model consists of separate models for singing notes and for no-melody segments. The musicological model uses key estimation and note bigrams to determine the transition probabilities between notes. Viterbi decoding produces a sequence of notes and rests as a transcription of the singing melody. The performance of the method is evaluated using the RWC popular music database for which the recall rate was 63% and precision rate 46%. A significant improvement was achieved compared to a baseline method from MIREX05 evaluations.

Keywords: singing transcription, acoustic modeling, musicological modeling, key estimation, HMM

1. Introduction

Singing melody transcription refers to the automatic extraction of a parametric representation (e.g., a MIDI file) of the singing performance within a polyphonic music excerpt. A melody is an organized sequence of consecutive notes and rests, where a note has a single pitch (a note name), a beginning (onset) time, and an ending (offset) time. Automatic transcription of singing melodies provides an important tool

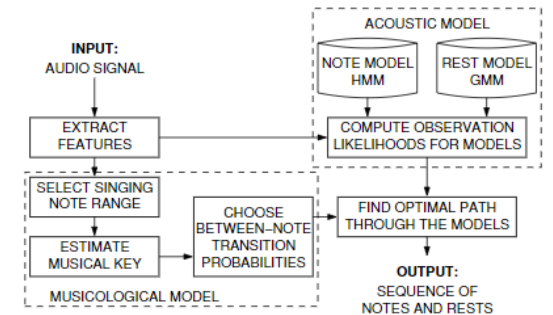


Figure 1. The block diagram of the transcription method.

the proposed method which was evaluated second best in the Music Information Retrieval Evaluation eXchange 2005 (MIREX05)¹ audio-melody extraction contest. Ten state-of-the-art melody transcription methods were evaluated in this contest where the goal was to estimate the F0 trajectory of the melody within polyphonic music. Our related work includes monophonic singing transcription [8].

Figure 1 shows a block diagram of the proposed method. First, an audio signal is frame-wise processed with two fea-

[1] Chris Donahue, John Thickstun, and Percy Liang. Melody transcription via generative pre-training, 2022

[2] François Rigaud and Mathieu Radenou. Singing voice melody transcription using deep neural networks.

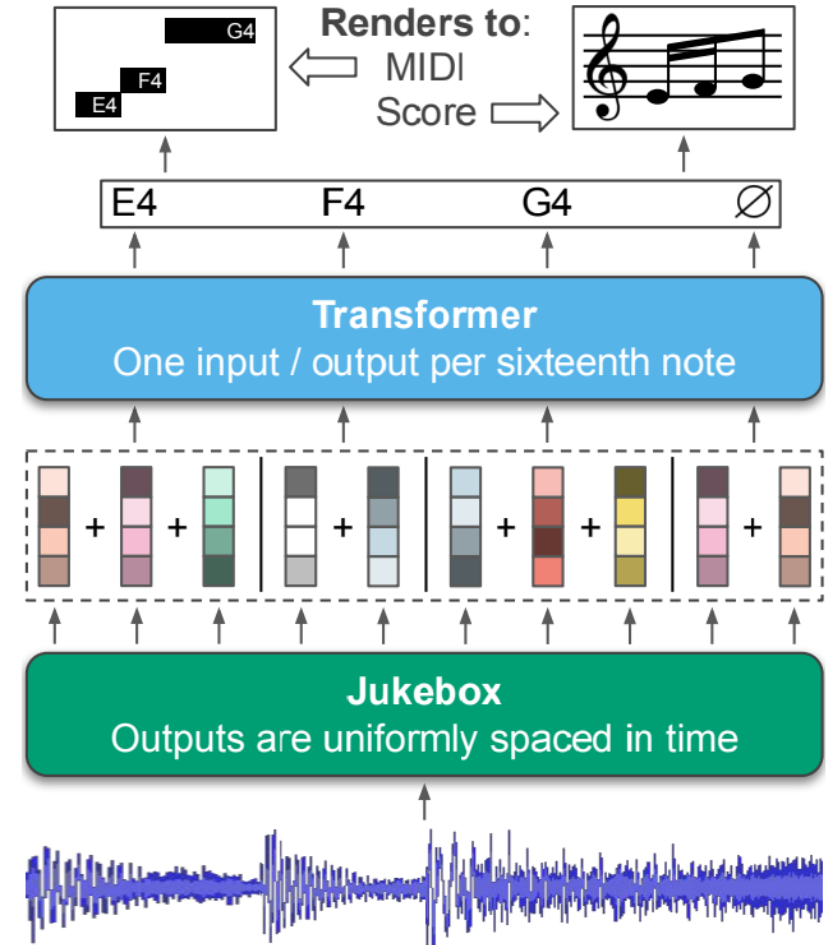
[3] Matti Ryynänen and Anssi Klapuri. Transcription of the Singing Melody in Polyphonic Music.

[4] Xianke Wang, Bowen Tian, Weiming Yang, Wei Xu, and Wenqing Cheng. Musicyolo: A vision-based framework for automatic singing transcription

Trabajos existentes

Transcripción Automática de Música (AMT)

- **Singing voice melody transcription using deep neural networks:**
Extracción de voz de música polifónica. Usa 2 capas BLSTM y se basa en encontrar melodías desde su representación en espectrogramas.
- **Melody transcription via generative pre-training:**
Transcripción de melodías obtenidas de un modelo generative (Jukebox) para instrumentos poco estudiados. Propone una Ventana de análisis mínima en 16vos de Negra para alimentar una arquitectura transformer y transcribir a partitura.
- **Transcription of the Singing Melody in Polyphonic Music**
Transcribe melodías cantadas sobre música polifónica. Propone la separación en 2 etapas: Detección de tono y detección de silencios mediante cadenas de Markov y Mezclas Gaussianas.



Problema particular

Los **tarareos** son señales de audio compuesta por

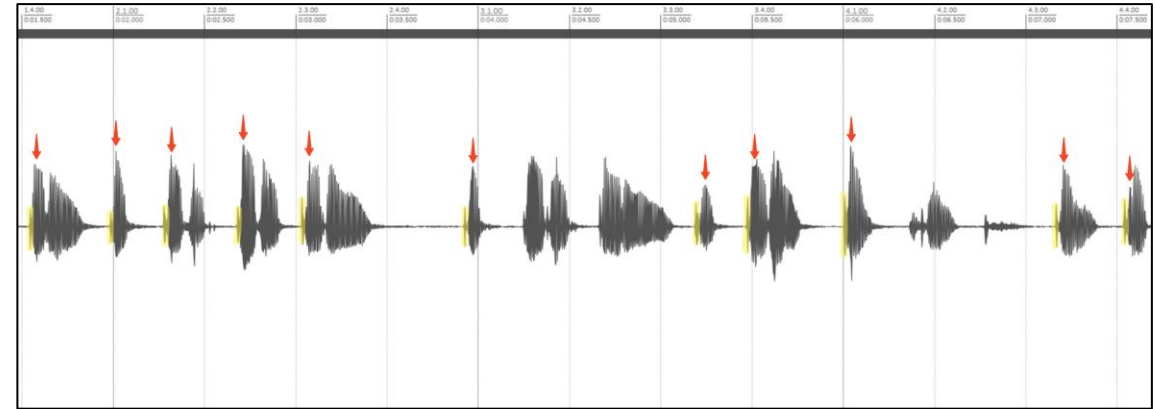
- Formantes
- Silencios

Las **partituras** son una codificación escrita del lenguaje propio de la música.

