Universidad de Buenos Aires
Facultad de Ciencias Exactas y Naturales
Facultad de Ingeniería
Maestría en Explotación de Datos y Descubrimiento de Conocimiento

Sistemas de transcripción automática de batería: diseño y evaluación de enfoques basados en aprendizaje profundo moderno y aprendizaje automático tradicional.

Tesis presentada para optar al título de Magister en Explotación de Datos y Descubrimiento de Conocimiento

Tesista: Tomás Marcos
Director: Pablo Riera

Buenos Aires, 9 de septiembre de 2025

Resumen

Los sistemas de transcripción automática de batería (TAB) intentan generar una transcripción de los eventos ocurridos en un sistema de batería a partir de una pista o canción, lo cual resulta una tarea desafiante. En este trabajo exploraremos diferentes enfoques para integrar un sistema de TAB: modelos o algoritmos de reconocimiento del instrumento, de detección de eventos y de separación de fuentes. Se hará énfasis en los modelos de reconocimiento, probando así diferentes técnicas. Algunas de estas tradicionales como máquinas de soporte vectorial; otras en cambio, basadas en aprendizaje profundo como son las redes convolucionales (CNN) tanto con características basadas en espectrogramas como con características basadas en otros modelos como Music undERstanding model with large-scale self-supervised Training (MERT). Por último, se hace prueba de enfoques basados en prompts de lenguaje natural tales como Contrastive Language-Audio Pretraining (CLAP.)

En principio, se utilizan las anotaciones de verdad de campo (groundtruth) para los modelos de reconocimiento y posteriormente se incorporan métodos de detección de eventos para prescindir de estas, logrando así un sistema de TAB utilizando pistas de batería aisladas.

Como prueba final, se incorporan las pistas con todos los instrumentos y se prueba entrenar modelos con estas pistas completas. Además, se entrenan estos modelos de detección de eventos y reconocimiento del instrumento pero utilizando un modelo de separación de fuentes como parte del preprocesamiento (Hybrid-Demucs). Este último enfoque resulta tener mayor éxito que el primero, principalmente debido a una mejora en el modelo de detección de eventos.

Realizamos las evaluaciones de dichos métodos utilizando el conjunto de datos llamado “MDB Drums”. Este mismo, tiene la ventaja de poseer tanto pistas completas como aisladas por tipo de instrumento, lo cual nos permitió evaluar la utilidad de un algoritmo de separación de fuentes para la tarea de TAB. De la comparación se desprende que los métodos basados en CNN demuestran un rendimiento superior en el reconocimiento de los instrumentos que componen la batería. En el agregado, se encuentra un alto rendimiento (medido por F1-score) para los instrumentos de bombo, HiHat y redoblante; mientras que por otro lado se encuentra uno bajo para los instrumentos de pandereta/side stick, toms y platillos, siendo una de las causas de esto, el tamaño muestral del conjunto de entrenamiento. Se encuentra que los resultados no varían significativamente al incorporar un método de detección de eventos.

Por último, se proponen alternativas a futuro para incrementar la calidad del reconocimiento en los instrumentos donde esta no resultó favorable, tales como incorporar técnicas de data augmentation.

Palabras clave: Transcripción Automática de Batería, Reconocimiento de Instrumentos, Detección de Eventos, Separación de Fuentes, Recuperación de Información Musical.

Automatic drum transcription systems: design and evaluation of approaches based on modern deep learning and traditional machine learning.

Abstract

Automatic drum transcription (ADT) systems aim to generate a transcription of the events occurring in a drum set from an audio track or song, which is a challenging task. In this work, we explore different approaches to integrating an ADT system: instrument recognition models or algorithms, event detection, and source separation. Emphasis is placed on recognition models, testing different techniques. Some of these are traditional, such as support vector machines; others are based on deep learning, such as convolutional neural networks (CNNs), using both spectrogram-based features and features derived from other models such as the Music undERstanding model with large-scale self-supervised Training (MERT). Finally, we experiment with approaches based on natural language prompts, such as Contrastive Language-Audio Pretraining (CLAP).

Initially, ground truth annotations are used for the recognition models, and later event detection methods are incorporated to remove the dependency on these annotations, thus achieving an ADT system using isolated drum tracks. As a final test, full mixes with all instruments are incorporated, and models are trained on these complete tracks. Additionally, event detection and instrument recognition models are trained using a source separation model as part of the preprocessing stage (Hybrid-Demucs). This latter approach proves more successful than the former, mainly due to improvements in the event detection model.

We evaluate these methods using the MDB Drums dataset, which offers the advantage of containing both full mixes and isolated tracks per instrument, allowing us to assess the usefulness of a source separation algorithm for the ADT task. From this comparison, CNN-based methods demonstrate superior performance in recognizing drum instruments. Overall, high performance (measured by F1-score) is achieved for bass drum, hi-hat, and snare, while lower performance is observed for tambourine/side stick, toms, and cymbals—partly due to the limited sample size in the training set. We also find that results do not vary significantly when incorporating an event detection method.

Finally, future directions are proposed to improve recognition quality for instruments with lower performance, such as integrating data augmentation techniques.

Keywords: Automatic Drum Transcription, Instrument Recognition, Event Detection, Source Separation, Music Information Retrieval.

Índice

Índice de figuras y tablas

1. Introducción

La batería es un instrumento musical conformado por un conjunto de instrumentos de percusión, configurados para ser tocados por una sola persona[1]. Un kit de batería típico está compuesto por varios de estos instrumentos, como bombos, toms, Hi-Hats y platillos, cada uno con características tímbricas distintivas.

A su vez, estos instrumentos pueden clasificarse en membranófonos e idiófonos [1]. Los membranófonos utilizan una membrana tensa (parche) para producir sonido, y dicho sonido depende de la tensión del parche. Algunos de estos son el bombo, el redoblante y los toms. Por otro lado, los idiófonos son instrumentos que utilizan la vibración de su propio cuerpo para generar sonido. Algunos de estos son el Hi-Hat (abierto), el Crash y el Ride (ambos platillos, CY para nuestra nomenclatura). Estos últimos tienen un sonido de mayor duración, diferenciándose así del primer grupo.

Esta riqueza de sonidos generada por los múltiples instrumentos que contiene una batería, junto con el hecho de que pueden existir variaciones de ejecución entre distintos bateristas y superposición entre los diferentes instrumentos, introduce desafíos específicos en su transcripción automática. La mayoría de ellos serán abordados a lo largo de este trabajo.

La transcripción automática de batería (TAB) es un área activa y de gran utilidad para la transcripción automática de música (TAM). Si bien se han desarrollado diferentes métodos con buen rendimiento, producir transcripciones robustas aún resulta un gran desafío. Para una transcripción exitosa, cada evento en una pieza de música debe ser caracterizado musicalmente con tono, momento en el que ocurre, duración y el tipo de instrumento que ocurre en el evento [2]. Concordantemente, el problema de transcripción automática de música puede dividirse en tres partes: detección de eventos/onsets (DO), el cual es comúnmente conocido como onset detection, estimación de la frecuencia fundamental (f0-estimation) y reconocimiento del instrumento (RI), también comúnmente conocido como instrument recognition. Tratándose en nuestro caso, de sonidos en un kit de batería (TAB), sólo deberíamos concentrarnos en la primera y la tercera parte del problema. Dicho de otro modo, buscaremos detectar cuándo ocurre un sonido de batería (detección de eventos) y si este ocurre, qué tipo de instrumento de percusión produjo dicho sonido (reconocimiento del instrumento).

Las tareas mencionadas anteriormente, pueden realizarse con pistas de batería aisladas o en presencia de múltiples instrumentos. En este último escenario, es donde los algoritmos de separación de fuentes (SS) resultan de gran utilidad.

1.1. Antecedentes

Para resolver el problema de reconocimiento del instrumento, existen aproximaciones basadas en algoritmos tradicionales de aprendizaje automático como Máquinas de Soporte Vectorial (SVM) [3], así como otros basados en redes neuronales [4, 5, 6].

Thomson et. al [3] , utilizan un enfoque basado en SVM, creando un conjunto de datos sintético para generar patrones rítmicos y luego generar un modelo que clasifique no instrumentos de batería sino patrones rítmicos de batería para luego transcribirlos.

Southhal et. al [6], emplean múltiples redes neuronales recurrentes (RNN) tanto bidireccionales (offline) como sólo recurrentes (online)[2], para detectar los onsets así como para reconocerlos; con una representación de las variables de entrada mediante la transformada de Fourier en el tiempo o Short Fourier Time Transform (STFT), combinada con Hanning window.

Jacques y Roebel [5], utilizan un enfoque basado en múltiples redes neuronales convolucionales (CNN) tanto para detección de eventos como para reconocimiento del instrumento, omitiendo el detector de eventos como un paso de preprocesamiento en un primer caso; en segundo caso utilizan la CNN para detección de onset y posteriormente un método conocido como non-negative matrix deconvolution para el reconocimiento de cada instrumento.

[1] https://en.wikipedia.org/wiki/Bater%C3%ADa_(instrumento_musical)

[2] Se entiende por online cuando el algoritmo puede realizar inferencias en tiempo real. En el caso de una red recurrente unidireccional, se puede considerar online porque es posible realizar la inferencia a medida que recibimos pedazos de la señal o grabación. El caso de una red recurrente bidireccional por el contrario se considera offline, puesto que necesitamos toda la señal o grabación para poder realizar la inferencia, puesto que la red que está en la dirección de derecha a izquierda, comienza a producir los estados ocultos o "hidden states"(h) desde el último fragmento de la señal al primero. Existen excepciones donde redes bidireccionales infieren en tiempo real, tales como casos donde la ventana es lo suficientemente pequeña sumado un buen poder de computo.

Ishizuka et. al [4], mencionan la dificultad que tiene aprender patrones rítmicos propios de la batería solamente utilizando representaciones tiempo-frecuencia (frame level). Para responder a esta problemática, los autores proponen combinar una red neuronal recurrente convolucional (CRNN) que infiera directamente la secuencia de los onsets a nivel tatum[3] a partir de un espectrograma (mel), con un modelo de lenguaje que represente una distribución probabilística de los onsets de batería.

Por otro lado, también existen metodologías de carácter no supervisado, como la planteada por Choi y Cho [7]. Los autores presentan DrummerNet, un sistema de TAB basado en dos módulos de aprendizaje profundo (analysis y synthesis module) donde se aprenden a reconstruir las señales originales, intentando minimizar la distancia entre la señal de entrada y la de salida (o equivalentemente maximizar la onset spectrum similarity); y esto sirve como un indicador indirecto (proxy) de la diferencia entre la etiqueta (label) estimada y la etiqueta de groundtruth.

1.2. Objetivos

Este trabajo, tiene por objetivo principal evaluar y comparar distintos métodos de transcripción automática de batería dado el conjunto de datos "MDB Drums" que presentaremos con mayor detalle en la sección 2. Más precisamente se busca:

- Evaluar si es posible visualizar patrones por los cuales el ser humano puede distinguir entre instrumentos de batería mediante un espectrograma o bien la señal en el dominio del tiempo. Hacer esto tanto para la señal en presencia de batería como para la misma bajo la presencia del resto de los instrumentos (voz, bajo, etc).
- Comparar distintos métodos tradicionales, de aprendizaje automático y aprendizaje profundo para el reconocimiento del instrumento y la detección de eventos, e incluso enfoques modernos de transfer learning o contrastive learning.
- Evaluar el impacto de múltiples instrumentos sobre el sistema de TAB. Verificar si un método de separación de fuentes puede contribuir a la performance de dicho sistema.
- Plantear las directrices a seguir para mejorar los modelos en el futuro. Es importante destacar que gran parte de los problemas de este campo provienen de la falta de anotaciones (tamaño muestral) tal como se señala en [1] y en nuestro caso no es la excepción, por lo que debemos plantear posibles soluciones a dicha problemática.
- Disponibilizar la mayor parte del código posible en el repositorio de este proyecto[4]. con el fin de garantizar la reproducibilidad de los resultados y en lo posible brindar herramientas adicionales para las tareas de TAB.

1.3. Enfoque del trabajo y estrategias propuestas

En este trabajo nos concentraremos en construir un sistema robusto de transcripción automática de batería. Para ello, haremos foco en la detección de eventos y daremos un mayor énfasis en evaluar modelos de reconocimiento del instrumento. Intentaremos garantizar la robustez del sistema tanto en presencia de múltiples instrumentos, como en presencia de la batería solamente. Para ello, pondremos a prueba un algoritmo de separación de fuentes.

Primero, haremos un análisis exploratorio de nuestro conjunto de datos. Evaluaremos si existen patrones observables tanto en el dominio del tiempo como aplicando la transformada de Fourier en el tiempo (STFT), para obtener una representación tiempo-frecuencia. En principio se aplica con la pista de batería aislada y luego se verifica el efecto que tiene incorporar el resto de los instrumentos sobre dicha representación tiempo-frecuencia. Vinculado a esto, mostramos qué ocurre en la representación cuando a las señales mixtas (con todos los instrumentos) le aplicamos un algoritmo de separación de fuentes. Esto se muestra con mayor detalle en la sección 3.1.

Abordaremos la tarea de reconocimiento de instrumentos con diferentes enfoques. Desde algunos enfoques clásicos como máquinas de soporte vectorial (SVM) hasta otros basados en lenguaje natural tal como Contrastive Language Audio Pretraining (CLAP) [8].

[3] https://en.wikipedia.org/wiki/Tatum_(music)
[4] https://github.com/tomasmarcos/drums-ml/

Un primer enfoque utilizado es el de máquinas de soporte vectorial, de manera similar a [3]. La diferencia con dicho trabajo es que en el presente no utilizamos clasificación bar-wise basada en patrones, sino que utilizamos varios SVM binarios (uno para cada tipo de sonido). Los detalles de este modelo planteado pueden encontrarse en la sección 3.2.1.

El segundo enfoque planteado es aquel basado en redes convolucionales al igual que [5], diferenciándonos de este en algunos detalles de la arquitectura y principalmente, que utilizamos solamente un canal sin filtros de entrada en la transformada de Fourier en el tiempo, en lugar de un multicanal con filterbanks como se efectúa en aquel trabajo. Esta familia de modelos será explorada en la sección 3.2.2.

De manera similar al enfoque presentado en el párrafo anterior, se desprende que la tarea de TAB puede resolverse sin una capa de detección de eventos previa ni tampoco tomando un clasificador binario por cada tipo de instrumento. Es decir, puede resolverse tomando una red cuya salida sea un vector de unos y ceros donde el uno represente si ocurre el instrumento j-ésimo en dicho momento del tiempo. La desventaja de este enfoque es que cada tipo de instrumento debe compartir una misma representación del espacio de características: No necesariamente puede existir una representación de características (features) que permita tanto detectar la presencia de múltiples sonidos correctamente; como contracara, tener una representación compartida de las características muy posiblemente conlleve a mayor robustez del modelo y una mayor eficiencia computacional a la hora de entrenar e inferir. Este tipo de modelos vectoriales, los exploraremos en la sección 3.2.3.

Además se evaluarán dos enfoques de reconocimiento del instrumento basados en transferencia del aprendizaje (transfer-learning): El primero, utilizando "Music undERstanding model with large-scale self-supervised Training" (MERT) [9] para la extracción de características y luego aplicando una red CNN más pequeña. Esto será presentado en la sección 3.2.4. El segundo, utilizando Contrastive Language Audio Pretraining (CLAP) [8], donde utilizamos diversos prompts para clasificar los instrumentos de batería y hacemos fine-tuning de este modelo. Sobre esto, es importante mencionar que también exploramos la alternativa zero-shot, que ofrecen este tipo de modelos basados en lenguaje.

Luego, se utilizan modelos de detección de eventos en lugar de las etiquetas de verdad de campo o groundtruth. El objetivo subyacente es verificar si la robustez del sistema se mantiene al prescindir de estas etiquetas. Para abordar este problema, se plantearon tres enfoques simples: dos basados en redes (CNN, RNN) y uno basado en modelos no entrenables [10]. Este tipo de modelos se describen en la sección 3.3.

Como último paso, se evalúa si la incorporación de la pista completa en lugar de la batería aislada, conlleva a un detrimento de los modelos de reconocimiento del instrumento y detección de eventos. Se propone incorporar un algoritmo de separación de fuentes como paliativo, conocido como HybridDemucs [11].