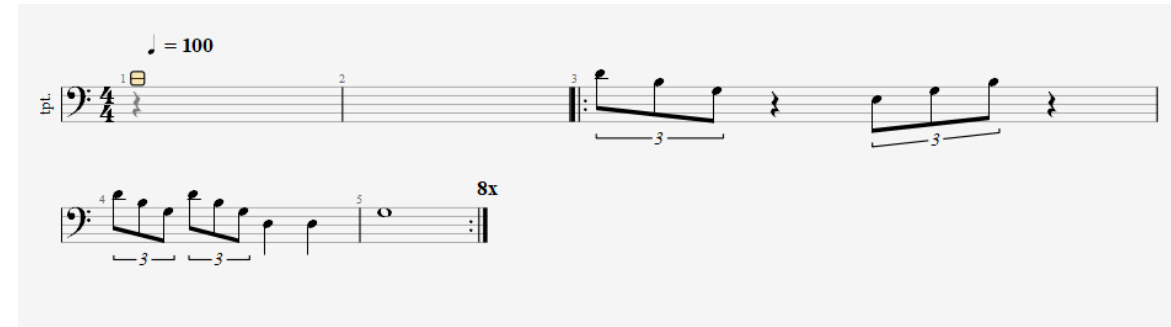
Se puede comunicar

Protocolo **MIDI**

Muestra de tarareo



Muestra de partitura



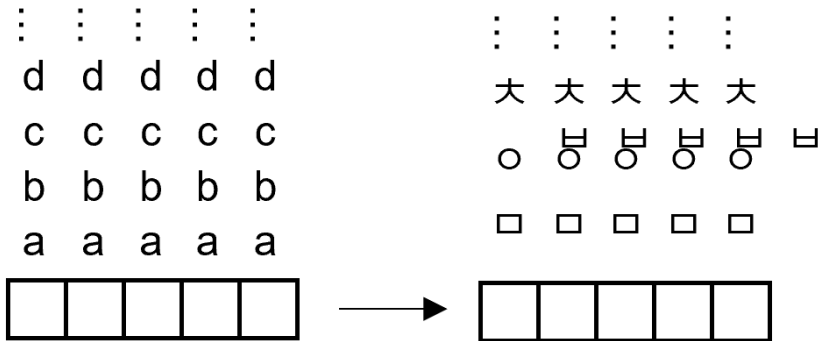
MIDI

Problema particular

El problema consiste en transcribir:

Señal continua  Mensajes discretos

Sistema traductor



La voz humana es característica de cada persona:

- El timbre es particular
- Todas las personas emiten un tono a la vez
- La intensidad y ataque son distintas

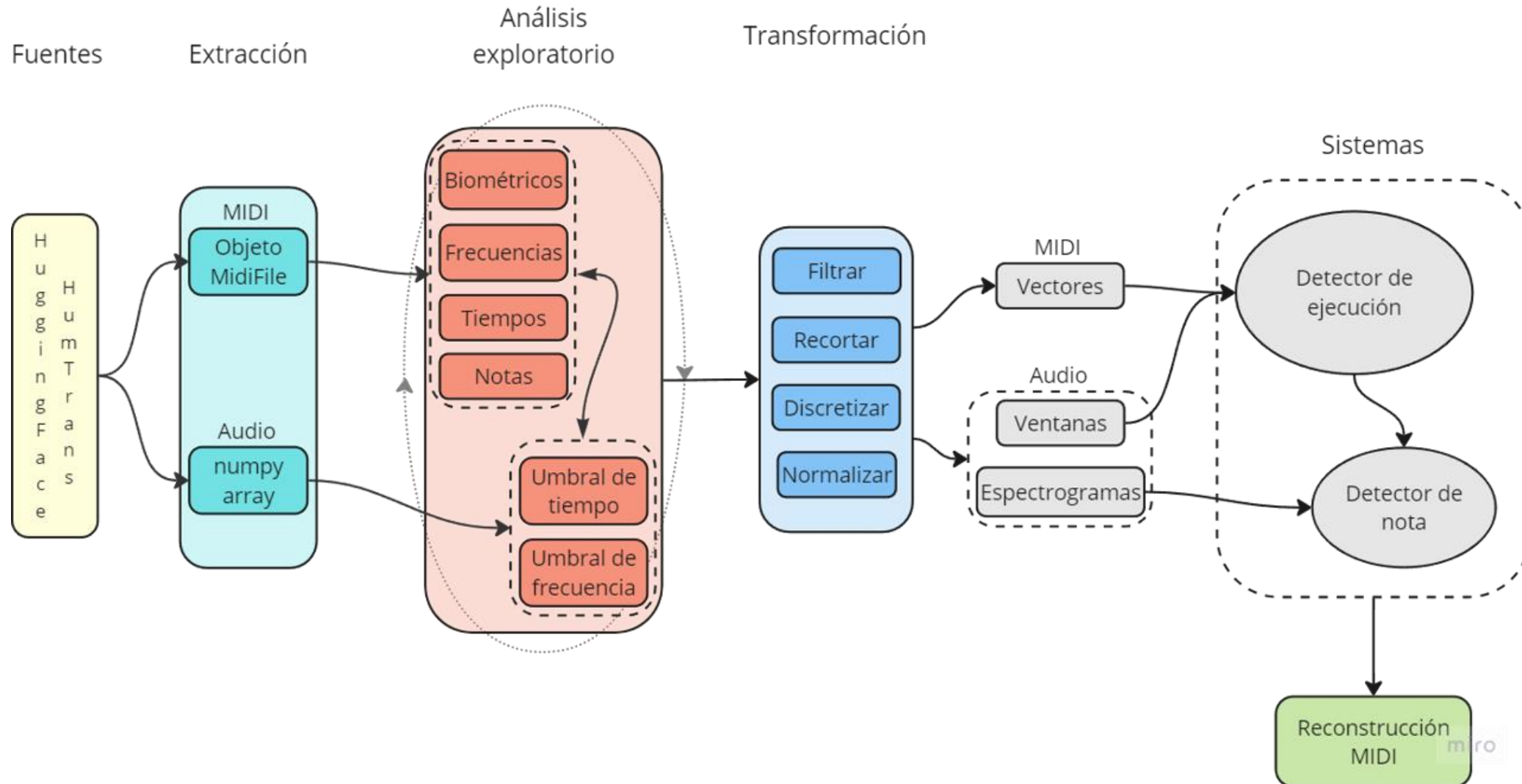
Todos entendemos cuando alguien habla o tararea.

Un sistema basado en Inteligencia Artificial podría ayudar a reconocer los patrones de un *tarareo* tratando el problema que supone el ruido natural en la voz humana.



Planeación y diseño

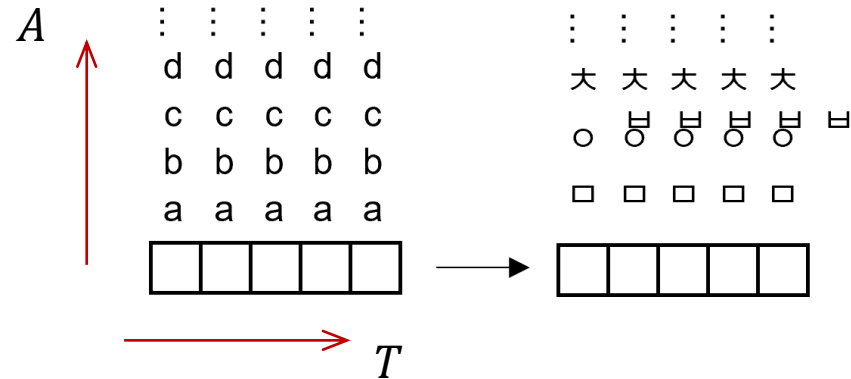
Planeación y diseño



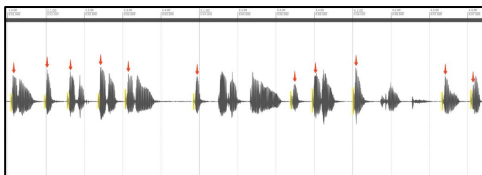
Etapas de sistemas de IA

Etapa 1 Sistema detector de ejecución

Analizar la dimensión temporal por separado. Buscando patrones de **ataque de notas**.



-> Arquitectura **Secuencia a Secuencia**

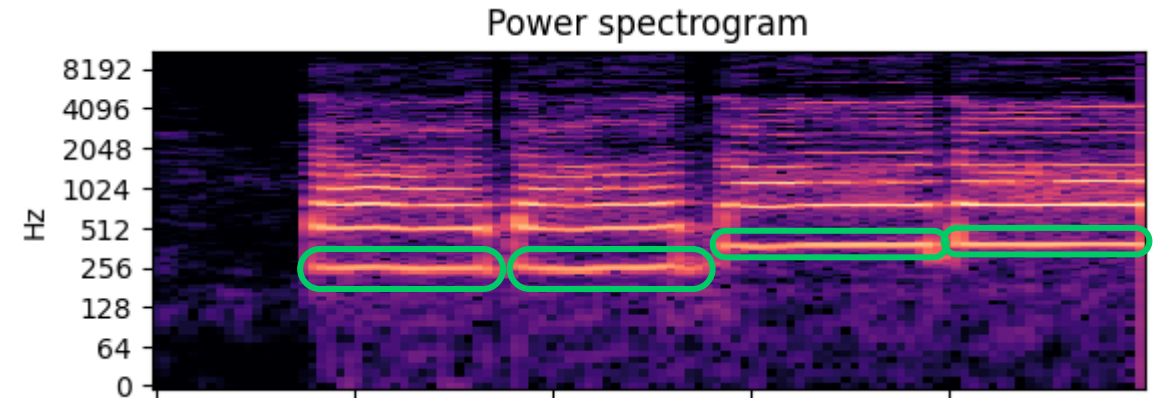


→ [0, 1, 2, ...]

Etapa 2 Sistema detector de notas

Analizar la dimensión de frecuencias para encontrar **Notas**.

- Explora la cualidad del único tono de la voz
- Grupos separados por octavas



Etapas de sistemas de IA

Etapa 1

Sistema detector de ejecución

Requisitos:

- **Input:** array (muestras, tiempo)
- **Output:** vector (tiempo,)

Etapa 2

Sistema detector de notas

Requisitos:

- **Input:**
 - espectrograma (frecuencia, tiempo)
 - Salida sistema 1
- **Output:** diccionario (índices, nota)



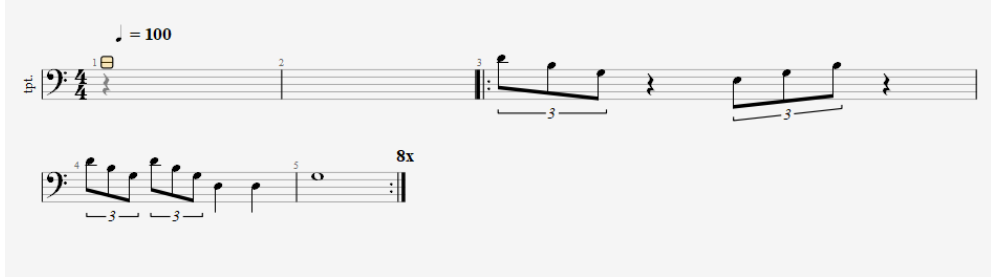
Adquisición y exploración de datos

Origen de los datos

Consulta

- Fuente:
HuggingFace: dadinhh2/HumTrans (Septiembre, 2023)
<https://doi.org/10.48550/arXiv.2309.09623>
- Información:
 - Muestras de tarareos: wav
 - Etiquetas: midi
 - Duración: > 57 horas
 - Peso: > 15gb
 - 10 voces de músicos
 - 14,610 patrones

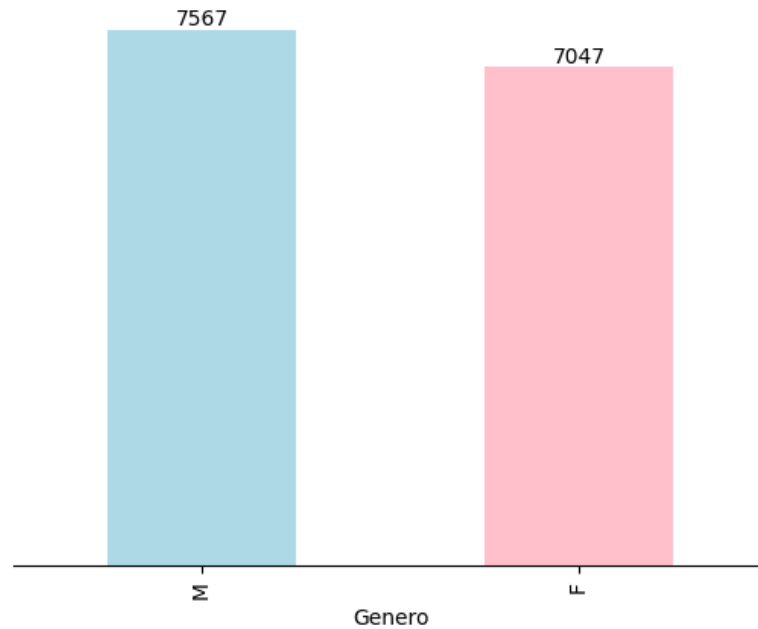
Construcción

- Fuente:
Miembros de la Tuna de la Benemérita Universidad de Oaxaca
- Contexto:
Los tarareos pueden provenir de distintas voces, con distintos timbres y representando distintas formas melódicas. Por esta razón, se requiere generar un conjunto de datos **diverso en figuras musicales**. Esto es, contemplar las notas de los distintos registros de las tesituras existentes y métricas.
- Metodología:
 1. Planear distintos **patrones** cortos (máximo 4 compases) que abarcaran un intervalo de notas sencillo (5 notas, menor a 1 octava)
 2. Generar en MIDI 63 patrones para **barítono** (tesitura predominante de la fuente).
 3. Grabar los patrones con cada uno de los participantes de la Tuna.

Análisis de datos

del conjunto de datos HumTrans

Proporción de muestras por sexo



El conjunto cuenta con aproximadamente el mismo número de muestras de audios grabados por hombres como por mujeres.

Sin embargo, provienen de sólo 5 personas distintas en cada caso:

PersonID	Conteo
F01	1959
F02	1637
F03	1218
F04	1435
F05	798
M01	1773
M02	1935
M03	1149
M04	1969
M05	741

Análisis de datos

del conjunto de datos HumTrans

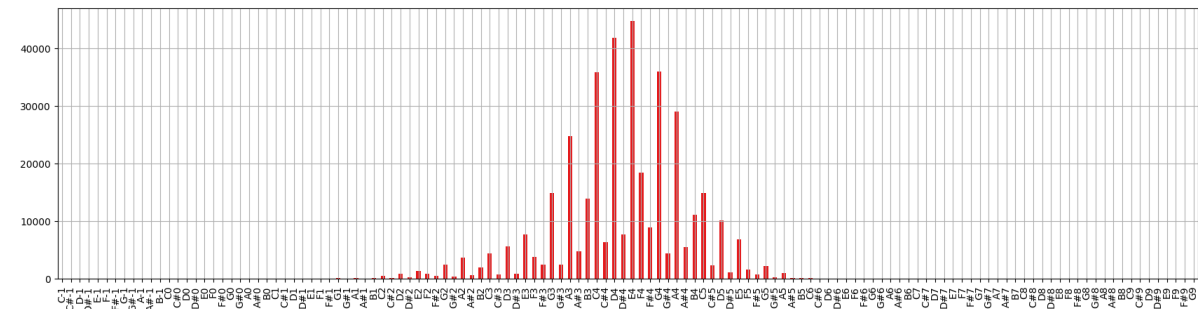
Distribución de notas

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

La distribución de frecuencia de las notas en el conjunto de datos sugiere una distribución normal

Una **distribución normal** es consistente con la distribución esperada de las voces humanas, pues es muy poco probable encontrar a una persona con voz más baja que un **E2**, o una persona con voz más alta que el límite soprano **C6**.

Esto muestra que existen más muestras en el **registro 4**, que corresponde a una tesitura “alto”.