También se visualiza en la sección de análisis exploratorio de datos, el resultado de aplicar este modelo de separación de fuentes sobre una señal mixta y no de batería aislada, es decir sobre una señal con todos los instrumentos.

Al igual que el análisis exploratorio de datos, todos los modelos serán evaluados sobre el conjunto de datos conocido como "MDB Drums" [12], que contiene pistas de batería y será presentado en la sección 2. Cabe destacar, que este conjunto de datos cuenta con anotaciones de platillos (CY) y otros instrumentos que serán utilizados para entrenar los modelos y reportar sus respectivas métricas.

Los resultados de los tres tipos de modelos planteados, se presentan en la sección número 4. En esta sección, se realiza una comparación de los algoritmos de reconocimiento del instrumento, así como los de detección de eventos. Los algoritmos ganadores de la comparación se combinan en dos nuevos experimentos: Un primer experimento utilizando las pistas de batería que consiste en aplicar detección de eventos y luego reconocimiento del instrumento sobre los eventos detectados. Un segundo experimento utilizando las pistas con todos los instrumentos (full mix), que al igual que el primero se aplica detección de eventos y posteriormente reconocimiento del instrumento. Pero previo a esto, se utiliza un modelo de separación de fuentes para aislar la batería. Esto se compara con un mismo caso, pero sin aplicar dicho modelo de separación de fuentes.

También existe una desagregación a nivel pista para los modelos de reconocimiento del instrumento y el de separación de fuentes.

Finalmente, se presentan algunas conclusiones y posibles mejoras para trabajos futuros, como lo es, utilizar datos sintéticos para incrementar el tamaño muestral del conjunto de entrenamiento.

2. Conjunto de datos

Con el objetivo de entrenar y evaluar los modelos de detección de eventos y clasificación de sonidos, utilizamos el conjunto de datos MDB [12]. Este conjunto se basa en el subconjunto MusicDelta del conjunto de datos MedleyDB; consiste en 23 grabaciones no-sintéticas (real-world recordings) con una variedad de géneros musicales. A su vez, estas 23 grabaciones contienen en su totalidad 7994 onsets. Estos onsets se dividen en 6 clases y en 21 subclases, pero en nuestro caso sólo utilizaremos las siguientes 6 clases: KD (bombo), SD (redoblante), HH (hi-hat), OT (pandereta/side stick), TT (high/mid/low tom) y CY (cymbal o platillo). Mas precisamente, se utilizan 4 de estas 6 clases para la mayoría de los experimentos (KD, SD, CY, HH), exceptuando algunos de casos de reconocimiento del instrumento como SVC, CLAP y CNN binarios. Esto se debe a que se obtuvieron F1-Scores casi nulos, por lo que no proseguimos a verificar los resultados de los modelos restantes, tal como veremos en la sección 4.1.

Table 1. Onset classes in *MDB Drums* dataset.

Class	Subclass	Description	Onsets
KD	KD	kick drum	1539
SD	SD	snare drum	1510
	SDB	snare drum: brush	332
	SDD	snare drum: drag	2
	SDF	snare drum: flam	11
	SDG	snare drum: ghost note	790
	SDNS	snare drum: no snare	9
HH	CHH	hi-hat: closed	1847
	OHH	hi-hat: open	269
	PHH	hi-hat: pedal	523
TT	HIT	high tom	4
	MHT	high-mid tom	26
	HFT	high floor tom	14
	LFT	low floor tom	46
CY	RDC	ride cymbal	835
	RDB	ride cymbal: bell	16
	CRC	crash cymbal	126
	CHC	china cymbal	15
	SPC	splash cymbal	10
OT	SST	side stick	38
	TMB	tambourine	32

Figura 1: Divisiones y subdivisiones correspondientes a las anotaciones del dataset MDBdrums (copia de la tabla proveniente del paper original) [12]

Cada grabación musical cuenta con dos versiones: una donde únicamente se ejecuta la batería (pistas aisladas), y otra donde acompañan también otros tipos de instrumentos (por ejemplo: guitarra, voz, bajo). En este trabajo usaremos las pistas aisladas de batería para la gran mayoría de los experimentos, exceptuando uno donde utilizaremos la pista con todos los tipos de instrumentos. Este último experimento consiste en verificar si es factible aún el sistema de TAB en presencia de dichos instrumentos; en caso afirmativo, evaluar la pérdida de performance del mismo en caso de que esta existiese.

A su vez, estas grabaciones las partimos en un dataset de desarrollo y otro de heldout (test), en concordancia con [12][5].

[5]El archivo de las particiones se encuentra aquí https://github.com/CarlSouthall/MDBDrums/blob/master/MIREX2017.md (última revisión 10/01/2023).

Training Set	Test Set
MusicDelta_80sRock	MusicDelta_Beatles
MusicDelta_BebopJazz	MusicDelta_Country1
MusicDelta_Britpop	MusicDelta_FreeJazz
MusicDelta_CoolJazz	MusicDelta_Gospel
MusicDelta_Disco	MusicDelta_Grunge
MusicDelta_FunkJazz	MusicDelta_Hendrix
MusicDelta_FusionJazz	MusicDelta_LatinJazz
MusicDelta_Reggae	MusicDelta_ModalJazz
MusicDelta_Rock	MusicDelta_Punk
MusicDelta_Rockabilly	MusicDelta_SpeedMetal
MusicDelta_Shadows	MusicDelta_SwingJazz
MusicDelta_Zepplin	

Cuadro 1: Partición en train/test correspondiente al trabajo de MDB Drums; o equivalentemente, developement/heldout en el presente trabajo. Training para nosotros representa desarrollo (train+validation); mientras que test representa out of sample (heldout dataset).

Es importante notar que lo que se conoce como "training set" (conjunto de datos de entrenamiento) en la figura 1, se utilizará para el desarrollo de los modelos (internamente, se realiza una partición de este conjunto en entrenamiento y validación); mientras que lo que se conoce como "test set" en tal figura se utilizará para mostrar los resultados finales y se conocerá también como out of sample (fuera de muestra) o heldout set.

En lo que respecta al balance de datos, podemos observar la cantidad de eventos por tipo de instrumento y partición a continuación en la tabla 2. Se encuentra un total de 3732 eventos para el conjunto de desarrollo y 4138 para el conjunto de prueba o heldout, sumando un total[6] de 7870 eventos.

Clase	**Desarrollo**		**Heldout**	
	Cantidad de eventos	**Porcentaje**	**Cantidad de eventos**	**Porcentaje**
HH	1542	41.32	1036	25.04
KD	624	16.72	878	21.22
SD	1280	34.3	1350	32.62
CY	265	7.1	735	17.76
TT	15	0.4	75	1.81
OT	6	0.16	64	1.55
Total	3732	100	4138	100

Cuadro 2: Cantidad de eventos para el conjunto de desarrollo y heldout, separado por tipo de instrumento.

Cuando evaluamos por tipo de instrumento, podemos observar que para las clases TT y OT la cantidad de eventos en el conjunto de desarrollo puede resultar insuficiente como para entrenar un modelo. Esto explica la obtención de resultados nulos o muy bajos para este tipo de instrumentos tal como veremos en secciones posteriores. Un caso similar pero de mucha menor gravedad se observa para el CY. En el agregado, se observa un cierto desbalance de clases propio de los ritmos de batería, el cuál intentamos morigerar en la mayoría de los casos de reconocimiento del instrumento.

3. Metodología

3.1. Análisis exploratorio de datos

En este apartado, tomamos los primeros cinco segundos de la grabación llamada "MusicDelta_Rock_Drum.wav"de nuestro conjunto de datos MDB drums. Es importante notar que destacamos los primeros 5 segundos de esta grabación puesto que proviene de un ritmo popularmente utilizado

[6]Observamos una pequeña discrepancia respecto al paper 'MDB drums' en la cantidad de eventos ya que ellos mencionan que la totalidad suma 7994

cuando un usuario aprende batería, por su cualidad de sencillo. Las ocurrencias y el tipo de sonido de acuerdo a las anotaciones son las siguientes, que se reportan en el cuadro 3:

índice	onset_time	drum_type	onset_samples
0	0.000	KD	0
1	0.020	HH	441
2	0.280	HH	6174
3	0.528	SD	11648
4	0.550	HH	12127
5	0.830	HH	18301
6	1.062	KD	23424
7	1.090	HH	24034
8	1.360	HH	29988
9	1.620	SD	35712
10	1.640	HH	36162
11	1.910	HH	42115
12	2.154	KD	47488
13	2.170	HH	47848
14	2.450	HH	54022
15	2.694	SD	59392
16	2.720	HH	59976
17	2.990	HH	65929
18	3.260	HH	71883
19	3.274	KD	72192
20	3.540	HH	78057
21	3.802	SD	83840
22	3.830	HH	84451
23	4.090	HH	90184
24	4.370	HH	96358
25	4.389	KD	96767
26	4.640	HH	102312
27	4.894	SD	107903
28	4.910	HH	108265

Cuadro 3: Anotaciones, primeros 5 segundos de la canción "MusicDelta_Rock_Drum.wav"del conjunto de datos MDB Drums. Los valores de la columna "drum_type" conocidos como KD, HH y SD representan el bombo (kick drum), Hi-Hat y redoblante (snare drum) respectivamente.

Donde:

- **onset_time**, es el segundo donde ocurre el sonido.
- **drum_type**, es el tipo de sonido que ocurre.
- **onset_samples**, es el número de muestra donde ocurre el sonido (samplingRate multiplicado por onset_time).

Debe notarse que el número de muestra o onset_samples es lo que se verá en el eje de abscisas de los siguientes gráficos. El lector deberá poder hacer la correspondencia entre cada sonido marcado en la tabla y uno de estos gráficos.

3.1.1. ¿Es posible detectar el sonido de una batería?

Un interrogante que puede surgir previo a intentar detectar si estamos en presencia de un sonido de batería y qué tipo de instrumento ocurre, consiste en observar si existen patrones perceptibles ante el ojo humano, que puedan dar cuenta de esto. La respuesta depende del tipo de grabación a analizar. En nuestro caso, refiriéndonos a los primeros cinco segundos de la pista mencionada anteriormente, podemos evaluar la figura 2, que se presenta a continuación:

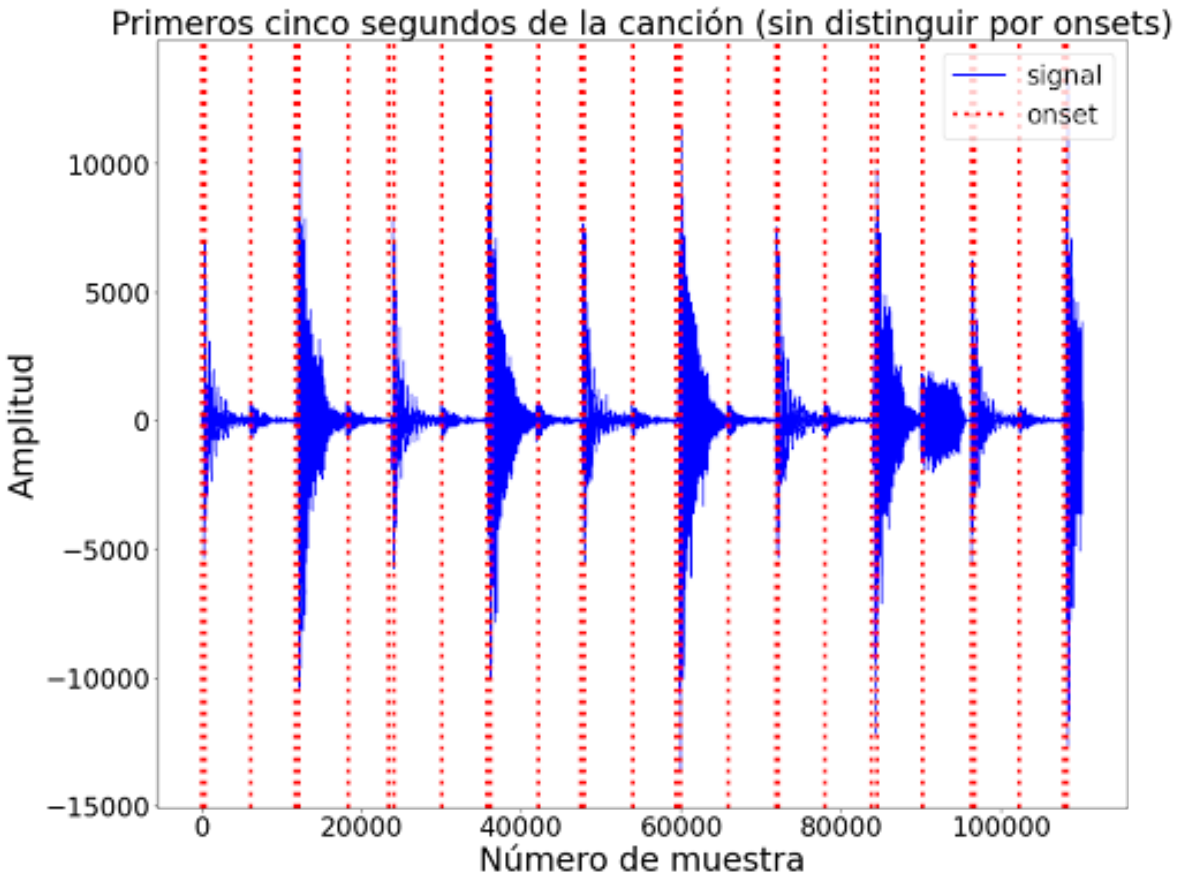

Figura 2: Amplitud en cada muestra (representación de la señal en el dominio del tiempo), sin distinguir el tipo de instrumento. Las barras rojas representan los onsets, y la línea azul, la señal.

En donde puede verse la señal cruda (dominio del tiempo) como una serie de tiempo, donde en el eje de abscisas se ve el número de muestra (onset_sample); mientras que en el de ordenadas, se puede observar la amplitud de la señal. Puede observarse que existen incrementos en la amplitud asociados a la presencia del sonido (línea punteada roja). A medida que el sonido se extingue, la amplitud disminuye. A su vez, se observan diferencias entre las distintas amplitudes, por ejemplo el primer evento ocurre con mayor amplitud que el segundo y el tercero con mayor amplitud que sus dos predecesores. Esto evidencia que no sólo es posible distinguir la presencia de un sonido de batería, sino que también existen diferencias entre las señales que permiten predecir con cierta exactitud el tipo de instrumento que ocurre en este fragmento.

3.1.2. ¿Es posible distinguir qué sonido de batería ocurre?