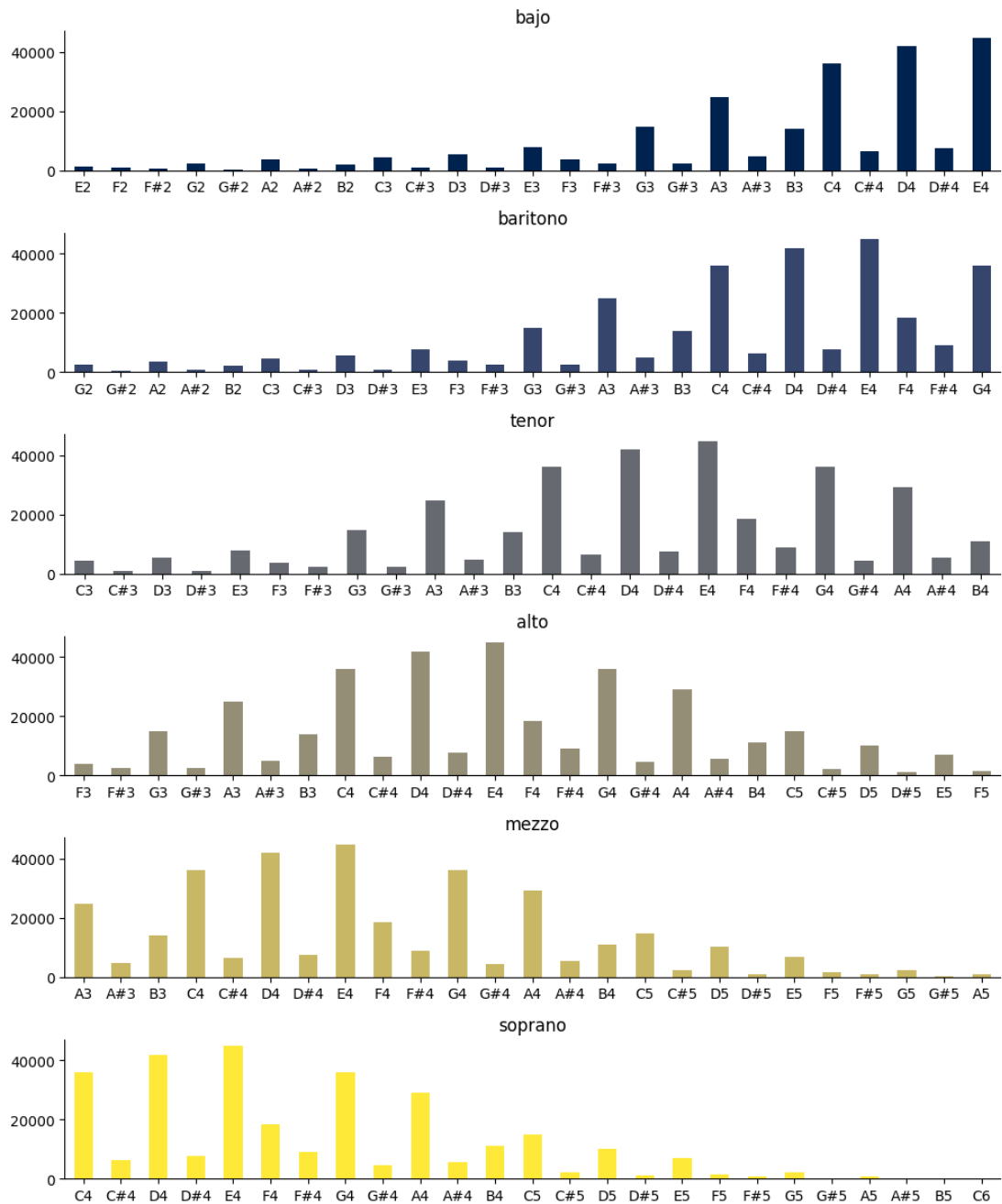
Análisis de datos

del conjunto de datos HumTrans

Esta visualización muestra nuevamente que la tesitura “alto” muestra una distribución menos sesgada.

Lo anterior es debido a que cada tesitura captura una sección de la distribución normal de todos los tonos que pueden vocalizarse.

Tesitura	Conteo de notas
Bajo	234726
Barítono	295433
Tenor	336392
Alto	353632
Mezzo	334254
Soprano	291217



Análisis de MIDI

MIDI – Estandarizar expresiones

No problemático

```
Message('note_on', channel=0, note=52, velocity=81, time=2560),  
Message('note_off', channel=0, note=52, velocity=0, time=512),  
Message('note_on', channel=0, note=57, velocity=101, time=0),  
Message('note_off', channel=0, note=57, velocity=0, time=1536),
```

Problemático

```
Message('note_on', channel=0, note=67, velocity=80, time=0),  
Message('note_on', channel=0, note=67, velocity=0, time=341),  
Message('note_on', channel=0, note=69, velocity=80, time=19),  
Message('note_on', channel=0, note=69, velocity=0, time=113),  
Message('note_on', channel=0, note=72, velocity=80, time=7),  
Message('note_on', channel=0, note=72, velocity=0, time=227),  
Message('note_on', channel=0, note=72, velocity=80, time=13)...
```



```
Message('note_on', channel=0, note=67, velocity=80, time=0),  
Message('note_off', channel=0, note=67, velocity=0, time=360),  
Message('note_on', channel=0, note=69, velocity=80, time=0),  
Message('note_off', channel=0, note=69, velocity=0, time=120),  
Message('note_on', channel=0, note=72, velocity=80, time=0),  
Message('note_off', channel=0, note=72, velocity=0, time=240),  
Message('note_on', channel=0, note=72, velocity=80, time=0)...
```

	count	ticks
Estándar	10053	1024
No estándar	4561	480

Solución

- Detectar MIDIs que no tengan “note_off”.
- Añadir los ticks residuales al mensaje “note_on” anterior.
- Cambiar el tipo a “note_off”.

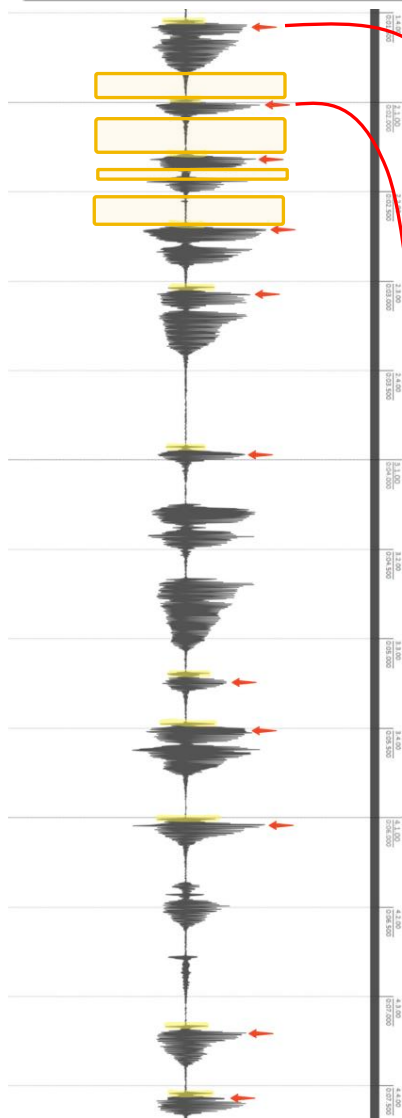
NOTA

¿Cómo saber que los ticks son residuales?

- Los audios no contemplan figuras más rápidas que una semicorchea. Por lo que se fijó un umbral de fusa (1/32).

Análisis de MIDI

Tarareo



Midi
Track única

MidiTrack([

```
MetaMessage('track_name', name='é\x92ç\x90', Piano', time=0),  
MetaMessage('time_signature',  
    numerator=2, denominator=4,  
    clocks_per_click=24,  
    notated_32nd_notes_per_beat=8,  
    time=0),  
MetaMessage('key_signature', key='C', time=0),  
MetaMessage('set tempo', tempo=555556, time=0),
```

Metadatos

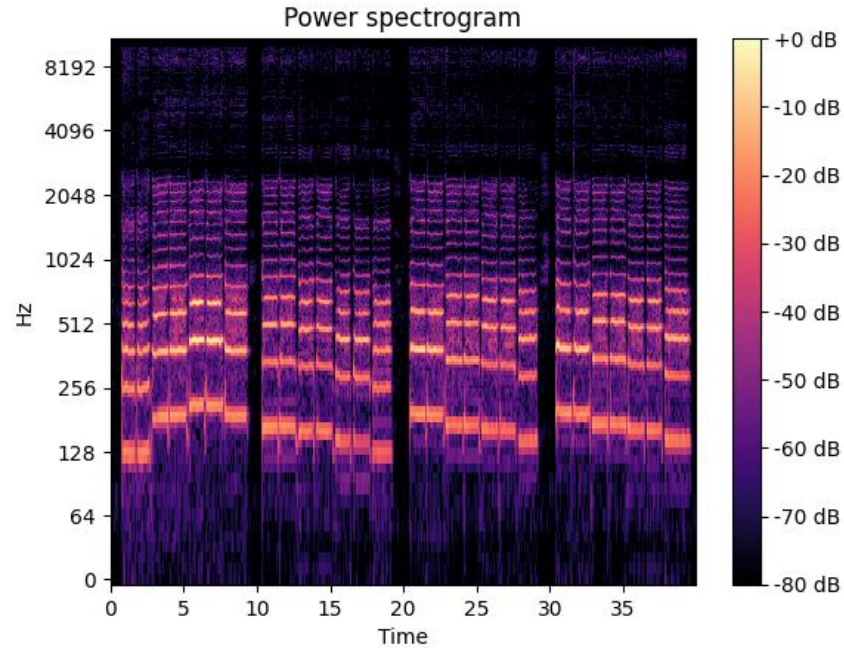
```
Message('note_on', channel=0, note=67, velocity=80, time=0),  
Message('note_on', channel=0, note=67, velocity=0, time=341),  
Message('note_on', channel=0, note=69, velocity=80, time=19),  
Message('note_on', channel=0, note=69, velocity=0, time=113),  
Message('note_on', channel=0, note=72, velocity=80, time=7),  
Message('note_on', channel=0, note=72, velocity=0, time=227),  
Message('note_on', channel=0, note=72, velocity=80, time=13)...
```

Los “Message” con velocidad>0 y los eventos anteriores definen la duración y espacio entre los tarareos.

Análisis de audio

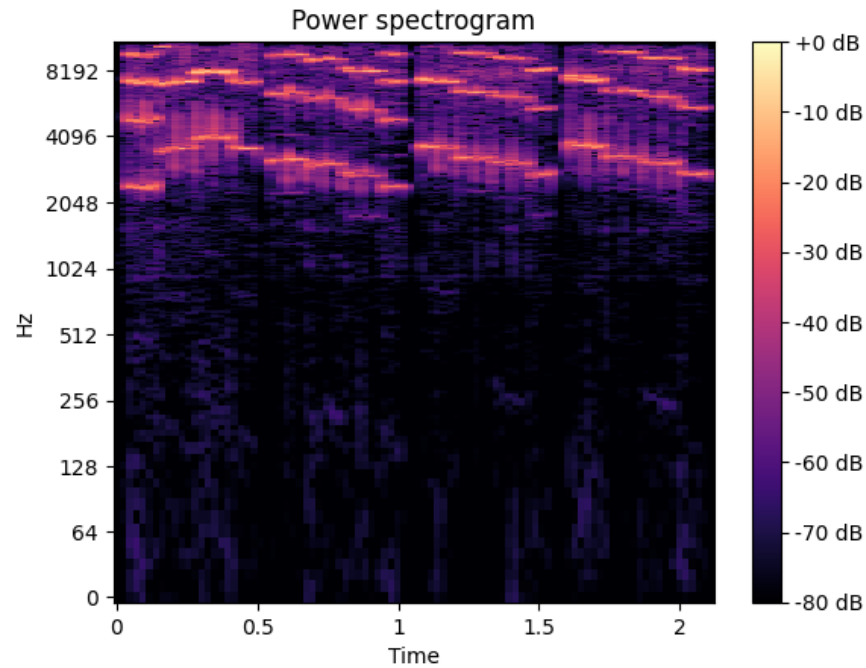
Reducción de dimensionalidad

Frecuencia



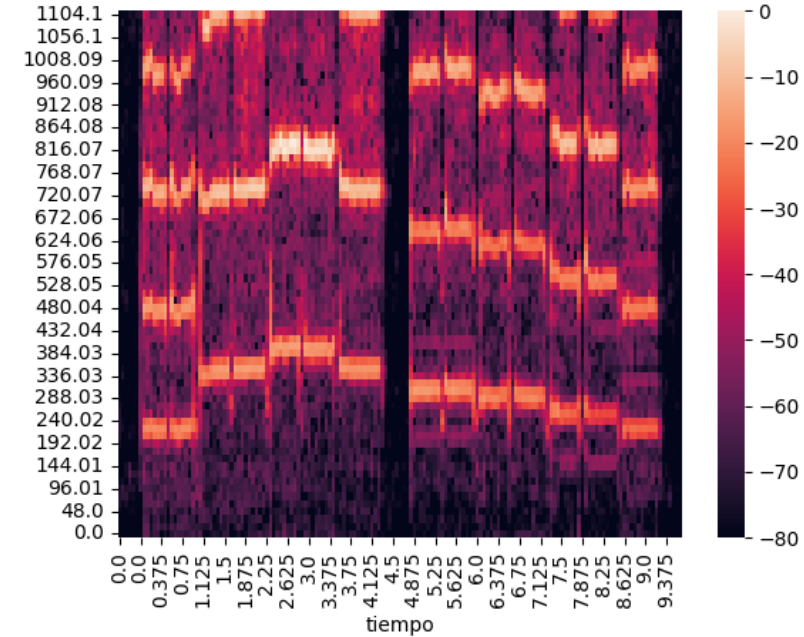
(1025, 1723)

Señal de origen



(1025, 92)

Resampleado
2 veces máxima frecuencia



(53, 1723)

Original recortado
A rango de frecuencias

Análisis de audio

Reducción de dimensionalidad

Tiempo

Se probó una ventana de longitud $L = 0.125 \text{ s}/2$ y distintas duraciones estándar

(muestras, tiempo)

- 5 segundos: (2756, 80)
- 10 segundos: (2756, 160)
- 15 segundos: (2756, 240)



Análisis de datos

Características extraídas

- Tiempo más corto: 0.125 segundos
- Rango de frecuencias: (18, 1167) Hz
- Longitud de ventana:
- Duración (D): 10 s
- Número de ventanas temporales (N): 160
- Dimensión de frecuencia (L): 70

$$L(n) = \text{lowest_time}/n$$



Pre-procesamiento

Pre-procesamiento de datos

Pre-procesamiento de etiquetas MIDI

```
Message('note_on', channel=0, note=67, velocity=80, time=0),
Message('note_on', channel=0, note=67, velocity=0, time=341),
Message('note_on', channel=0, note=69, velocity=80, time=19),
Message('note_on', channel=0, note=69, velocity=0, time=113),
Message('note_on', channel=0, note=72, velocity=80, time=7),
Message('note_on', channel=0, note=72, velocity=0, time=227),
Message('note_on', channel=0, note=72, velocity=80, time=13)...
```

Ataque
sostenido
Silencio

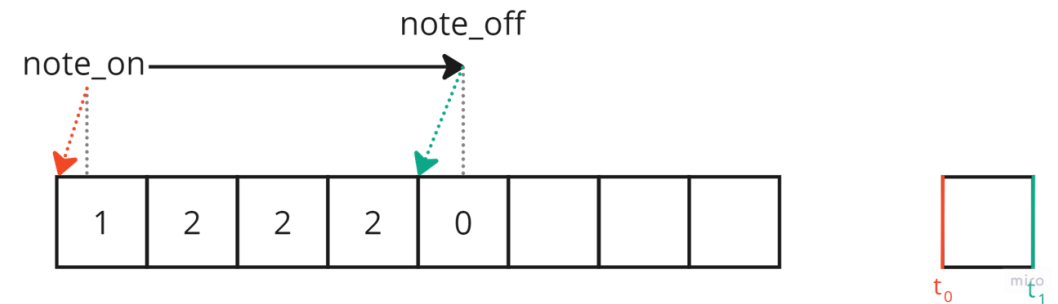
Unidad de tiempo del MIDI: ticks (dependen del tempo de la canción)

Unidades del tempo: Beats / minuto

Se crea un sistema de etiquetas para cada ventana de muestreo:

- 1: Si la ventana contiene un ataque.
- 2: Si la ventana no contiene un ataque, pero sí la continuación de la nota ejecutada.
- 0: Si la ventana contiene silencios

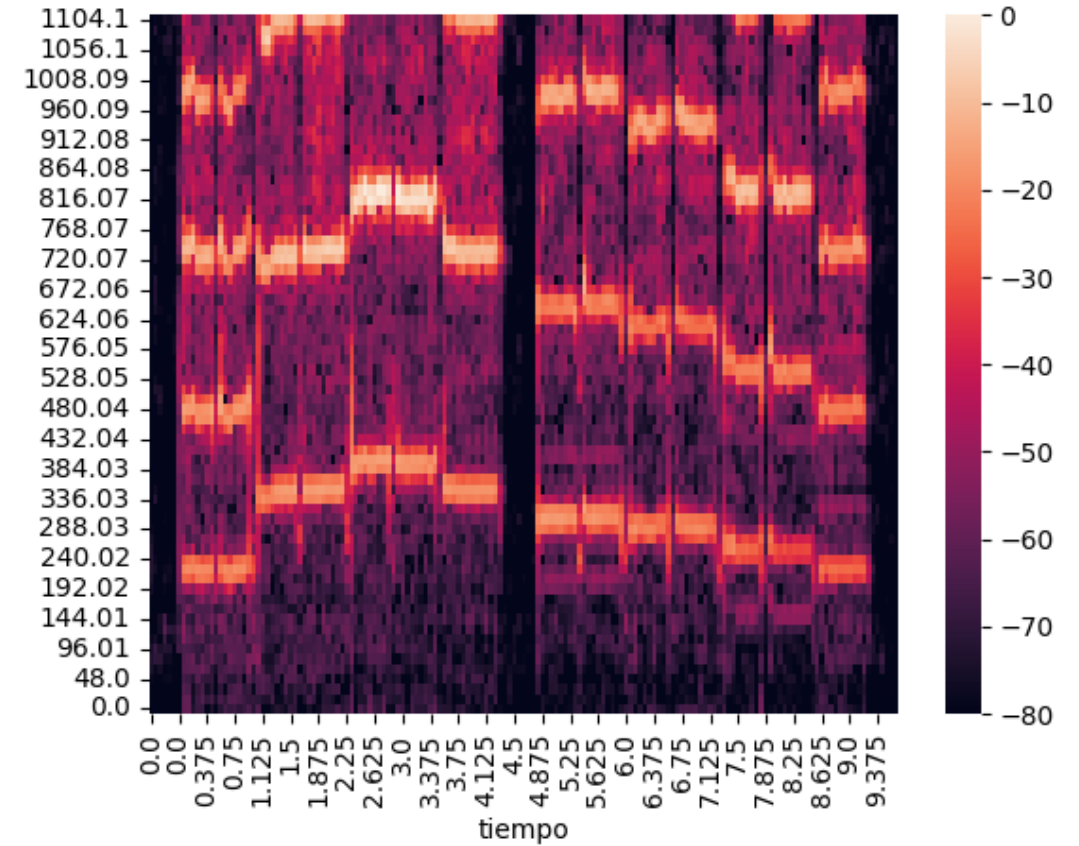
Se crea un ciclo que recorre los mensajes del MIDI.
Se inicializa una lista para el MIDI.