La respuesta, al igual que en el caso anterior, depende de la grabación a analizar. En nuestro caso, graficamos la señal resultante de los cinco segundos de la canción:

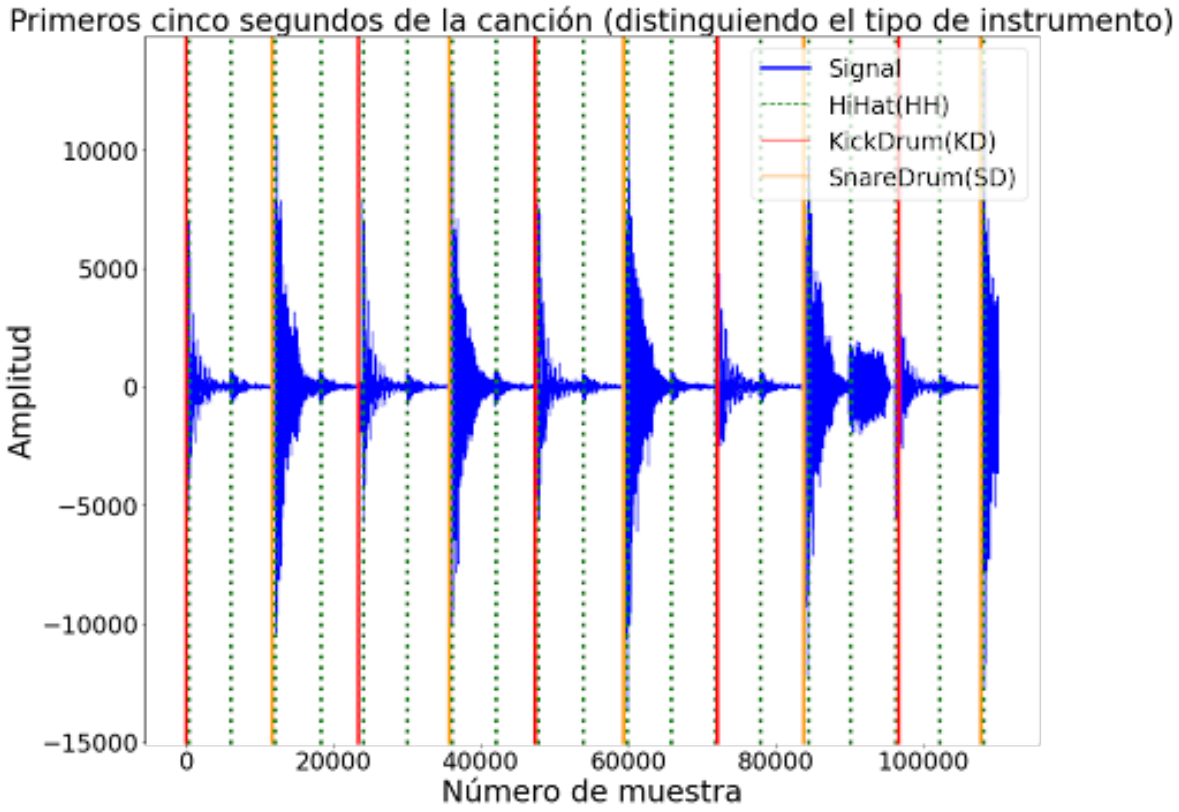

Figura 3: Amplitud en cada muestra (representación de la señal en el dominio del tiempo), distinguiendo el tipo de instrumento. Las barras de los distintos colores representan onsets correspondientes a distintos instrumentos; La línea azul, la señal. Notar que en varios casos - como el primer onset - ocurre más de un sonido al mismo tiempo (HH+KD).

Observando con detenimiento la figura 3, notamos que el bombo o KD (marcado en rojo) puede distinguirse de un Hi-Hat solo (marcado en verde, línea punteada) ya que posee mayor amplitud; a su vez, el redoblante posee mayor amplitud que ambos sonidos señalados anteriormente. Dicho en otras palabras, podemos distinguir visualmente estos tres casos puesto que algunos suenan más fuerte que otros. Sobre este punto, debe aclararse que estas interpretaciones no siempre son así y existen excepciones tales como las "ghost notes"[7] en el redoblante, donde puede que allí el redoblante sea el sonido de menor amplitud de todos.

La figura número 4, corresponde al espectrograma de los 5 segundos de dicha grabación. Para realizarla, se tomaron 100 frames por segundo (FPS) y un tamaño de ventana de 2048 muestras, por lo que en el eje de ordenadas tenemos como máximo 1024 frecuencias (bins) disponibles (2048/2). Como son 5 segundos de canción, podemos ver 500 capturas o frames.

[7] https://en.wikipedia.org/wiki/Ghost_note

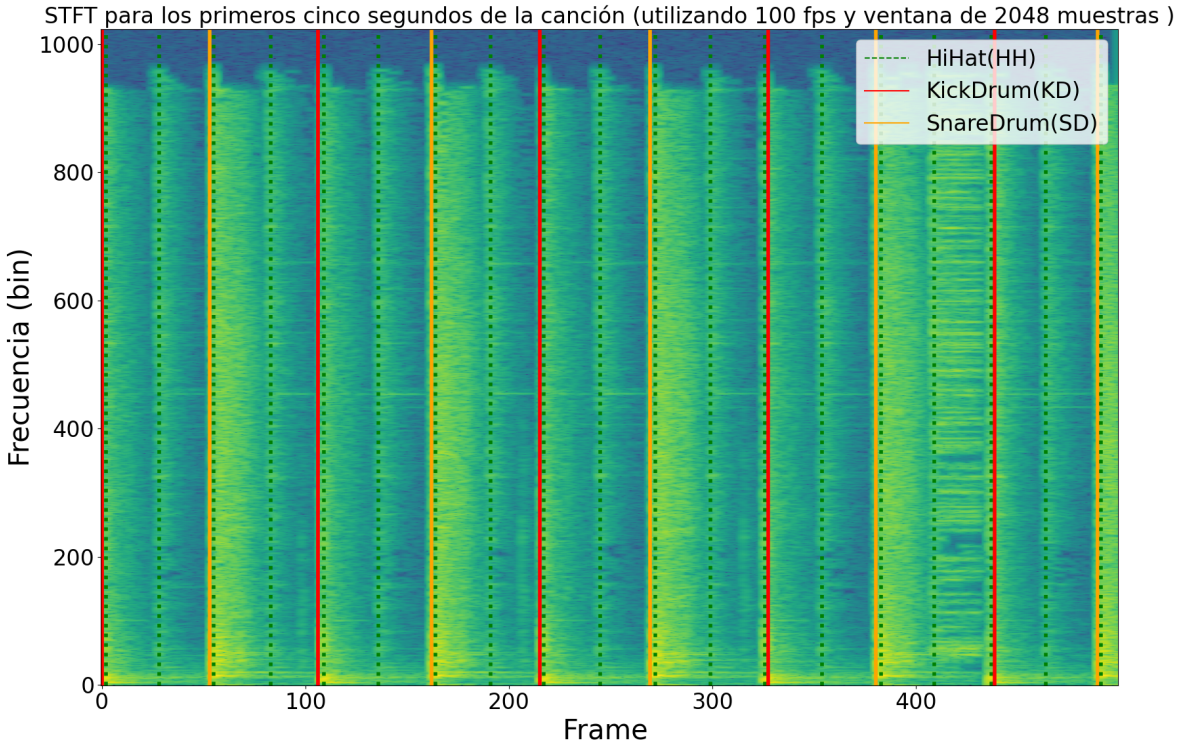

Figura 4: Representación de la señal en el dominio tiempo-frecuencia (amplitud_ventana = 2048; FPS = 100). Los colores más claros (oscuros), representan mayor (menor) intensidad en la frecuencia, en ese momento del tiempo (frame). Las barras sólidas horizontales representan los diferentes instrumentos. Se aplica una transformación logarítmica base e para una mejor visualización.

Puede notarse que los sonidos de frecuencias más bajas se corresponden con un bombo (línea roja) + Hi-Hat (línea verde punteada); mientras que cuando ocurre un redoblante, no predominan las frecuencias bajas sino también las medias e incluso altas. Si bien en esta gráfica no es tan clara, puede distinguirse el Hi-Hat como una señal corta a diferencia del redoblante y sin predominancia en las frecuencias bajas a diferencia del bombo.

La figura 5 es similar a la figura 4 pero se aplica una transformación logarítmica base 10 y posee un filtro de 12 bandas por octava aplicado previamente. Allí se pueden ver patrones similares pero se observa con mayor claridad las frecuencias predominantes, principalmente del bombo y redoblante.

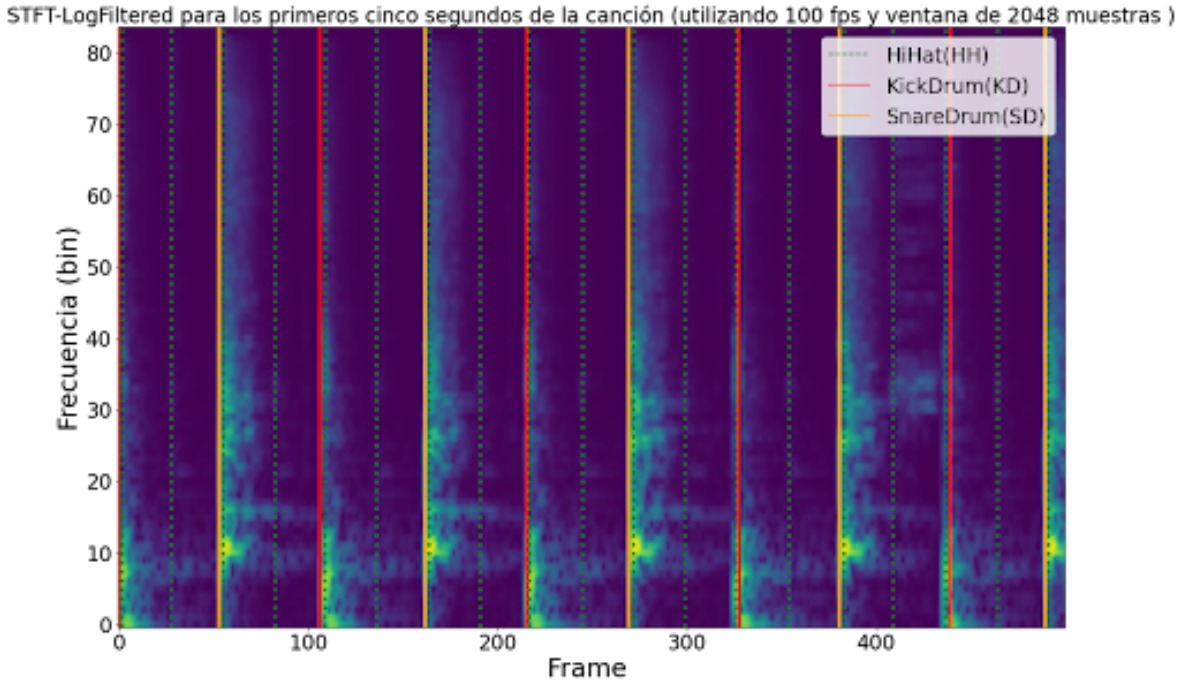

Figura 5: Representación de la señal en el dominio tiempo-frecuencia (amplitud_ventana = 2048; FPS = 100) pero en escala logarítmica y con la aplicación previa de un filterbank logarítmico. Los colores más claros (oscuros), representan mayor (menor) intensidad en la frecuencia, en ese momento del tiempo. Las barras sólidas horizontales representan los diferentes instrumentos.

3.1.3. Distinción en presencia de otros instrumentos.

En secciones anteriores, vimos que podían observarse patrones visuales por los cuales podíamos distinguir un instrumento de batería de otro, como por ejemplo, el bombo del redoblante puesto que en el primero las frecuencias más bajas ocurren con mayor intensidad que en el segundo y en este último sí ocurren frecuencias medias con cierta intensidad. Este análisis de los primeros 5 segundos de la pista, ocurrió cuando la misma tenía solamente sonidos de batería. Esto desprende el siguiente interrogante: ¿Qué ocurre cuando usamos la canción con todos los instrumentos y no solo la batería? Por suerte, el conjunto de datos MDB Drums, posee no solo la pista de batería aislada sino que también, su versión en presencia de todos los instrumentos tales como: voz, bajo, batería, guitarra y otros.

Inspeccionemos por un momento la pista con todos los instrumentos en la figura 6. Podemos observar que si bien podemos distinguir algunos patrones propios de los instrumentos de batería, tales como las frecuencias bajas del bombo (punto amarillo al inferior de todo), distinguir cada instrumento se vuelve más difícil que antes.

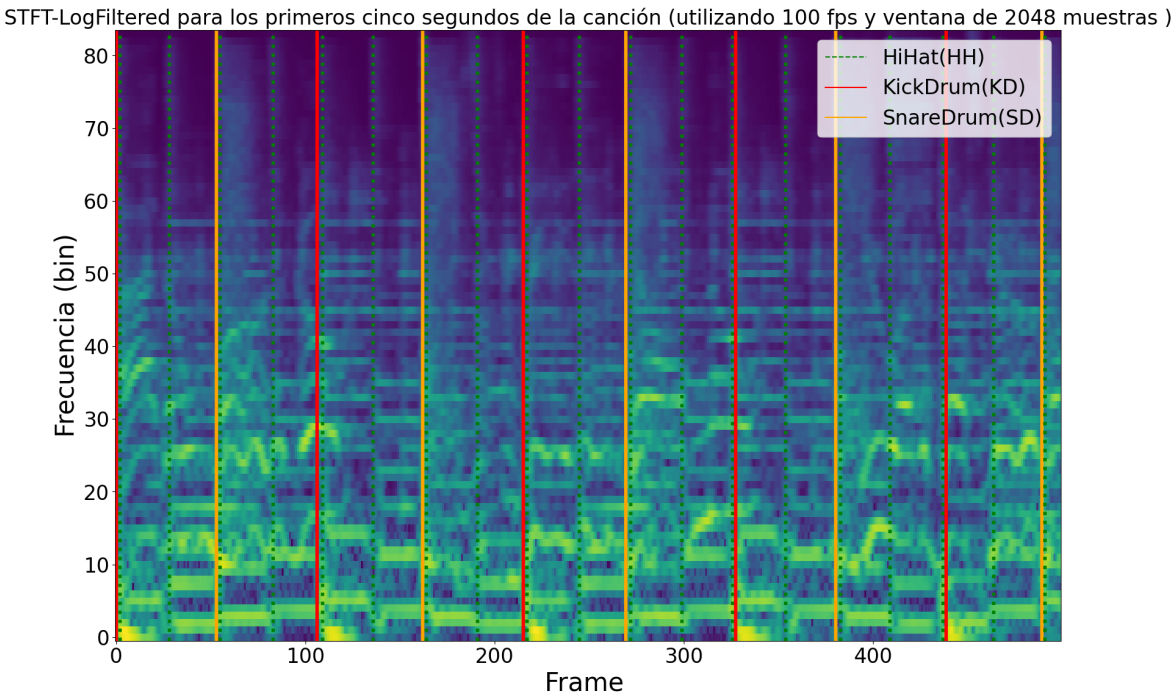

Figura 6: Representación de la señal, con todos los instrumentos (no solo batería), en el dominio tiempo-frecuencia (amplitud_ventana = 2048; FPS = 100) pero en escala logarítmica y con la aplicación previa de un filterbank logarítmico. Los colores más claros (oscuros), representan mayor (menor) intensidad en la frecuencia, en ese momento del tiempo. Las barras sólidas horizontales representan los diferentes instrumentos.

Este incremento en la dificultad para distinguir los patrones de la batería en el espectrograma, posiblemente ocurra también para un algoritmo de aprendizaje automático. Por ello, puede que utilizar algoritmos de separación de fuentes y aislar la pista de batería ayude a la tarea de TAB cuando estamos en presencia de la señal que contiene al mix completo de instrumentos.

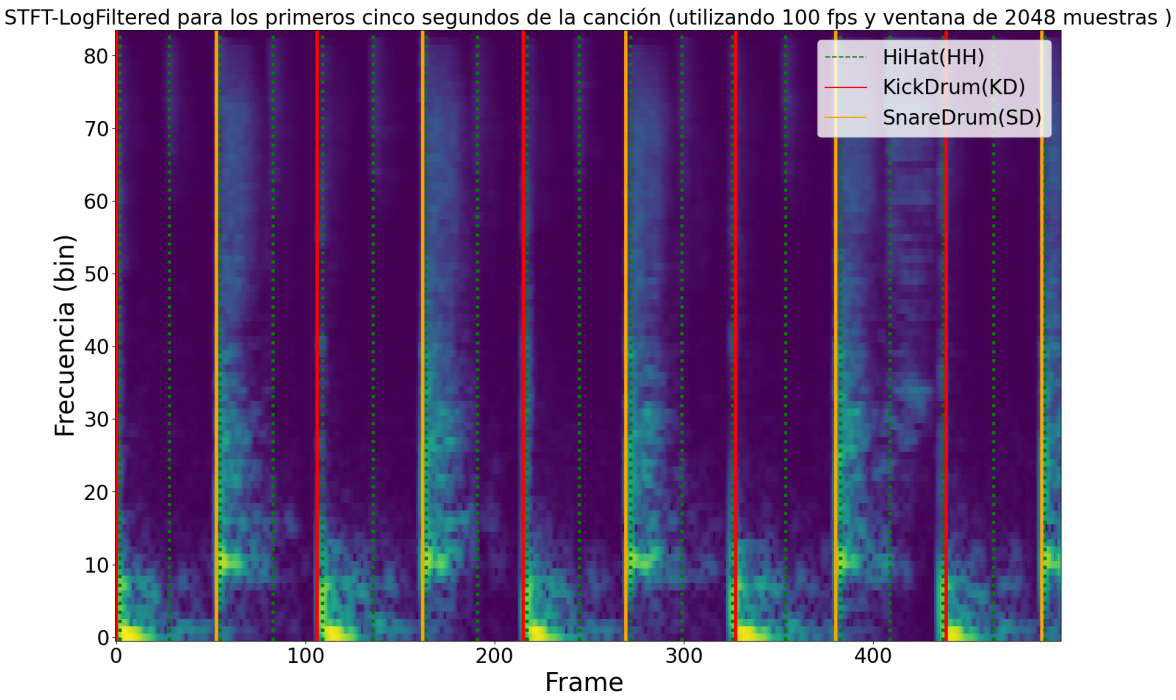

Figura 7: Representación de la señal, con todos los instrumentos (no solo batería), en el dominio tiempo-frecuencia (amplitud_ventana = 2048; FPS = 100) pero en escala logarítmica y con la aplicación previa de un filterbank logarítmico. Los colores más claros (oscuros), representan mayor (menor) intensidad en la frecuencia, en ese momento del tiempo. Las barras sólidas horizontales representan los diferentes instrumentos. A diferencia de la figura anterior, aquí se aplica separación de fuentes para aislar la pista de batería.