Pre-procesamiento de datos

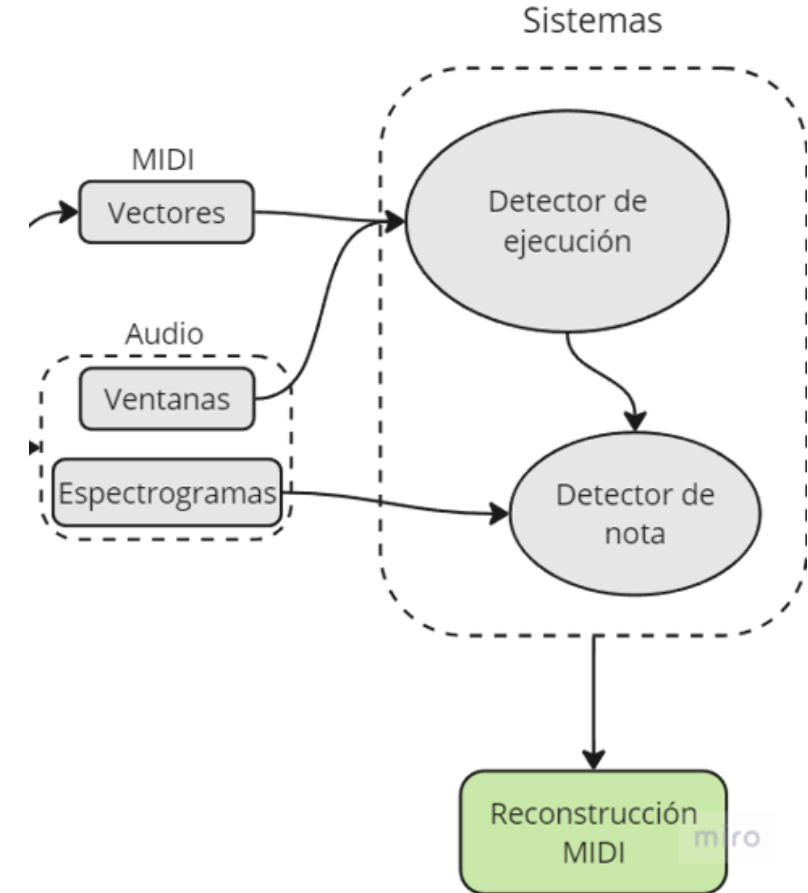
Pre-procesamiento de audio

1. Recorte de tiempo
2. Normalización
3. Generación de frames temporales
4. Generación de espectrogramas



Datos procesados

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

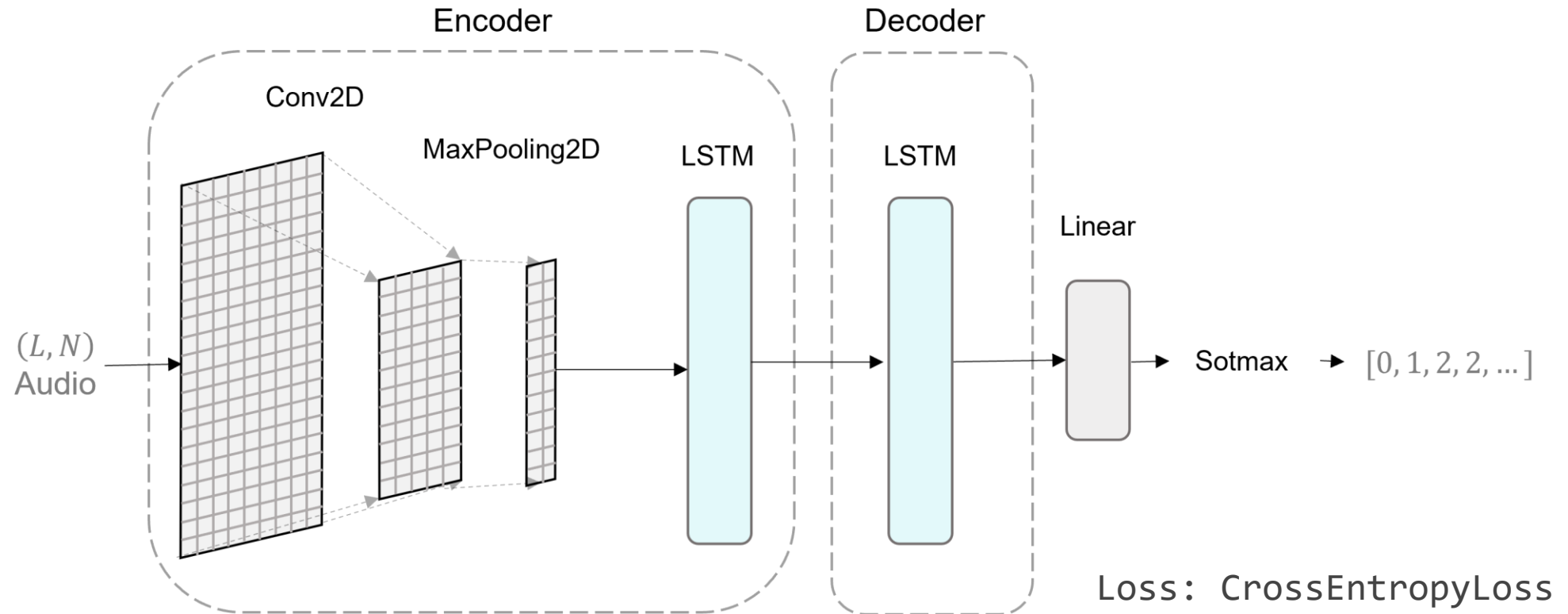




Modelado

Modelado

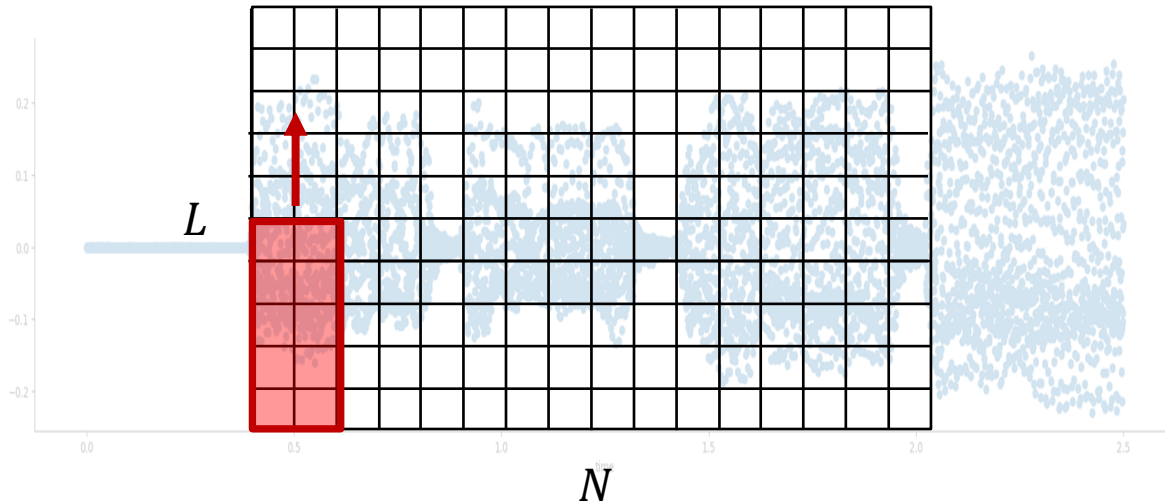
ETAPA 1: Sistema detector de ejecución



Modelado

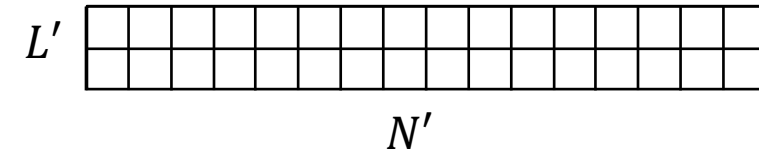
- Conv2D

- Input dim: (batch_size, 1, L, N)
- Canales: 1
- Kernel: (86,4) ← $[L//16 = 86]$
- Stride: (1,1)



- MaxPooling2D

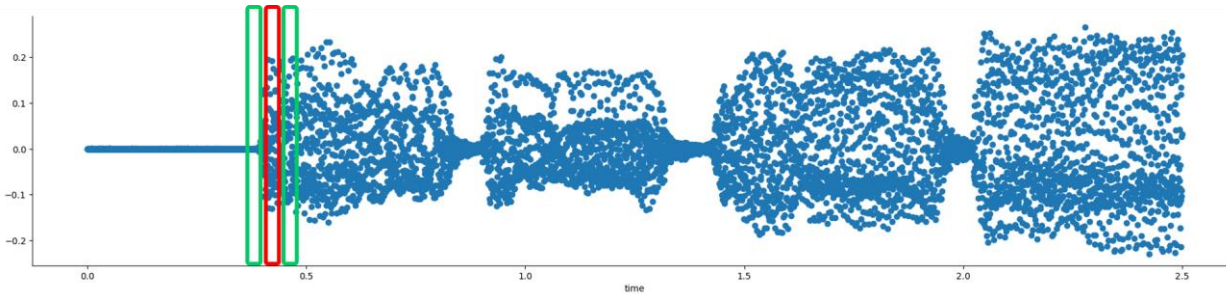
- Input dim: (batch_size, 1, L', N')
- $L' = 1293$ y N' variable.
- Kernel: (21,2) ← $[86//4 = 21]$
- Stride: (21,2)



Modelado

- Encoder LSTM

- Input dim: (batch_size, N'' , L'')
 $L''=61$ y N'' variable. Se elimina el canal de apoyo
- Canales: 1
- Hidden dim: 30



- Decoder LSTM

- Input dim: (batch_size, N'' , encoder_hidden_dim)
 $L' = 1293$ y N' variable.
- Hidden dim: encoder_hidden_dim

- Linear

- Input dim: (batch_size, N'' , encoder_hidden_dim)
- Output dim: (batch_size, N'' , 3)
Debido a 3 categorías (0,1 y 2).

Métricas de evaluación

F1-score (nivel ventana): Ya que la clase 1 sucede sólo una vez por nota ejecutada, las clases están desbalanceadas.

Levenshtein (nivel vector): Movimientos de etiquetas para recuperar la secuencia real.



Trabajo Futuro

Trabajo futuro

Proceso

Output Etapa 1 (serie de tiempo longitud N y etiquetas 0,1,2)
Espectrograma ($M \times N$)

-> Filtrar columnas de espectrograma

Extraer aquellas columnas con índices iguales a los índices de casillas con valor 1 en el Output Etapa 1.

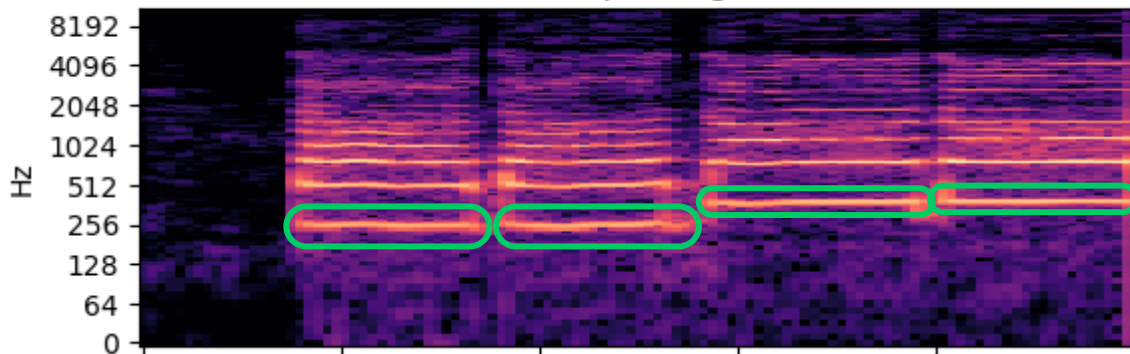
-> Aplicar **clustering**

A cada ventana temporal en el espectrograma: $M \times L$

-> Ajustar a notas de la escala cromática

-> Escoger el clúster con centro de menor frecuencia
(Nota fundamental)

Power spectrogram



Algoritmo

K-means: El algoritmo es computacionalmente rápido y al ser un proyecto de dos etapas, es preferible. Además, la mayor frecuencia de voz aproximada es de $\sim 1100\text{Hz}$ y considerando que el humano sólo ejecuta una nota a la vez, los armónicos posibles están lo suficientemente separados para que k-medias pueda funcionar.

Mezclas Gaussianas: Debido a que es más flexible con la forma y desvanecimiento de los clústeres, se preferiría este algoritmo. Sólo se considera el tiempo de cómputo.

Clústeres: 5 (son los rangos en promedio con mayor fuerza en el timbre de las personas).

Parámetros

Silhouette Score: Para medir que tan compactos son los clústeres, lo que permitiría diferenciar entre armónicos poco separados.

Métricas de clasificación multietiqueta: Ya que se escogerá la nota fundamental por cada ventana, entonces se puede comparar con la información del archivo MIDI.

Trabajo futuro

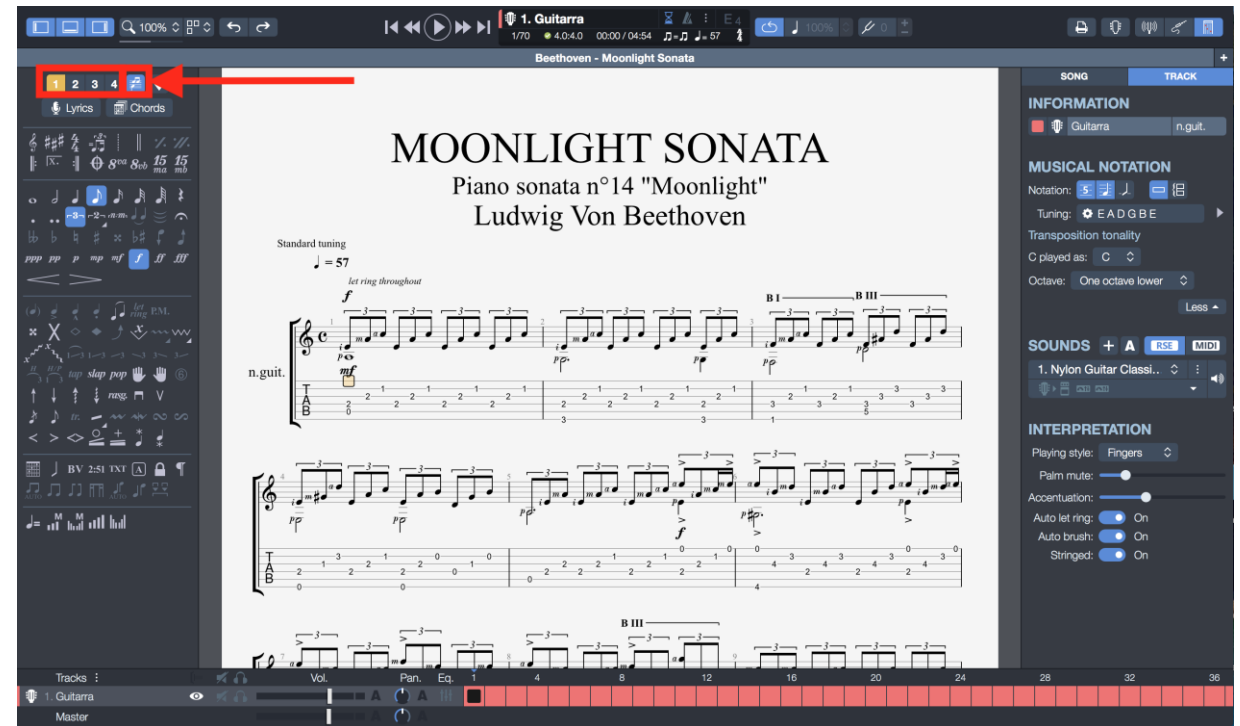
Completar Pipeline a partitura

Este trabajo se centra en transcribir a un tipo de partitura que es la comunicación MIDI.