La figura 7 representa el espectrograma tras aplicar un modelo preentrenado de separación de fuentes conocido como Hybrid-Demucs, el cual presentaremos en la sección 3.4. Puede notarse que tras aplicar la separación de fuentes, distinguir los patrones de cada instrumento de batería resulta más fácil e incluso estos patrones son similares o iguales a los que observamos en la pista de batería aislada (sin separar fuentes).

Tras una observación manual de las señales pertenecientes a las figuras 5 y 7 puede notarse que los audios de estas son similares, lo cual representa otra cara de la misma moneda, en el sentido de que los espectrogramas resultan parecidos.

3.2. Algoritmos de reconocimiento del instrumento (recognition)

A continuación, presentaremos las metodologías aplicadas para los diferentes algoritmos de reconocimiento del instrumento. A cada metodología, por simplicidad, dividimos en cinco partes: (1) Breve introducción teórica; (2) Preprocesamiento de etiquetas (labels); (3) Preprocesamiento de la señal; (4) Partición del conjunto de datos y por último (5) Detalles del clasificador utilizado.

3.2.1. Reconocimiento vía SVM

3.2.1.1. Introducción teórica

Máquinas de soporte vectorial[8] es un algoritmo de clasificación binaria cuyo objetivo es encontrar el hiperplano separador de máximo margen (máxima separación entre las observaciones de distinta clase), sujeto a que todas las observaciones se clasifiquen correctamente. El algoritmo en su versión primal, busca resolver el siguiente problema de optimización:

[8]Aquí presentamos una breve introducción al algoritmo de SVM, por lo cual se omiten algunos detalles tales como la presentación del algoritmo en su versión dual, que habilita a explicar el truco del kernel con mayor precisión por ejemplo. Para una lectura en mayor profundidad se recomienda http://cs229.stanford.edu/summer2020/cs229-notes3.pdf

$$\min_{w,b} \frac{1}{2}||w|| \quad s.a \quad y_i(w^T\ x_i + b) \geq 1$$

$$i = 1, 2, .., n$$

Notar que:

- $x_i \in \mathbb{R}^P$ es el vector del conjunto de datos, cada i representa una observación del mismo con $(i = 1, .., n\ )$; n es el tamaño muestral.
- $w \in \mathbb{R}^P$ es el vector de pesos del hiperplano separador; P es la cantidad de variables en el conjunto de entrenamiento; $b \in \mathbb{R}$, es el sesgo o bias.
- $\frac{1}{2}||w|| = \frac{1}{2}w^T w$ es la inversa del margen (maximizar el margen es equivalente a minimizar su inversa).
- $y_i(w^T x_i + b) \geq 1$ está exigiendo que cada observación i-ésima se clasifique correctamente, puesto que $y_i \in \{-1, 1\}$, entonces cuando ocurra $w^T x_i + b > 0$ la predicción es positiva, luego $y_i = 1$ para satisfacer la restricción (exige que sea clase positiva o 1 también); lo mismo ocurre si $w^T x_i + b < 0$ donde se exigirá $y_i = -1$ para satisfacer la restricción. Esto se conoce como clasificador de margen duro (hard margin classifier) puesto que la restricción no admite que alguna observación sea clasificada incorrectamente.

En general, y en nuestro caso no es la excepción, se suele relajar la restricción de que todas las observaciones sean clasificadas correctamente puesto que rara vez es posible hacerlo, y en muchos casos puede llevar a sobreajuste. Tras relajar este supuesto, el problema de optimización es similar pero se complejiza levemente, tal como se muestra a continuación:

$$\min_{w,b,\zeta} \frac{1}{2}||w|| + C\sum_{i=1}^{n} \zeta_i$$

$$s.a \quad y_i(w^T\ x_i + b) \geq 1 - \zeta_i\,; \quad \zeta_i \geq 0$$

$$i = 1, 2, .., n$$

Notar que:

- ζ_i es la variable de holgura, que actuará cuando ocurra que alguna observación sea clasificada incorrectamente. Es decir que cuando ocurra $y_i(w^T\ x_i + b) < 1$ para algún i, ζ_i tomará un valor positivo y lo suficientemente grande, de manera que la restricción $y_i(w^T\ x_i + b) \geq 1 - \zeta_i$ se satisfaga de todos modos. Para penalizar que ocurra esto, se incorpora el término $\sum_{i=1}^{n} \zeta_i$ como parte de la función objetivo, ponderado por el parámetro C. Cuanto mayor sea el parámetro C, mayor será la penalización como consecuencia de clasificar una observación incorrectamente. En nuestro caso, buscaremos el hiperparámetro C por búsqueda en grilla (gridsearch).

Aún nos encontramos con el problema de que las clases no necesariamente son linealmente separables, es decir que encontrar un hiperplano puede que no sea la mejor opción. Para dar solución a esto, se plantea el "truco del kernel" (también conocido como "kernel trick"), donde se busca una alternativa a "transformar el conjunto de datos" en otro de dimensión mayor, de manera tal que allí las clases sean linealmente separables y sin incurrir en el costo computacional que conlleva transformar dicho conjunto en una dimensión superior (por ejemplo creando el cuadrado y cubo de cada una de las variables del conjunto de datos).

3.2.1.2. Preprocesamiento de etiquetas

En lugar de entrenar un modelo multiclase, se entrenaron varios modelos de clasificación binaria, por lo que hubo que transformar las etiquetas de manera que indicasen si ocurre el instrumento del tipo a clasificar o no. A modo de ejemplo, mostramos a continuación las tablas antes de la binarización (cuadro 4) y después de la misma (cuadro 5) para el caso del Hi-Hat (HH). Notar que lo mismo haremos para el resto de los tipos de instrumentos.

índice	onset_time	drum_type
0	0	KD
1	0.02	HH
2	0.28	HH
3	0.528254	SD

Cuadro 4: Ejemplo de anotaciones previo a la binarización de etiquetas, para el caso del instrumento tipo Hi-Hat.

índice	onset_time	drum_type
0	0	OTHER
1	0.02	HH
2	0.28	HH
3	0.528254	OTHER

Cuadro 5: Ejemplo de anotaciones posterior a la binarización de etiquetas, para el caso del instrumento tipo Hi-Hat.

Una vez que tengamos binarizadas nuestras etiquetas, para cada onset tomaremos una ventana de $\pm\Delta ms$ de su señal para obtener la tupla (señal, etiqueta) que representará nuestras features y target respectivamente. Esto se verá con mayor detalle a continuación en la sección 3.2.1.3.

A su vez, si ocurre un sonido en la ventana de ± 50ms, se lo elimina, por lo que el onset de índice 0 del cuadro 5 no se incluye para el entrenamiento. Esto se realiza para garantizar la pureza del sonido (aunque introduce un sesgo), ya que si bien el evento de índice 0 y el de 1 tendrían una señal muy parecida, en el primer caso la etiqueta indicaría que no ocurre este sonido mientras que en el segundo lo contrario; y dado que casi tienen la misma señal, puede considerarse como agregar ruido al entrenamiento. A esta altura es importante aclarar que en la evaluación de los algoritmos, tanto en entrenamiento como out of sample, no se remueve ningún onset.

3.2.1.3. Preprocesamiento de la señal

Para generar los modelos de reconocimiento del instrumento, utilizamos el algoritmo señalado en el cuadro 6:

Algoritmo de preprocesamiento de la señal
1) Identificar el segundo donde ocurre el onset en la anotación. 2) Tomar una ventana de $2 \times \Delta ms$ (delta milisegundos) centrada en el onset (en nuestro caso $2 \times 50ms = 100ms$). 3) Realizar padding correspondiente para garantizar que todas las ventanas de la señal tengan la misma dimensión o shape (si la señal es de 100ms y la $samplingRate = 22050$ entonces la ventana tendrá $22050 \times 0{,}1 = 2250$ muestras. En consecuencia, el padding debe garantizar que la señal tenga al menos 2250 elementos. El padding se hace de manera tal que la señal quede centrada (tenga igual cantidad de ceros a derecha e izquierda). 4) Tomar la transformada de Fourier en el tiempo (STFT) de dicha señal con zero-padding.
Parámetros utilizados: Se aplicó la STFT con una amplitud de ventana de 1024 muestras, y un stride de 256 muestras. La cantidad de milisegundos desde el onset fueron 50ms hacia adelante y lo mismo hacia atrás. A su vez utilizamos padding de la señal transformándola en 5120 muestras. Notar que, tomar una cierta cantidad de milisegundos hacia atrás garantiza capturar el ataque del sonido [13] y prevenir micro-errores en el etiquetado manual.

Cuadro 6: Algoritmo para el preprocesamiento de los modelos de reconocimiento del instrumento: SVM y CNN.

Posterior a la aplicación del algoritmo anterior, se obtuvieron múltiples señales de dimensión $513 \times$

17[9], utilizamos el aplanado (flatten) de esa señal para nuestro SVM y normalización de Z-score[10], lo que resultó en clasificadores de $513 \times 17 = 8721$ características. En este punto, es importante notar que esta es una de las debilidades de no aplicar filterbanks, ya que si lo hubiésemos hecho, la cantidad de variables sería mucho menor (ej: filter bank de 80 bins; $80 \times 17 = 1360$ variables), permitiendo entrenar modelos en mucho menor tiempo. La contracara de esto, es que no existe pérdida de información - posiblemente relevante para este algoritmo - como consecuencia del filtro.

3.2.1.4. Partición del conjunto de datos

La partición del conjunto de datos fue en un primer dataset de desarrollo y otro de heldout, de acuerdo con lo presentado en la sección 2. Dentro del conjunto de desarrollo, dividimos los eventos de este conjunto en entrenamiento (80 %) y validación (20 %) de manera aleatoria estratificando por instrumento.

En cuanto al tipo de señal utilizada, se utilizaron las pistas aisladas de batería para este modelo, tal como se señala posteriormente (sección 4.1).

3.2.1.5. Detalles del clasificador

Como fue mencionando anteriormente en la subsección 3.2.1.2 (preprocesamiento de etiquetas), se entrenaron varios clasificadores SVM binarios en lugar de uno multiclase, uno para cada tipo de instrumento a reconocer. A esta familia de clasificadores binarios también se la suelen conocer como clasificadores "OneVersusRest"[11] dado que el instrumento a reconocer se marca como el 'evento' y el resto de las cosas se las pone bajo la etiqueta de "otro".

Cabe destacar que en una señal pueden ocurrir más de un tipo de sonido al mismo tiempo, tal como fue ilustrado en la sección 3.1.2, donde en el primer evento, ocurre un sonido de bombo y uno de Hi-Hat al mismo tiempo (o casi), por lo que en ese pedazo de señal hubo 2 sonidos y no uno. Notar que si utilizamos un clasificador multiclase donde los tipos de sonido fuesen mutuamente excluyentes, supondría que sólo ocurre un sonido y no más de uno (ej: ocurre bombo pero no Hi-Hat o viceversa); es por eso que adoptamos varios clasificadores binarios en lugar de uno multiclase.

Para cada clasificador SVM, se aplicó optimización de hiperparámetros con búsqueda en grilla y validación cruzada con 5 particiones (folds) para seleccionar el mejor conjunto de hiperparámetros. Se entiende por mejor conjunto, aquel que maximice el F1-Score[12].

El espacio de hiperparámetros planteado se muestra en el cuadro 7.

Kernel	C	Otros parámetros
lineal	[0.01,0.1,1,2,5]	
polinomial	[0.01,0.1,1,2,5]	grados del polinomio: [2,3,5]
rbf	[0.01,0.1,1,2,5]	gamma: ['scale','auto',0.001,0.01,0.02,0.03,0.1,1]

Cuadro 7: Espacio de hiperparámetros elegido para la búsqueda en grilla del modelo SVC.

Posteriormente a la elección de los mejores hiperparámetros del modelo, los utilizamos para re-entrenar el modelo junto con todo el conjunto de entrenamiento sin dejar una de las 5 particiones fuera.

[9] El número 513 surge de que el tamaño de la ventana es 1024 y de qué una señal real es simétrica. Es por ello que podemos tomar los primeros 512+1 elementos (incluyendo el componente de corriente directa o direct current y también el Nyquist porque las señal es par). Por otro lado, el número 17, representa la cantidad de frames que entran en la señal tras ejecutar padding, moviendose de a 256 muestras (hop size). Debemos aclarar que por defecto la librería utlizada agrega padding de $n_{fft}/2 = 1024 = 512$ ceros de cada lado, por lo tanto: $1 + (4096 + 512 * 2 - 1024)/256 = 17$. Este nivel de padding puede considerarse por encima del óptimo, sin embargo, no creemos que bajarlo lleve a mejores o peores resultados aunque sí consideramos que su reducción incremente levemente la eficiencia computacional en entrenamiento e inferencia. Queda para trabajos futuros investigar el impacto de este parámetro.

[10] https://en.wikipedia.org/wiki/Standard_score

[11] https://scikit-learn.org/stable/modules/multiclass.html

[12] Si bien esta heurística se considera aceptable, es importante destacar que en ciertas situaciones existen mejores, como lo es maximizar el meanTestF1score - stdTestF1score, de esta manera teniendo en cuenta la estabilidad de los clasificadores a lo largo de las particiones.

3.2.2. Reconocimiento vía CNN (clasificación-multiclase)

3.2.2.1. Introducción teórica

Las redes neuronales convolucionales (CNN - Convolutional neural networks) son un tipo de arquitectura de red neuronal comúnmente utilizada para imágenes, que contiene la operación de convolución[13] como parte de sus capas.

Una red neuronal convolucional, se caracteriza por tener la siguiente arquitectura:

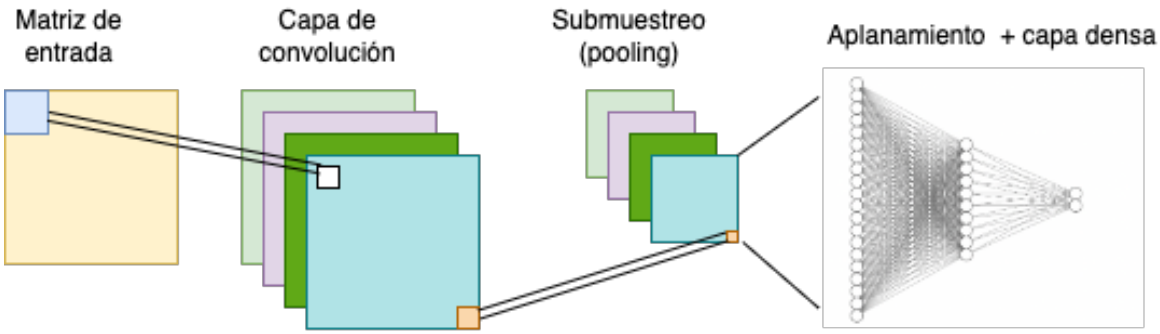

Figura 8: Ejemplo de una arquitectura simple de red convolucional. En nuestro caso, hablamos de matriz de entrada puesto que utilizamos un sólo canal con la STFT; si hubiésemos utilizado más de un canal, estaríamos hablando de un tensor de entrada en lugar de una matriz.