Se podría implementar un paso al Pipeline

- Transcripción MIDI
- Rutina en PyGuitarPRO
- Generación de partitura a partir de MIDI

[0,18,128,128, ...]



Correcciones

- Definición del objeto Dataset para importar los datos

```
class TarareoMIDIIDataset(torch.utils.data.Dataset):
    def __init__(self, tarareo_directory: str, midi_directory: str, samples: list, extensions: tuple[str, str]):
        self._tarareo_directory = tarareo_directory
        self._midi_directory = midi_directory

        self._samples = samples # Nombre del elemento de grabación
        self._extensions = extensions # extensión del nombre del tarareo y vector midi. Ejm audio_name[extension] (incluye formato)

        # Si usamos PADDING, será conveniente tener un vocabulario como si se tratara de una traducción en NLP
        # Esto ayuda a ordenar las etiquetas por orden de frecuencia
        # Colocando en las primeras casillas a las etiquetas de inicio y final
        # self._vocabulary = vocabulary

    def __len__(self):
        return len(self._samples)

    def __getitem__(self, idx):
        # Recuperar nombre de la muestra
        file_name = self._samples[idx]
        tarareo_name = file_name + self._extensions[0]
        midi_name = file_name + self._extensions[1]

        # Importar
        tarareo_path = os.path.join(self._tarareo_directory, tarareo_name)
        tarareo = np.load(tarareo_path)

        midi_path = os.path.join(self._midi_directory, midi_name)
        midi = np.load(midi_path)

        # Agregar <SOS> y <EOS> ?
        return torch.tensor(tarareo, dtype=torch.float32), torch.tensor(midi, dtype=torch.int)
```

Preparación de Dataset

Correcciones

- Definición de `collate_fn` para crear batches y homologar las dimensiones de audios de distinta duración.

```
def audio_vector_collate_fn(batch: list[tuple[torch.Tensor, torch.Tensor]]) -> tuple[torch.Tensor, torch.Tensor]:
    """Función para agregar padding a los audios y vectores de etiquetas por lote.

    :param batch: Una lista de tuplas donde cada tupla contiene un audio y su vector de etiquetas.
    :return: Una tupla que contiene dos tensores:
        1) Un tensor que contiene todos los audios del lote, apilados juntos. Forma: [batch_size, L_max, N_max].
        2) Un tensor que contiene todos los vectores de etiquetas del lote, rellenos con zeros para que tengan la misma longitud.
           Forma: [batch_size, K_max].
    """
    audios, labels = zip(*batch)

    # Determinar las dimensiones máximas
    L_max = max(audio.size(0) for audio in audios)
    N_max = max(audio.size(1) for audio in audios)
    K_max = max(label.size(0) for label in labels)

    # Inicializar los tensores con padding
    padded_audios = torch.zeros(len(audios), L_max, N_max, dtype=torch.float32)
    padded_labels = torch.zeros(len(labels), K_max, dtype=torch.int64)

    for i, (audio, label) in enumerate(batch):
        L = audio.size(0)
        N = audio.size(1)
        K = label.size(0)

        # Copiar el audio y el vector de etiquetas a los tensores con padding
        padded_audios[i, :L, :N] = audio
        padded_labels[i, :K] = label

    return padded_audios, padded_labels
```

`collate_fn`

PAD a secuencias de distinta longitud

Correcciones

Encoder

```
class Encoder(torch.nn.Module):
    def __init__(self, input_dim, hidden_dim):
        super(Encoder, self).__init__()

        # Conv2D Layer
        self.conv2d = torch.nn.Conv2d(in_channels=1,
                                       out_channels=1,
                                       kernel_size=(int(1378//16), 4),
                                       stride=(1, 1))

        # Calculate output dimension after Conv2D
        self.conv2_H_dim = convout_calc(input_dim, kernel=int(1378//16), stride=1)

        # MaxPooling Layer
        self.max_pool2d = torch.nn.MaxPool2d(kernel_size=(int(86//4), 2))

        # Calculate output dimension after MaxPooling
        self.pool_H_dim = convout_calc(self.conv2_H_dim, kernel=int(86//4), stride=int(86//4))

        # Encoder LSTM
        self.encoder_lstm = torch.nn.LSTM(input_size=self.pool_H_dim,
                                          hidden_size=hidden_dim,
                                          batch_first=True)

    def forward(self, x):
        # Pass through Conv2D
        x = x.unsqueeze(1)
        conv_out = self.conv2d(x)
        # Pass through MaxPooling
        pool_out = self.max_pool2d(conv_out)
        # Remove channel dimension added by Conv2D and adjust for LSTM input
        pool_out = pool_out.squeeze(1)
        pool_out_t = pool_out.transpose(1, 2)

        # Pass through LSTM
        encoder_outputs, (hidden, cell) = self.encoder_lstm(pool_out_t)

        return encoder_outputs, (hidden, cell)
```

Decoder

```
class Decoder(torch.nn.Module):
    def __init__(self, output_dim, hidden_dim):
        super(Decoder, self).__init__()

        # Decoder LSTM
        self.decoder_lstm = torch.nn.LSTM(input_size=hidden_dim,
                                          hidden_size=hidden_dim,
                                          batch_first=True)

        # Capa lineal que mapeará las salidas del LSTM a la dimensión deseada de salida (es decir, el número de clases)
        self.fc = torch.nn.Linear(hidden_dim, output_dim)

    def forward(self, hidden, cell, target_len, teacher_forcing_ratio=0.5, target_seq=None):
        batch_size = hidden.size(1)
        print('batch: ', batch_size)
        output_dim = self.fc.out_features
        print('output_dim programada: ', output_dim)
        outputs = torch.zeros(batch_size, target_len, output_dim).to(hidden.device)
        print('outputs placeholder: ', outputs.size())
        # Entrada inicial del decodificador (vector de ceros para cada elemento del batch)
        decoder_input = torch.zeros((batch_size, 1, hidden.size(2))).to(hidden.device)
        print('decoder input placeholder: ', decoder_input.size())

        print("RECURSION")
        # Iterar sobre cada paso temporal
        for t in range(target_len):
            # Pasar a través del LSTM
            output, (hidden, cell) = self.decoder_lstm(decoder_input, (hidden, cell))
            print('deLSTM output: ', output.size())
            print('deLSTM hidde: ', hidden.size())
            print('deLSTM cell: ', cell.size())
            # Pasar la salida del LSTM por la capa totalmente conectada
            output = self.fc(output)
            print('linear output: ', output.size())
            print('linear.squeeze output: ', output.squeeze(1).size())
            # Almacenar las predicciones de salida
            outputs[:, t, :] = output.squeeze(1)

            # Determinar si estamos utilizando teacher forcing
            teacher_force = (torch.rand(1).item() < teacher_forcing_ratio) if target_seq is not None else False

            # Obtener la siguiente entrada al decodificador
            print('target seq:', target_seq.size())
            print('target seq slice:', target_seq[:, t].size())
            print('target seq slice unsquee:', target_seq[:, t].unsqueeze(1).size())
            if teacher_force:
                decoder_input = target_seq[:, t]
            else:
                decoder_input = output

        return outputs
```

Conclusiones

1. Los archivos MIDI y de audio pueden requerir codificación muy especializada para adecuarlos a los propósitos del proyecto.
 - 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.
2. El sistema que propone Matti Ryynänen y Anssi Klapuri por etapas propone la detección de silencios. El preprocesamiento de datos de este proyecto podría beneficiarse de ese enfoque.
3. Con base en los trabajos previos, la arquitectura con redes recurrentes es una buena opción para seguir trabajando.