A continuación describiremos los elementos que consideramos más importantes en dicha arquitectura:

- **Capa Convolucional**: La capa convolucional consiste en convolver la matriz o tensor de entrada con un kernel o núcleo para generar una imagen filtrada, pero los pesos del kernel (elementos de esa matriz o tensor) son parámetros a encontrar como parte del método (no son fijos). En su versión discreta y bidimensional (conv2d):
  $h(x,y) = \sum_{dx=-a}^{a} \sum_{dy=-b}^{b} w(dx,dy) f(x-dx, y-dy)$; donde notamos que h(x,y) es el resultado de convolver la imagen f con el kernel w, en las coordenadas (x, y) de la imagen. Como dijimos, los elementos de la matriz kernel son parámetros de la red a ser encontrados por métodos como el de propagación hacia atrás (backpropagation).

  En la práctica, las librerías tradicionales como *pytorch* suelen usar la operación de correlación cruzada en lugar de convolución, puesto que se alcanzan resultados equivalentes a los fines prácticos pero con menor costo computacional.

- **Capa pooling**: La capa de pooling realiza una operación de agregación (generalmente máximo max-pooling o promedio avg-pooling) sobre la imagen, que permite reducir la dimensión de la imagen o feature maps resultante de la operación de convolución.

- **Operación de aplanamiento o flattening**: La operación de aplanamiento o flattening, consiste en transformar la matriz o tensor en un vector, donde cada uno de los componentes del mismo se corresponde con un elemento de la matriz, de manera ordenada.

- **Capa completamente conectada**: La capa completamente conectada o fully-connected, consiste en conectar cada neurona de entrada con cada una de las de salida/ocultas, mediante un peso w y además adicionar un peso b a la capa; tanto cada elemento del vector de pesos w como el escalar b pueden encontrarse vía el algoritmo de propagación hacia atrás.

- **Función de activación**: La función de activación introduce una no linealidad en las neuronas, de manera tal que facilite la representación de fenómenos complejos en la red. Una de las funciones de activación más usadas es la *ReLu*, que consiste en $f(x) = max(0, x)$, $x \in \mathbb{R}$; de manera tal que desactivará las neuronas cuyo resultado sea negativo. Otra, que también usaremos en este

[13] Como veremos luego, en la práctica se utiliza la operación de correlación cruzada puesto que alcanza los mismos objetivos que la convolución pero con un menor costo computacional.

trabajo, es la *softmax*,[14] función que toma un vector de $K > 1$ elementos $x \in \mathbb{R}^K$ y los transforma en un espacio de probabilidad mediante la siguiente expresión: $softmax(x)_i = \sigma(x)_i = \frac{e_i^x}{\sum_{j=1}^{K} e_j^x}$ donde el índice i representa el elemento i-ésimo del vector x.

- **Capa Dropout**: Se asume que cada neurona capta una característica distinta de la imagen y las neuronas deben aprender a no depender de la presencia de siempre la misma característica, y si esta no está presente poder detectar otras. Con ese fin, la capa dropout anula aleatoriamente algunas de las neuronas (multiplicar cada neurona por una $Bernoulli(p)$ donde p es un hiper-parámetro de la red). Esto suele brindarle mayor robustez a la red, a la falta de ciertas características. En la práctica de las librerías[15] tradicionales, no sólo se multiplica por $B \sim Bernoulli(p)$ a cada neurona, sino que adicionalmente se multiplica por el escalar $\frac{1}{1-p}$ (sólo durante la fase de entrenamiento de la red); para solucionar temas de escala durante la inferencia.

Estos elementos que conforman una red convolucional, serán utilizados secuencialmente en la sección 3.2.2.5, donde presentaremos la arquitectura de la red convolucional entrenada para cada tipo de instrumento, junto con otros detalles.

3.2.2.2. Preprocesamiento de etiquetas

Al igual que en el caso del reconocimiento vía SVM, que ya fue explicado con más detalle en la sección 3.2.1.2, las etiquetas fueron binarizadas y se quitaron aquellas que compartían parte de la señal.

3.2.2.3. Preprocesamiento de la señal

Para el preprocesamiento de la señal, también utilizamos el algoritmo planteado para el modelo de reconocimiento de SVM en la sección 3.2.1.3. La única diferencia es que no se utilizó el flatten de las características, sino que la entrada fue simplemente una matriz de 513x17, en lugar de las 8721 características resultantes de un aplanado (flattening). Además, se utilizó normalización Z-score para garantizar una convergencia más rápida de los gradientes.

Es importante aclarar que esta normalización no necesariamente lleve a mejores o peores resultados, simplemente se espera que permita disminuir la función de pérdida en una menor cantidad de iteraciones durante el entrenamiento en comparación con el caso donde no se normaliza[16]. Respecto a si esto puede mejorar los resultados o no, explorar dicha hipótesis queda fuera del alcance de este trabajo, aunque creemos que no existirán cambios mayores en la performance del modelo.

3.2.2.4. Partición del conjunto de datos

La partición en desarrollo y heldout fue la misma que la realizada en el caso anterior. El conjunto de datos de desarrollo fue partido en entrenamiento (80 %) y validación (20 %), del mismo modo.

Con el conjunto de entrenamiento y validación se hicieron varias pruebas de manera manual, variando la cantidad de epochs y los algoritmos de descenso de gradiente, junto con ligeros detalles en la arquitectura, como la capa dropout.

En cuanto al tipo de señal utilizada, esta varía experimento a experimento. Cuando evaluamos solamente los resultados de los modelos de reconocimiento del instrumento (sección 4.1) utilizamos las pistas de batería aisladas. Cuando buscamos evaluar la efectividad de los algoritmos de separación de fuentes, utilizamos en un primer caso los modelos de RI con las pistas con todos los instrumentos y lo comparamos con un segundo caso que fue aplicar primero separación de fuentes sobre las pistas con todos los instrumentos y luego entrenar los modelos CNN binarios con la predicción generada por el modelo de separación de fuentes (nos quedamos sólo con la batería) tal como será descrito en los respectivos experimentos (secciones 4.5 y 4.7).

[14] En realidad, la entropía cruzada implementada en pytorch (CrossEntropyLoss), incorpora indirectamente la función softmax. Para más detalle ver https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html.

[15] https://pytorch.org/docs/stable/generated/torch.nn.Dropout.html

[16] https://developers.google.com/machine-learning/crash-course/numerical-data/normalization

3.2.2.5. Detalles del clasificador

Al igual que en los modelos de SVM, se entrenaron varios clasificadores binarios en lugar de uno multiclase; pero en este caso de tipo red convolucional (CNN). La otra opción disponible a explorar, es un clasificador vectorial (tipo red neuronal), donde la salida sea un vector de dimensión $\mathbb{R}^K$ donde K es la cantidad de sonidos posibles, y donde la capa de salida tenga un 1 si ocurre ese tipo de sonido y 0 en caso contrario. La desventaja de esto, es que los pesos de la arquitectura para cada tipo de clasificador se comparten, lo que le da menor flexibilidad para la representación del espacio de características,[17] pero quizás mayor robustez.

No utilizamos un algoritmo de búsqueda de hiperparámetros particular (debido a la capacidad computacional disponible), sino que exploramos diferentes posibilidades manualmente, tales como cambiar el tipo de optimizador, la cantidad de epochs, el tamaño del lote (batch-size) y algunos detalles de la arquitectura (i.e variar capa dropout). La arquitectura final se muestra en la figura 9.

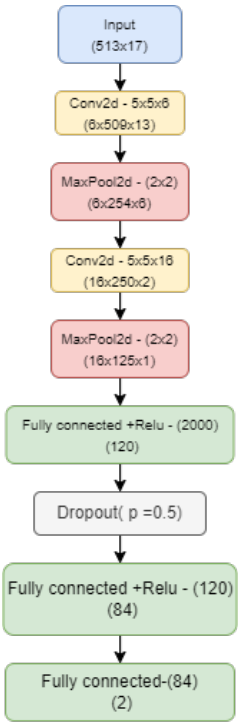

Figura 9: Arquitectura de la red CNN para la tarea de reconocimiento del instrumento. En la segunda línea entre paréntesis se encuentran las dimensiones, mientras que en la primera, los parámetros del modelo.

Finalmente, tras algunas iteraciones, se plantearon 100 épocas (epochs) con descenso de gradiente estocástico en mini batches (mini batch SGD) sin detención temprana (early stopping) para cada uno de los clasificadores binarios; dependiendo cada caso, se decidió detener el entrenamiento antes de culminar todas las epochs, mirando el conjunto de validación.

3.2.3. Reconocimiento vía CNN (clasificación-multietiqueta)

3.2.3.1. Introducción teórica

En la sección anterior (sección 3.2.2.1), se presentó la arquitectura de una red convolucional del tipo binaria. La diferencia con la red convolucional anterior es que en aquél caso la función de activación era tipo *Softmax*[18] pero en este caso nos vemos forzados a utilizar la función sigmoidea. La función sigmoidea es la siguiente: $y = sigmoid(x) = e^x/(1 + e^x)$.

En nuestro caso tendremos una función sigmoidea para cada tipo de instrumento, dentro de una misma

[17] Por ejemplo, en un clasificador multiclase, los pesos del kernel de convolución (conv2d) deberían compartirse para los distintos tipos de instrumentos; en el caso de usar múltiples clasificadores binarios, esto no pasa.

[18] También podríamos haber usado una función de activación sigmoidea. Se probó la red CNN con esta función de activación y se obtienen resultados similares, pueden encontrarse en el repositorio de github

red. Es decir que tomaremos su representación vectorial, de manera que reciba de entrada un vector $x \in \mathbb{R}^{\mathbb{K}}$ y devuelva otro vector $y \in \mathbb{R}^{\mathbb{K}}$ donde K es la cantidad de instrumentos a predecir.

3.2.3.2. Preprocesamiento de etiquetas

Para el procesamiento de las etiquetas, se tomó a cada uno de los onsets ocurrentes de las anotaciones de verdad de campo. Para cada uno de ellos, se tomó una ventana de ±50 milisegundos, y aquellos onsets que estaban dentro de la ventana se consideraban como presentes en la señal. Puesto de otra manera, si en las anotaciones teníamos que en el segundo 0.51 existía un bombo, tomábamos la señal perteneciente al intervalo [0.51-0.05 ; 0.51+0.05) y filtrábamos los instrumentos presentes en dicho intervalo de acuerdo con las anotaciones. Esto resulta en la tabla 8 que se presenta a modo de ejemplo.

	Instrumento			
Onset time	KD	SD	HH	CY
0.020	1	0	1	0
0.290	1	0	1	0
0.296	1	0	1	0
0.570	0	1	1	0

Cuadro 8: Ejemplo del procesamiento de las etiquetas para la red CNN de tipo multi-etiquetas. Para la primera fila, el onset que ocurre en el segundo 0.02 en realidad representa la señal ocurrente en el intervalo [0.02-0.05, 0.02+0.05) y los instrumentos que ocurren en dicho intervalo son el bombo (KD) y el HiHat (HH); esto se representa ya que tienen un uno en dicha fila; si no ocurren tendrán un 0.

Cabe mencionar que a diferencia de casos anteriores, no se eliminaron los onsets lo suficientemente cercanos. Esto tiene el efecto de que puede que le estemos presentando data repetida o muy similar al modelo y representa algo que debería corregirse en trabajos futuros.

3.2.3.3. Preprocesamiento de la señal

Para el preprocesamiento de la señal, también utilizamos el algoritmo planteado para el modelo de reconocimiento de SVM en la sección 3.2.1.3. La única diferencia es que no se utilizó el flatten de las características, sino que la entrada fue simplemente una matriz de 513x17, en lugar de las 8721 características resultantes de un aplanado (flattening). Además, se utilizó normalización Z-score para garantizar una convergencia más rápida de los gradientes.

3.2.3.4. Partición del conjunto de datos

La partición en desarrollo y heldout fue la misma que la realizada en el caso anterior. El conjunto de datos de desarrollo fue partido en entrenamiento (80 %) y validación (20 %) del mismo modo.

Con el conjunto de entrenamiento y validación se hicieron varias pruebas de manera manual, variando la cantidad de epochs y los algoritmos de descenso de gradiente, junto con ligeros detalles en la arquitectura, como la capa dropout.

En cuanto al tipo de señal utilizada, se utilizaron las pistas aisladas de batería para este modelo, tal como se señala posteriormente (sección 4.1).

3.2.3.5. Detalles del clasificador

Como fue mencionado, aquí se utilizó un clasificador multi-etiquetas. La desventaja de esto, es que los pesos de la arquitectura para cada tipo de clasificador se comparten, lo que le da menor flexibilidad para la representación del espacio de características, pero quizás mayor robustez. Con esta red se busca explorar si el efecto de la pérdida de representación es superior al efecto de la mayor robustez como consecuencia de compartir los pesos para una misma representación. Resulta fácil de observar que esta arquitectura conlleva una mejora en términos de eficiencia computacional, ya que requiere mantener una sola red en memoria y hacer la inferencia una sola vez a diferencia de la red CNN anterior donde deberíamos mantener en memoria una red para cada tipo de instrumento y a su vez hacer su inferencia para cada uno de estos. En esta oportunidad no se incrementó el tamaño de la red, puesto que eso

implicaría un deterioro en su velocidad de inferencia y requisitos de memoria. Queda para trabajos futuros explorar si una red más compleja otorga mejoras de inferencia sin detrimento en sus resultados.

No utilizamos un algoritmo de búsqueda de hiperparámetros particular (debido a la capacidad computacional disponible), sino que exploramos diferentes posibilidades manualmente, tales como cambiar el tipo de optimizador, la cantidad de epochs, el batch-size y algunos detalles de la arquitectura (i.e variar capa dropout).

3.2.4. Reconocimiento apoyado en la extracción de características provistas por MERT

La comunidad del aprendizaje profundo ha presenciado un crecimiento exponencial en el interés del aprendizaje auto supervisado o self-supervised learning (SSL). Este aún permanece inexplorado en cómo construir un framework que permita aprender representaciones útiles partiendo de la señal cruda en una manera SSL [14]. En esta instancia probaremos las características generadas por este modelo, aplicadas a nuestra tarea de reconocer el instrumento ocurrente (recognition).

3.2.4.1. Introducción teórica

Nuestra aproximación consiste en utilizar en lugar de un espectrograma, el resultado de realizar inferencia con el modelo MERT [9] como características. Es decir, utilizar los estados ocultos (hidden states) de **M**usic Unde**r**standing **t**ransformer (MERT) como extractor de características.

MAP-MERT es un modelo de aprendizaje auto supervisado que en lugar de utilizar etiquetas predeterminadas, utiliza masked language modeling (MLM) al estilo BERT [15]. El proceso implica enmascarar aleatoriamente una porción de la señal de entrada y luego predecirla a partir de su contexto adyacente. Esto es análogo al uso de tokens enmascarados en BERT, pero aquí se trabaja con segmentos de audio enmascarados. Para ser más preciso, en realidad no se usa la señal cruda, sino que se reduce la dimensionalidad de las características de entrada. Entonces, en lugar de predecir la señal enmascarada se predice una reducción dimensional de la misma tal como se emplea en HuBERT [16] que utiliza clustering sobre los coeficientes Mel-Frequency Ceptral (MFCCs) para adquirir las pseudo etiquetas. Adicionalmente los autores mencionan que a diferencia de HuBERT para reconocimiento del habla, las características otorgadas por MFCCs son insuficientes puesto que no proveen demasiada información de timbre (aunque sí de acústica), por lo que utilizan por un lado K-medias de 300 dimensiones sobre los espectrogramas log-Mel junto con K-medias de 200 dimensiones sobre las características Chroma. Por otro lado, un segundo enfoque (que es el que probamos en este trabajo), es reducir la dimensionalidad de las etiquetas (segmento de señal enmascarado) utilizando un compresor acústico conocido como EncoDec [17] que consiste en una arquitectura encoder-decoder que utiliza transformers y cuantización (quantization).

Luego esto habilita a poder usar la siguiente función de pérdida:

$$\mathcal{L}_H(f; x, M, Z) = \sum_{t \in M} \log\left(\frac{\exp(\text{sim}(T(o_t), e_c)/\tau)}{\sum_{c'=1}^{C} \exp(\text{sim}(T(o_t), e_{c'})/\tau)}\right)$$

Notar que T es la transformada lineal de los resultados de o_t, que es el output del modelo previo a la transformación lineal. Esta transformación lineal se efectúa para que o_t sea un vector de la misma dimensión que e_c y la expresión de similitud (similitud coseno) pueda ser computada. Por su parte e_c es nuestra pseudo etiqueta (el resultado de aplicar reducción de dimensionalidad sobre nuestra señal). De este modo, se busca obtener las características (estados ocultos) que alcancen la máxima similitud posible entre la transformación de nuestras salidas (outputs) y nuestras pseudo etiquetas (cada una de las M pseudo etiquetas). Nos queda mencionar que τ es el parámetro de temperatura para escalar las logits.

Finalmente, una vez entrenado el modelo, se utilizan los estados ocultos que se derivan de los insumos del modelo entrenado. Estas características resultantes son un apilado (stack) de 25 capas (layers) de dimensión 1024 que se generan cada 400 muestras. Es decir, dado que la tasa de muestreo es de 24000 muestras por segundo, se generan 60 matrices de 25x1024 por segundo de sonido.

3.2.4.2. Preprocesamiento de etiquetas

Al igual que en el caso del reconocimiento vía SVM, que ya fue explicado con más detalle en la sección 3.2.1.2, las etiquetas fueron binarizadas y se quitaron aquellas que compartían parte de la señal.

3.2.4.3. Preprocesamiento de la señal

Se partió de la señal cruda, es decir que no se utilizó ninguna transformación particular, más allá de las transformaciones implementadas por el modelo de HuggingFace-MAP-MERT[19]. La única excepción a mencionar, es el remuestreo de la señal original para ajustarse al correspondiente modelo de MERT.

Como mencionamos previamente, en el caso de K-medias, el modelo utiliza log-MEL y Chroma para generar las características de entrada al cluster. En el caso del modelo de EncoDec [17], menciona que la entrada es la transformada de Fourier en el tiempo con parte real e imaginaria.

3.2.4.4. Partición del conjunto de datos

La partición en desarrollo y heldout fue la misma que la realizada en el caso anterior. El conjunto de datos de desarrollo fue partido en entrenamiento (80 %) y validación (20 %) del mismo modo.

Con el conjunto de entrenamiento y validación se hicieron varias pruebas de manera manual, variando la cantidad de epochs y los algoritmos de descenso de gradiente, junto con ligeros detalles en la arquitectura, como la capa dropout.

En cuanto al tipo de señal utilizada, se utilizaron las pistas aisladas de batería para este modelo, tal como se señala en la sección 4.1.

3.2.4.5. Detalles del clasificador

Al igual que en los modelos de SVM, se entrenaron varios clasificadores binarios en lugar de uno multiclase; pero en este caso añadimos una pequeña red convolucional (CNN) y usamos las características generadas por MERT en lugar de la transformada de fourier en el tiempo tal como se había realizado en los modelos de CNN anteriores.

No utilizamos un algoritmo de búsqueda de hiperparámetros debido a la capacidad computacional disponible, sino que exploramos diferentes posibilidades manualmente, tales como cambiar la cantidad de epochs, el batch-size, el tipo de escalamiento de las features (incluso hacerlo sin escalar, puesto que el pipeline del modelo hace los preprocesamientos necesarios).

Inicialmente planteamos la posibilidad de agregar, en lugar de una pequeña CNN, directamente una capa densa que actúe como cabeza (o head) de la red. No llegamos a explorar estos resultados puesto que el modelo tuvo dificultades para converger incluso al set de datos de entrenamiento. Posiblemente con una mayor cantidad de épocas alcance convergencia. Como consecuencia, se probó una pequeña CNN sobre las features de MERT.

Queda para trabajos futuros, explorar distintos tipos de modelos downstream, es decir plantear diferentes tipos de arquitecturas y verificar sus resultados. Entre ellas, una combinación de capas densas y pooling pero sin utilizar una CNN, plantear múltiples capas densas con alguna función de activación entre otros tipos de arquitecturas. Entendemos que las capas convolucionales permiten generar distintos espacios de características, una labor similar a la que cumple usar el modelo de MERT, por lo que es razonable pensar que la arquitectura planteada podría prescindir de estas.

La arquitectura finalmente presentada se muestra en la figura 10.

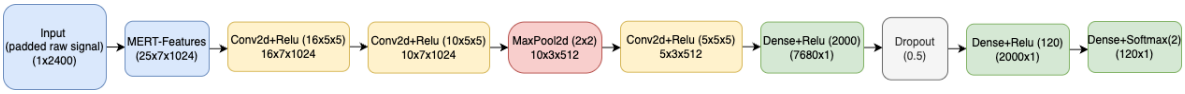

Figura 10: Arquitectura de la red tipo CNN con features basadas en MERT. Esta fue utilizada para la tarea de reconocimiento del instrumento. En la segunda línea se encuentran entre paréntesis las dimensiones, mientras que en la primera, los parámetros del modelo.

3.2.5. Reconocimiento vía clasificador zero-shot con un enfoque basado en lenguaje natural.

3.2.5.1. Introducción teórica

CLAP [8] es un método inspirado en CLIP [18] que busca conectar entradas de texto y audio mediante dos diferentes encoders y una medida de similitud.

Para dar una ilustración más precisa, supongamos que tenemos un lote (batch) de N entradas de texto y audio. Para cada entrada de texto X_t existe una entrada de audio correspondiente X_a. Las

[19] https://huggingface.co/m-a-p/MERT-v1-330M

entradas de audio X_a se encuentran en forma de espectrograma con dimensión FxT; siendo F el número de bins del espectrograma y T ventanas de tiempo del mismo.

Con el objetivo de generar una representación semántica y vectorial del texto aplicaremos un encoder de texto $f_t(.)$, transformando a los N strings X_t en vectores U dimensionales de modo que tenemos:

$$\hat{X_t} = f_t(X_t) \quad donde \quad \hat{X_t} \in \mathbb{R}^{NxU}$$

De la misma manera, aplicaremos un encoder distinto para el audio llamado $f_a(.)$, transformando a cada uno de los N espectrogramas de dimensión FxT en dimensión V. En consecuencia, tendremos N vectores V dimensionales, resultando su concatenación en una matriz $\hat{X_a} \in \mathbb{R}^{NxV}$, es decir:

$$\hat{X_a} = f_a(X_a) \quad donde \quad \hat{X_a} \in \mathbb{R}^{NxV}$$

Tras aplicar ambos encoders tenemos tanto texto como audio en representaciones vectoriales. Recordemos que el objetivo del aprendizaje por contraste supervisado (o supervised contrastive learning) es que queremos que para un par texto y audio, estos dos sean tan similares como sea posible, y contrasten cuanto más sea posible del resto de los pares texto-audio del lote. Pero para poder comparar los pares texto-audio requerimos que estén en la misma dimensión, por lo cual aplicaremos una transformación lineal $L_a(.)$ sobre la matriz de audio y una sobre el texto $L_t(.)$ de modo que llevemos ambas matrices a dimensión $N \times D$ (misma dimensión).

$$E_a = L_a(\hat{X_a}) = L_a(f_t(X_a)) \quad donde \quad E_a \in \mathbb{R}^{Nxd}$$

$$E_t = L_t(\hat{X_t}) = L_t(f_t(X_t)) \quad donde \quad E_t \in \mathbb{R}^{Nxd}$$

Realizadas nuestras transformaciones de audio y texto al mismo espacio vectorial, podemos compararlos mediante el producto punto, que representa una de las formas de evaluar la similitud entre dos vectores (pensar la matriz como un array de vectores):

$$C = \tau(E_a \cdot E_t^T)$$

Donde τ es el parámetro de temperatura para escalar las logits; y $C \in \mathbb{R}^{NxN}$ es una matriz cuadrada pero no simétrica que contiene las similitudes entre las combinaciones encodeadas texto-audio. Notar que para el elemento $C_{i,j}$ si $i = j$ veremos la similitud entre un par audio-texto correcto mientras que si $i \neq j$ estamos en frente de un par incorrecto. Lo llamamos par correcto puesto que en nuestro conjunto supervisado hemos ingresado que ese audio se correspondía con ese texto (ej un prompt que sea "A sound of a Hi-Hat" cuando el sonido sobre el cual estamos evaluando la similitud es el de un Hi-Hat). Por el contrario, llamamos par incorrecto cuando estamos cruzando un audio que no se corresponde con su texto en el conjunto supervisado (ej un prompt que sea "A sound of a Hi Hat" cuando el sonido sobre el cual estamos evaluando la similitud es el de un bombo).

La función de pérdida adoptada es el promedio de la entropía cruzada para el audio y el texto respectivamente:

$$L = \frac{1}{2}(l_{text}(C) + l_{audio}(C))$$

Donde $l_{text}(C)$ representa la entropía cruzada del texto:

$$l_{text}(C) = \frac{1}{N}\sum_{i=0}^{N} log(diag(softmax_{axis=i}(C))$$

Por otro lado $l_{audio}(C)$ representa la entropía cruzada del audio:

$$l_{audio}(C) = \frac{1}{N}\sum_{i=0}^{N} log(diag(softmax_{axis=j}(C))$$

Cabe destacar que $l_{text}(C)$ y $l_{audio}(C)$ se diferencian solamente en que la función softmax está aplicada sobre las filas en el primer caso y en el segundo caso está aplicada sobre las columnas. Por último, para hacer la inferencia de manera zero-shot, simplemente debemos insertar una lista de A audios y P prompts, estos serán encodeados y proyectados linealmente en misma dimensión, de igual manera que antes mediante E_a y E_t respectivamente. Luego se efectuará el cálculo de la matriz de similitud

C usando la ecuación $C = \tau(E_a \cdot E_t^T)$ quedando por ejemplo la similitud del prompt i=1 con el audio j=1 para el elemento $C_{1,1}$, la del prompt i=2 con el audio j = 1 para el elemento $C_{1,2}$ de la matriz. Finalmente, dado que en nuestro caso buscamos hacer inferencia multitarget, utilizamos una función sigmoidea o directamente una cota de similitud; para los elementos que superen la cota, decimos que el prompt está presente en ese audio. Contaremos detalles de como se genera en la cota en las siguientes secciones.

3.2.5.2. Preprocesamiento de etiquetas

Debemos separar el procesamiento de etiquetas en dos casos. El primer caso consiste en el clasificador de tipo zero-shot, donde no hubo un procesamiento de etiquetas particular ya que al utilizarse CLAP de manera zero-shot no se reentrenó el modelo. Solamente se contempló un conjunto de validación para reacomodar el punto de corte o "threshold", pero utilizando las etiquetas de manera tradicional.

El segundo caso es un poco más complejo, ya que en esta etapa hicimos fine-tuning del modelo, por lo que requerimos procesar las etiquetas de entrenamiento de manera tal que se acomoden al formato de CLAP. Para este segundo caso, buscamos un enfoque similar al de la CNN de tipo vectorial (sección 3.2.3), con la diferencia de que debíamos mapear la ocurrencia de esos sonidos a un prompt. Explicaremos los dos pasos a continuación en mayor detalle.

Como primer paso para el procesamiento de las etiquetas, se tomó a cada uno de los onsets ocurrentes de las anotaciones de verdad de campo. Para cada uno de ellos, se tomó una ventana de ± 50 milisegundos, y aquellos onsets que estaban dentro de la ventana se consideraban como presentes en la señal. Puesto de otra manera, si en las anotaciones teníamos que en el segundo 0.51 existía un bombo, tomábamos la señal perteneciente al intervalo [0.51-0.05 ; 0.51+0.05) y filtrábamos los instrumentos presentes en dicho intervalo de acuerdo con las anotaciones. Esto resulta en el cuadro 9 que se presenta a modo de ejemplo.

	Instrumento			
Onset time	KD	SD	HH	CY
0.020	1	0	1	0
0.290	1	0	1	0
0.296	1	0	1	0
0.570	0	1	1	0

Cuadro 9: Ejemplo del procesamiento de las etiquetas para el modelo CLAP bajo el esquema de fine-tuning. Para la primera fila, el onset que ocurre en el segundo 0.02 en realidad representa la señal ocurrente en el intervalo [0.02-0.05, 0.02+0.05) y los instrumentos que ocurren en dicho intervalo son el bombo (KD) y el HiHat (HH); esto se representa ya que tienen un uno en dicha fila; si no ocurren tendrán un 0.

Como segundo paso, se debía mapear estos instrumentos a un sólo prompt, por lo que se mapearon los instrumentos ocurrentes (aquellos que tienen el número 1 en el onset) en un sólo prompt. Por ejemplo si sólo ocurría el bombo (Kick Drum o KD) este se mapeaba en "A sound of a Kick Drum"; si ocurría el bombo y el Hi-Hat, el mapeo sería "A sound of a Kick Drum and a Hi-Hat"; si además ocurría el redoblante (Snare Drum o SD) quedaba como "A sound of a Kick Drum, a Hi-Hat and a Snare Drum". Ejemplos de estos mapeos pueden verse en el cuadro 9.

	Instrumento				
Onset time	KD	SD	HH	CY	Prompt resultante
0.020	1	0	1	0	A sound of a Kick Drum and a Hi Hat
0.290	1	0	1	0	A sound of a Kick Drum and a Hi Hat
0.296	1	0	1	0	A sound of a Kick Drum and a Hi Hat
0.570	0	1	1	0	A sound of a Snare Drum and Hi Hat

*Cuadro 10: Ejemplo del procesamiento de las etiquetas para el modelo CLAP bajo el esquema de fine-tuning. Para la primera fila, el onset que ocurre en el segundo 0.02 en realidad representa la señal ocurrente en el intervalo [0.02-0.05, 0.02+0.05) y los instrumentos que ocurren en dicho intervalo son el bombo (KD) y el HiHat (HH). Esto se representa ya que tienen un 1 en dicha fila; si no ocurren tendrán un 0. Para convertirlos en prompts, solamente agregamos el prefijo "A sound of a **Tipo de instrumento**", si ocurre un segundo instrumento le agregamos la frase "and a **Tipo de instrumento**", si ocurre un tercer instrumento lo separamos por comas excepto al último que tendrá la preposición en inglés "**and**".*

Por último es importante mencionar que tanto en el enfoque de CNN vectorial como en el de CLAP no se han quitado las observaciones cercanas, por lo que si bien no existe el riesgo de que sean mutuamente excluyentes como en el caso de los modelos de CNN binarios, sí está la posibilidad de repetir el dato si el evento ocurre en una ventana lo suficientemente similar. Este es el caso del segundo y tercer evento del cuadro 9 donde el evento ocurre en momentos muy similares por lo que la señal tomada será muy similar y el prompt resultante entre los eventos será el mismo. Quedará para trabajos futuros explorar esta variante.

3.2.5.3. Preprocesamiento de la señal

Se partió de la señal cruda, es decir que no se utilizó ninguna transformación particular, más allá de las transformaciones implementadas por el modelo de HuggingFace-CLAP[20]. El pipeline de HuggingFace CLAP utiliza representaciones del audio en formato log Mel con tasa de muestreo de 44100 hertz (nuestros audios de 22050 hertz son remuestreados), movimientos de la ventana de 320 secuencias (hop size), tamaño de la ventana de 1024 secuencias (window size) y 64 mel bins en el rango de 50-8000 hertz. El encoder que se utiliza es CNN14 [19] para el audio y BERT uncased [15] para el texto. A su vez los segmentos de audio de más de 5 segundos fueron recortados y los de menor duración que esto fueron rellenados con ceros (padding).

Queda pendiente para trabajos futuros, utilizar un modelo que posea la misma sampling rate que nuestro audio original. Posiblemente otra alternativa sea utilizar mejores algoritmos de remuestreo de la señal.

3.2.5.4. Partición del conjunto de datos

La partición en desarrollo y heldout fue la misma que la realizada en el caso anterior. Para el primer caso, donde no hubo fine-tuning, se utilizó todo el conjunto de desarrollo para el ajuste del punto de corte o threshold. Para el segundo caso, donde sí hubo fine-tuning, el conjunto de datos de desarrollo fue partido en entrenamiento (80 %) y validación (20 %). Utilizando el conjunto de entrenamiento se realizó el fine-tuning, mientras que el conjunto de validación se usó para el ajuste del punto de corte.

En cuanto al tipo de señal utilizada, se utilizaron las pistas aisladas de batería para este modelo, tal como se señala en la sección 4.1.

3.2.5.5. Detalles del modelo

Como hemos mencionado anteriormente, se utilizó un modelo CLAP, donde se instruían prompts del tipo "A sound of a *tipo de instrumento*". Este prompt era pasado por el encoder de texto, junto con la señal que era pasada por el encoder de sonido. Tras encodear ambas partes, conseguimos dos vectores de la misma dimensionalidad, que se cruzaban mediante un producto punto que servía como medida de similitud. Cuanto mayor era este producto, mayor la similitud alcanzada.

Si ocurría que el producto punto del prompt encodeado y el sonido encodeado superaban un punto de corte, se predecía que ese sonido estaba presente en el onset. El punto de corte fue determinado

[20] https://huggingface.co/docs/transformers/en/model_doc/clap

mediante el planteo de diferentes cuantiles de similitud para cada instrumento, sobre el conjunto de desarrollo o validación.

Se realizó esto para cada tipo de instrumento, es decir que se encodeaba cada uno de los instrumentos en un prompt diferente y se lo cruzaba mediante producto punto con el sonido encodeado. En esta instancia es importante destacar que no podríamos usar softmax o algo similar como se suele utilizar en este estilo de algoritmos, puesto que estamos en presencia de una predicción multi-etiqueta y no multiclase; es decir que la ocurrencia de los instrumentos no es mutuamente excluyente (puede ocurrir más de uno en el mismo onset).

3.3. Algoritmos de detección de onsets

A continuación, presentaremos algunas de las metodologías aplicadas para los diferentes algoritmos de detección de eventos. Solemos dividir estas metodologías, para mayor modularidad y simplicidad, en diferentes partes tales como: Breve introducción teórica, procesamiento de etiquetas, procesamiento de la señal, partición del conjunto de datos y detalles del clasificador.

3.3.1. Detección mediante redes neuronales recurrentes simples

3.3.1.1. Introducción teórica

Una red neuronal recurrente (RNN) es cualquier red que contiene un ciclo dentro de sus conexiones, lo que significa que el valor de alguna unidad es directa o indirectamente dependiente de sus propios resultados anteriores como insumo [20]. En este apartado consideraremos un subtipo de redes neuronales recurrentes conocidas como redes de Elman o redes simples [21]. Una red de Elman es una red que se caracteriza por las siguientes ecuaciones:

$$h_t = \sigma_h(W_h x_t + U_h h_{t-1} + b_h)$$

$$y_t = \sigma_y(W_y h_t + b_y)$$

Donde:

- x_t es el vector de entrada de la red.
- h_t es el vector de estado oculto (o hidden state).
- y_t es el vector de resultados (output).
- σ_h y σ_y son funciones de activación.
- W_h es la matriz de pesos que proyecta linealmente el vector de entrada de la red (información presente).
- U_h es la matriz de pesos que proyecta linealmente el vector de estado oculto anterior (en el momento *t-1*), de modo que este concentra la información que se ha retenido en la red en estados pasados tanto así como presentes (notar que h_t es una función que depende de h_{t-1} (información previa) y x_t (información presente) .
- W_y es la matriz de pesos que proyecta linealmente el vector de estado oculto presente (en el momento t).
- b_h y b_y son los vectores de pesos que acompañan a h_t y y_t respectivamente.

Puesto en otros términos, el resultado de la red y_t es la proyección lineal de h_t. Pero h_t no solo depende de la información presente x_t sino que depende del estado oculto en su momento anterior, h_{t-1}. Esto es lo que da la estructura autorregresiva de la red; y permite retener información temporal aunque con ciertas dificultades para retener la de largo plazo, sobre todo cuando se trata de secuencias largas.

Hasta aquí hemos mencionado las redes sencillas que parten de la información presente a la futura, pero a veces tenemos acceso a toda la secuencia y también podemos utilizar la red de izquierda a derecha, dando vuelta el orden de la información.

De este modo tendremos dos redes, una con el orden tradicional de la secuencia (izquierda a derecha) y otra con el orden invertido (derecha a izquierda), que llamaremos $h_t^f = RNN_{forward}(x_1, ..., x_t)$ y $h_t^b = RNN_{backward}(x_n, ..., x_t)$ respectivamente.

Luego, para tener una red bidirecciónal recurrente concatenamos el resultado de ambas redes, de modo que generamos nuestro vector de estados ocultos h_t. En algunos casos en lugar de concatenación hay adición o multiplicación elemento a elemento (en nuestro caso usamos concatenación). De este modo, para generar el vector de secuencias y_t utilizamos no solo la información pasada sino la futura en cada paso o step t de la inferencia. Es decir que quedaría $y_t = \sigma_y(W_y h_t + b_y)$ pero ahora h_t tiene el doble de dimensión puesto que es la concatenación de h_t^f y h_t^b.

El último paso que nos queda por mencionar, es que estas redes bidireccionales recurrentes se pueden apilar, convirtiéndose así en las conocidas *RNN apiladas* o (stacked RNN). Las RNN apiladas constan de múltiples redes donde la salida de una capa sirve como la entrada a una capa posterior. Las RNN apiladas generalmente superan a las redes de una sola capa. Una razón para este éxito parece ser que la red induce representaciones en diferentes niveles de abstracción entre capas. Así como las primeras etapas del sistema visual humano detectan bordes que luego se utilizan para encontrar regiones y formas más grandes, las capas iniciales de las redes apiladas pueden inducir representaciones que sirven como abstracciones útiles para capas adicionales; representaciones que podrían resultar difíciles de inducir en una sola RNN [20]. El número óptimo de RNN apilados es específico de cada aplicación y de cada conjunto de entrenamiento. Sin embargo, a medida que aumenta el número de pilas, aumentan los costes computacionales rápidamente.

3.3.1.2. Procesamiento de las etiquetas

Debemos recordar que el objetivo de este clasificador es identificar donde ocurre un evento de batería. Para ello, tomamos por ejemplo intervalos de 2048 muestras (en realidad se toman 3 ventanas con diferente longitud) en una ventana móvil de 10 milisegundos. Es decir, generamos 100 ventanas por segundo, en cada una de estas ventanas (o frames) evaluamos si al menos un evento de nuestras anotaciones de verdad de campo cae dentro de dicha ventana. En caso afirmativo, decimos que allí hubo un onset de batería y asignamos el valor 1. Por el contrario, en caso negativo, decimos que allí no hubo un onset de batería y asignamos el valor 0.

Expandiéndolo al caso de las 3 ventanas, si la ventana más grande (o equivalentemente si al menos una de las ventanas) contiene a una anotación de verdad de campo, decimos que el onset está presente y se asigna el valor 1; caso contrario decimos que el onset no está presente y se asigna el valor 0.

De esta manera, tenemos un conjunto de datos con nuestras llamadas características o $\mathbf{X}$ y nuestras llamadas etiquetas o $\mathbf{y}$.

Onset time	Onset
0.020	1
0.290	1
0.296	1
0.570	0

Cuadro 11: Ejemplo del procesamiento de las etiquetas para el modelo de detección de eventos.

3.3.1.3. Procesamiento de la señal

Para el procesamiento de la señal,[21] se remuestreó la señal a 44100 muestras por segundo y se utilizaron 3 espectrogramas apilados (stacked). El primer espectrograma se tomó de 1024 muestras para la STFT, un filtro de 3 bandas, se lo escaló logarítmicamente y luego se tomaron las diferencias positivas respecto a su frame anterior. Para el segundo y el tercero, se tomaron respectivamente 2048 y 4096 muestras, 6 y 12 bandas. El postprocesamiento (escalamiento logarítmico y diferencias positivas) para ambos fue el mismo que el del primer espectrograma.

3.3.1.4. Partición del conjunto de datos

[21] Este modelo, es una versión entrenada y adaptada con nuestros datos de ISMIR 2018 RNN Onset Detection Tutorial.

Para la partición del conjunto de datos, dentro del dataset de desarrollo, se tomaron las siguientes pistas como conjunto de validación:

```
MusicDelta_Britpop_Drum
MusicDelta_Rock_Drum
MusicDelta_FunkJazz_Drum
```

A diferencia del caso planteado en ISMIR 2018, no se utilizó validación cruzada, sino que simplemente usamos eventos de las pistas mencionadas anteriormente.

Las pistas restantes del conjunto de desarrollo fueron utilizadas para el entrenamiento del modelo. Por otro lado, las pistas que se consideran fuera de muestra, o set de testeo, fueron las mismas que en el resto de los modelos.

En cuanto al tipo de señal utilizada, esta varía experimento a experimento. Cuando evaluamos solamente los resultados del detector de onsets (sección 4.2) utilizamos las pistas de batería aisladas. Cuando buscamos evaluar la efectividad de los algoritmos de separación de fuentes, utilizamos en un primer caso el detector de onsets con las pistas con todos los instrumentos y lo comparamos con un segundo caso que fue aplicar primero separación de fuentes sobre las pistas con todos los instrumentos y luego entrenar el modelo de onset con la predicción generada por el modelo de separación de fuentes (nos quedamos sólo con la batería) tal como será descrito en los respectivos experimentos (secciones 4.6 y 4.7).

3.3.1.5. Detalles del clasificador

Se entrenó un clasificador binario que consiste en un stack de tres RNN bidireccionales simples o de Elman [21], seguida de una capa densa con una función de activación sigmoidea como head o cabeza. Esto puede verse en la figura 11 a continuación.

Para el entrenamiento se plantearon 100 épocas con entropía cruzada (cross entropy) como función de pérdida. Se utilizó descenso de gradiente estocástico con tasa de aprendizaje (learning rate) de 0.01 momentum de 0.9 y recorte de gradiente (gradient clipping), para prevenir el problema de desvanecimiento de gradiente (vanishing gradient problem). A su vez, se plantearon 20 épocas de early stopping con una mejora mínima de 1e-4, la función a monitorear fue la entropía cruzada sobre el conjunto de validación.

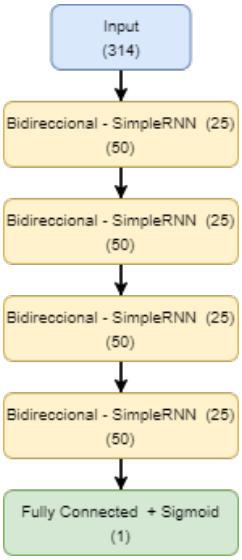

Figura 11: Arquitectura de la red tipo recurrente utilizada para la detección de eventos. La primera línea posee los parámetros del modelo, mientras que la segunda las dimensiones de la red.

3.3.2. Detección mediante redes convolucionales

En esta instancia utilizamos las redes convolucionales presentadas anteriormente para la detección de eventos u onsets.

Algunos lineamientos sobre este tipo de arquitecturas ya fueron explicados anteriormente. Para una breve introducción teórica referirse a la sección 3.2.2.1.

El procesamiento de las etiquetas, de la señal y la partición del conjunto de datos aplicada son los mismos que para la red recurrente (sección 3.3.1). Cabe destacar que el procesamiento de la señal basado en diferencias podría ser superado por uno tradicional (STFT sin diferencias) como aplicamos en los algoritmos de reconocimiento basados en CNN. Queda esta aplicación para trabajos futuros.

Podría pensarse que las CNN pueden identificar mejor espectrogramas que las RNN, pero las últimas brindan una estructura autorregresiva, clave en la detección de onsets. Esto motiva a ver si las ventajas de procesamiento de features de la CNN a la hora de recibir un espectrograma superan a las ventajas de la estructura autorregresiva brindada por las RNN, lo cual podremos observar en la sección de resultados.

3.3.3. Detección mediante algoritmos no entrenables

No necesariamente debemos entrenar un modelo de aprendizaje automático para la detección de eventos, sino que también este problema puede resolverse con un simple algoritmo. En nuestro caso probamos un algoritmo conocido como "superflux"[22], para el cual utilizaremos el mismo procesamiento de la señal que en la red recurrente (sección 3.3.1) y por otro lado, utilizamos el algoritmo provisto en la librería librosa. El algoritmo superflux [10] consiste en:

- 1) Tomar el espectrograma de la señal (en nuestro caso con tamaño de la ventana de 2048 muestras y 220 es el hopSize o movimiento de la ventana, implicando que tomamos 100 frames por segundo).
- 2) Sobre dicho espectrograma, tomar un filtro de doce bandas por octava, con frecuencia mínima y máxima de 30 y 17000 hertz respectivamente.
- 3) Pasarlo a escala logarítmica: Sumar 1 sobre el paso (2) y tomar el logaritmo base 10 del mismo.
- 4) Tomar las diferencias positivas del paso (3). Es decir aquí tendremos todos los incrementos de energía (cuando sube el nivel de sonido) a lo largo de las frecuencias.
- 5) Sumar los incrementos obtenidos de (4) a lo largo de las frecuencias. Eso nos indica un posible candidato.
- 6) Utilizar un algoritmo de peak-picking al igual que en [22], para filtrar solo los algoritmos que superen cierta cota del item (5) y aquellos que se consideren repetidos (en nuestro caso utilizamos el de madmom).

3.4. Algoritmo de separación de fuentes

La idea subyacente detrás de utilizar un algoritmo de separación de fuentes, fue verificar si los clasificadores anteriormente mencionados eran robustos a introducir el resto de las pistas o bien si requerían (o se beneficiaban) de un algoritmo de separación de fuentes como parte del preprocesamiento.

En esta instancia debemos recordar que siempre se probaron los distintos clasificadores bajo la pista de batería cruda, es decir, sin los otros tipos de instrumentos presentes tales como la guitarra, el bajo o la voz. Como nuestro conjunto de datos también posee pistas completas (con todos los instrumentos tales como voz, bajo, guitarra y batería), vamos a evaluar la capacidad de predicción de un clasificador usando estas y compararlo con la aplicación de un algoritmo de separación de fuentes previo, para generar la señal cruda.

En una primera instancia, evaluaremos los algoritmos de reconocimiento con versus sin separación de fuentes, utilizando las etiquetas de verdad de campo, para que nos diga en qué momento ocurre el onset y predeciremos sólamente cuál de éstos ocurre. En una segunda instancia, evaluaremos los algoritmos de reconocimiento en conjunto con los de detección de onsets, de modo que a diferencia del caso anterior, no solo diremos qué tipo de instrumento ocurre sino cuando. En una tercera instancia, evaluaremos solamente los algoritmos de detección de onsets.

Para estas pruebas, utilizamos el algoritmo de separación de fuentes conocido como *Hybrid-Demucs* [11].

[22] Las pruebas de este primero han sido inspiradas en el github de la conferencia ISMIR 2018. Link: https://github.com/slychief/ismir2018_tutorial/blob/master/Part_3a_Onset_Detection.ipynb

3.5. Medición

Al igual que en otros trabajos como [22, 2, 7, 4, 5, 6, 3] evaluaremos los resultados de los algoritmos de detección de eventos y reconocimiento del instrumento utilizando las siguientes métricas: Precision, Recall y F1-Score.

Precision: Se define como el total de las predicciones correctamente realizadas como proporción de las predicciones realizadas. Es decir $Precision = \frac{N_c}{N_p}$, donde N_c es la cantidad de predicciones correctamente realizadas y N_p es la cantidad de predicciones realizadas para ese tipo de instrumento.

Recall: Se define como el total de predicciones correctamente realizadas como proporción de las anotaciones del onset. Es decir $Recall = \frac{N_c}{N_g}$, donde N_c es la cantidad de predicciones correctamente realizadas y N_g es la cantidad de onsets anotados (ground truth) para ese tipo de instrumento.

F1-Score: Se define como la media armónica entre el precision (P) y el recall (R). Es decir, $F_1 = \frac{2}{1/R+1/P}$.

En algunos trabajos como [5] se menciona que una predicción debe ser calificada como correcta si la diferencia entre la predicción realizada y la observada es menor a 30 ms. En nuestro caso, dado que estamos tratando de algoritmos de reconocimiento del instrumento meramente, esto es trivial puesto que el momento en el que ocurre el onset está brindado por el groundtruth y no por un algoritmo que cumpla la función de detectar los eventos. Es importante aclarar, que esta consideración no resulta trivial cuando testeamos si los resultados del modelo de reconocimiento del instrumento se ven afectados (sección de resultados, 4.4) por el hecho de que a este se le provea o no las etiquetas de groundtruth (en qué momento ocurre el sonido). Para este caso, adoptamos como norma la diferencia de 30 ms mencionada anteriormente. Tal como se menciona en [22], hay que ser cautelosos a la hora de contar los verdaderos positivos en el momento de la medición. En este trabajo se toma la medición que los autores presentan como "no estricta" o "merged", pudiendo llevar levemente a mejores resultados[23].

Por otro lado, para evaluar la calidad del modelo de separación de fuentes existen diferentes métricas o formas de medición [23, 24]. Utilizaremos tanto escucha manual así como también la métrica conocida como **Scale-Invariant Signal to Distortion Ratio** (SI-SDR). Esta métrica computa el ratio entre la energía correspondiente a la señal original $|s|^2$ escalada por una proyección ortogonal al residuo α y la energía correspondiente a la diferencia entre la señal original proyectada y la predicha $|\hat{s}|^2$ (también conocida como energía de la distorsión escalada o residuo escalado), expresado en decibeles. Esto es:

$$SISDR = 10log_{10}\frac{|\alpha s|^2}{|\alpha s - \hat{s}|^2}$$

Donde:

- s es la señal original.
- $\hat{s}$ es la señal estimada por el algoritmo de separación de fuentes.
- $s - \hat{s}$ representaría el residuo o distorsión.
- α representa el coeficiente de escalamiento de manera tal que la señal original y dicho residuo sean ortogonales. Más precisamente: $\alpha = s^T s/|s|^2$.

Por ende el numerador $|\alpha s|^2$ representa la energía (o matemáticamente, la norma cuadrada) de la señal escalada y el denominador $|\alpha s - \hat{s}|^2$ la energía correspondiente a la distorsión escalada. Otra consideración es que multiplicar por 10 y transformar logísticamente este coeficiente, se lo usa a efectos de expresar la métrica en decibeles.

3.6. Software y hardware utilizado

Con respecto al software, se utilizó puramente el lenguaje de programación Python. En particular, se usaron principalmente las siguientes librerías:

- **Librosa** [25] y **madmom** [26], para el preprocesamiento de la señal y etiquetas.

[23] Cuando realizamos la comparación de metodologías, utilizando el modulo de medición de la librería madmom versus nuestra implementación, las diferencias no resultaron considerables.

- **Sklearn** [27], como framework general orientado a aprendizaje automático.
- **Torch** [28] y **Keras (backend tensorflow)** [29], como frameworks generales de aprendizaje profundo.
- **Transformers** [30] de HuggingFace como framework general para el uso de algunos modelos como MERT y CLAP.
- Finalmente, **Pandas** [31] y **numpy** [32], para realizar la mayor parte del preprocesamiento de los datos.

En lo que respecta al hardware, se desarrolló enteramente utilizando el espacio de trabajo de Google Colab.

4. Resultados

Como se adelantó anteriormente, los resultados se prueban sobre el conjunto de datos MDB Drums, utilizando la partición de out of sample mencionada en el apartado donde presentamos dicho conjunto de datos (sección 2).

4.1. Resultados de los modelos de reconocimiento del instrumento

Los resultados de los diferentes algoritmos de reconocimiento del instrumento, utilizando las pistas de batería aisladas, se encuentran en el cuadro 12, que presentamos a continuación:

	Comparación resultados de reconocimiento del instrumento.				
Método	Tipo de instrumento	Recall	Precision	F1-Score	Macro F1-Score
SVM	CY	0.371	0.608	0.461	0.767
	HH	0.886	0.703	0.784	
	KD	0.95	0.978	0.964	
	SD	0.946	0.788	0.86	
CNN Vector	CY	0.34	0.72	0.46	0.778
	HH	0.84	0.71	0.77	
	KD	0.96	0.97	0.97	
	SD	0.94	0.87	0.91	
CNN Binario	CY	0.533	0.758	0.626	**0.838**
	HH	0.792	0.825	0.808	
	KD	0.977	0.973	0.975	
	SD	0.953	0.932	**0.942**	
MERT	CY	0.416	0.73	0.53	0.834
	HH	0.915	0.882	**0.898**	
	KD	0.99	0.981	**0.985**	
	SD	0.937	0.907	0.922	
CLAP Zero-Shot	CY	0.995	0.316	0.48	0.707
	HH	0.824	0.637	0.719	
	KD	0.952	0.751	0.84	
	SD	0.916	0.694	0.789	
CLAP FineTuned	CY	0.761	0.579	**0.658**	0.773
	HH	0.787	0.642	0.707	
	KD	0.956	0.841	0.895	
	SD	0.967	0.734	0.835	

Cuadro 12: Resultados de los modelos de reconocimiento del instrumento utilizando las pistas de batería aisladas, para el conjunto de fuera de muestra (out of sample).

Se observan resultados superiores al 0.7 en F1-Score para el bombo (KD), el redoblante (SD) y el Hi-Hat (HH). Éstos, comparten en general la característica de que su duración no es prolongada y

tienen alto decay; exceptuando, cuando el Hi-Hat se toca abierto o semi abierto, donde el sonido es más prolongado. Quizás esa sea una de las razones por las cuales el rendimiento baja respecto a los otros dos instrumentos.

Tal como se señala en [7], es esperable que el mejor rendimiento sea exhibido por el KD, puesto que es el instrumento con menor variabilidad técnica de todos. Puesto en otros términos, el KD es un instrumento que siempre suena parecido, independientemente de la técnica utilizada[24].

Por otro lado, los resultados disminuyen considerablemente para el instrumento CY (platillo). Este instrumento presenta un menor decay y una menor cantidad de datos disponibles, lo que podría explicar la caída en el rendimiento. Además, y en menor medida que el Hi-Hat, los sonidos de los diferentes platillos pueden variar según factores como su tamaño o la zona de golpeo (ej: campana u otra parte del plato).

Tras analizar los resultados para OT (otros elementos de percusión) y TT (toms), se observa que estos fueron nulos o casi nulos. Esto se debe, en parte, a que nuestro conjunto de datos contiene solo unas 90 observaciones para TT y unas 80 para OT. En el futuro, deberíamos considerar la utilización de datos sintéticos o provenientes de otros conjuntos de datos para entrenar estos modelos de manera más efectiva. Notar que no incluimos dichos resultados en pos de lograr una mayor claridad en la presentación de las tablas.

Si comparamos modelo a modelo en lugar de resultados generales, puede notarse que el modelo de CNN-binario[25] (sección 3.2.2) es aquel que lidera el puesto en Macro F1-Score (promedio simple de los F1-Scores para cada instrumento) con 0.838, seguido de MERT con 0.834. Considerando el tamaño de la muestra de heldout, podemos pensar que estos modelos son casi iguales y si hubiésemos considerado otras métricas como micro/weighted F1-Score, el orden de los resultados variaría.

Situándose en último puesto se encuentra CLAP Zeroshot (sección 3.2.5) con Macro F1-Score de 0.707. Consideramos que este resultado es aceptable, dado que este modelo no requirió entrenamiento. No debe confundirse con su misma versión CLAP pero finetuneada, puesto que este último si requirió entrenamiento y consiguió un Macro F1-Score de 0.773.

Luego de CLAP Zero-shot, le sigue el modelo de SVM con un macro F1-Score de 0.767. Podemos pensar que los kernels planteados de los modelos SVM realizan un buen trabajo a la hora de transformar el espacio de características para su posterior separación. Sin embargo, este es superado por las CNN posiblemente debido a que pueden extraer características con distintos niveles de abstracción y conservar la relación espacial del espectrograma. Recordemos que a diferencia de las CNN, SVM debe hacer un flatten o aplanado de las características lo que lleva a una pérdida de información espacial.

Cuando examinamos resultados de los modelos instrumento por instrumento, notamos que MERT (sección 3.2.4) obtiene el mejor F1-Score para dos de los cuatro instrumentos evaluados en esta tabla, siendo estos el HiHat (HH) con 0.898 y el bombo con (KD). Respecto al redoblante (SD), el CNN binario es el aquél que posee un mayor F1-Score con 0.942. Por último, para el platillo (CY) el modelo CLAP Finetuned (sección 3.2.5) exhibe el mejor resultado con un F1-Score de 0.658.

Existen dos modelos a destacar que han quedado atrás en cuanto a los resultados: Por un lado el modelo CLAP Zero-Shot puesto que este ha obtenido buenos resultados (Macro F1-Score de 0.708) y a diferencia del resto no requirió entrenamiento previo más allá del ajuste del punto de corte. Por otro, el modelo CNN vectorial[26] (sección 3.2.3) que logró buenos resultados (Macro F1-Score de 0.778) requiriendo un solo modelo en memoria y con la posibilidad de realizar la inferencia para todos los instrumentos al mismo tiempo. Dicha posibilidad también existe en los modelos de CLAP puesto que requieren una lista de prompts de entrada, aunque los mismos requieren mayor memoria RAM. Queda para trabajos futuros una comparación del tiempo de inferencia de estos modelos.

Por último, debemos observar que el modelo de CNN vectorial obtuvo peores resultados que los múltiples CNN binarios, marcando una diferencia superior a 0.05 en términos de macro F1-Score. Esto posiblemente se deba a que el modelo de CNN vectorial debe compartir el espacio de características a lo largo de los cuatro instrumentos y el modelo de CNN binario puede utilizar un espacio para cada instrumento (recordemos que aquí se utiliza un modelo para cada tipo de instrumento) tal como señalamos previamente.

[24] Existen diversas técnicas para tocar el KD, como por ejemplo las que se mencionan en https://drummagazine.com/lesson-heel-up-vs-heel-down-bass-drum-foot-technique/ pero sin embargo, todas suenan similares. En cambio, cuando se trata de un redoblante por ejemplo, existen ghost notes, flams y otras técnicas que inducen a una mayor variabilidad del sonido tanto en su frecuencia como en la amplitud de su señal

[25] También lo referimos como CNN-multiclase o directamente CNN.

[26] También nos referimos ocasionalmente a este como CNN-multietiqueta.

4.2. Resultados de los modelos de detección de onsets

Los resultados de los diferentes algoritmos de detección de eventos (onsets) se encuentran en el cuadro 13, que presentamos a continuación:

Método	Recall	Precision	F1-Score
CNN Onset Detector	0.957	0.72	0.822
RNN Onset Detector	0.96	0.996	**0.978**
Superflux	0.846	0.972	0.905

Cuadro 13: Resultados de los modelos de detección de onsets sobre las pistas de batería aisladas, para el conjunto de fuera de muestra (out of sample).

Como puede observarse en este cuadro, el modelo que mejor anduvo es el basado en RNN con un F1-Score de 0.978; siguiendo por el algoritmo "superflux" con 0.907 y por último el método de CNN con 0.822. Algo clave aquí fue la decisión del punto de corte y como se realiza el levantamiento de los picos. Para el levantamiento de los picos o "peak picking" se utilizó el mismo algoritmo planteado en madmom, pero sí se varió el punto de corte para el cual decimos que algo es un onset o no.

Observando las diferencias, el modelo de RNN superó al algoritmo Superflux por más de 0.07 en F1-Score debido a que el primero es un modelo entrenable que aprende de las particularidades de nuestro conjunto de datos mientras que el segundo es un algoritmo no entrenable. Por otro lado, el modelo basado en CNN fue el último en la lista posiblemente debido al preprocesamiento aplicado: mientras que Superflux y RNN utilizan procesamientos basados en las diferencias de espectrograma positivas, CNN utiliza el espectrograma sin diferenciar, al igual que en el caso de reconocimiento del instrumento pero con una ventana de menor tamaño para generar la STFT.

4.3. Resultados de los modelos de separación de fuentes

En esta sección se presentan los resultados de la separación de fuentes. El experimento consiste en comparar: (1) la predicción de la señal brindada como consecuencia de aplicar el modelo Hybrid-Demucs sobre la pista completa y filtrar la pista de bateria con (2) la pista de batería original. Recordemos que el conjunto de datos MDB-Drums posee tanto pista de batería como la pista con todos los instrumentos juntos. Por comparación de (1) y (2), se entiende tomar "Scale Invariant Signal-To-Distortion Ratio" (SI-SDR) entre ambas señales. Se presentan tanto los resultados a nivel pista como absolutos en el cuadro 14 a continuación:

Pista	SI-SDR
MusicDelta_Beatles.wav	-2.157
MusicDelta_Country1.wav	-6.727
MusicDelta_FreeJazz.wav	-24.463
MusicDelta_Gospel.wav	-4.776
MusicDelta_Grunge.wav	-24.348
MusicDelta_Hendrix.wav	-17.193
MusicDelta_LatinJazz.wav	1.476
MusicDelta_ModalJazz.wav	4.833
MusicDelta_Punk.wav	-8.897
MusicDelta_SpeedMetal.wav	-3.583
MusicDelta_SwingJazz.wav	2.022
Promedio SI-SDR	**-7.619**

Cuadro 14: Resultados provenientes de aplicar Scale invariant Signal to Distortion Ratio (SI-SDR) por pista y también tomando el promedio de los valores por pista. La comparación se realiza entre la pista completa (todos los instrumentos) pero aplicandole separación de fuentes para luego filtrar la batería y por otro lado la pista de batería original.

Puede observarse que los resultados varían de canción a canción, siendo la de modal Jazz la que mejor resultado obtuvo (4.8) y la de FreeJazz la que peor resultado obtuvo (-24.46). El promedio de los

resultados es de -7.6. Si bien los resultados pueden parecer un tanto desalentadores,[27] es importante destacar que cuando se escucha los audios con y sin separación de fuentes de manera manual, en la mayoría de los casos se preserva la señal (aunque con una leve pérdida de calidad) y en todos los casos se preserva el ritmo de batería. Algo interesante destacar, es que encontramos que si bien el ritmo se preserva, quizás en algunos instrumentos la separación de fuentes no mantiene estabilidad en su sonido. Por ejemplo, que un platillo ride cambie su sonoridad de uno de 16 pulgadas a uno de 20 pulgadas.

También es importante destacar que, como veremos a continuación, la inclusión de este modelo de separación de fuentes conduce a mejores resultados que cuando no se incorpora.

4.4. Combinación del modelo de detección de eventos y los de reconocimiento del instrumento

En esta sección, combinamos los modelos de reconocimiento del instrumento con un modelo de detección de eventos, para comprobar si el hecho de no contar con las anotaciones de groundtruth afectan fuertemente el rendimiento de nuestros modelos de reconocimiento del instrumento. Es importante señalar que los modelos de RI por sí solos, tal como los diseñamos, no constituyen un sistema de TAB. Esto se debe a que no están diseñados para detectar el momento en el que ocurre un evento, sino únicamente para clasificar su tipo. Sin embargo, al combinarlos con un modelo de detección de eventos, sí conforman un sistema de TAB.

	RNN+CNN Binario			GroundTruth+CNN Binario			Variación absoluta		
Tipo de instrumento	F1-Score	Recall	Precision	F1-Score	Recall	Precision	F1-Score	Recall	Precision
KD	0.968	0.964	0.972	**0.975**	0.977	0.973	-0.007	-0.013	-0.001
SD	0.907	0.902	0.912	**0.942**	0.953	0.932	-0.035	-0.051	-0.020
HH	0.758	0.727	0.792	**0.808**	0.792	0.825	-0.050	-0.065	-0.033
CY	0.547	0.442	0.717	**0.626**	0.533	0.758	-0.079	-0.091	-0.041
OT	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
TT	0.000	0.000	0.000	**0.026**	0.013	0.500	-0.026	-0.013	-0.500

Cuadro 15: Resultados provenientes de la combinación del detector de eventos basado en RNN, y CNN para reconocimiento del instrumento (out of sample). A la derecha, se encuentran los resultados del reconocimiento del instrumento, pero utilizando las etiquetas de verdad de campo (groundtruth) en lugar de un modelo de detección de onsets. La comparación de F1-Scores indica que no existe mayor degradación por utilizar un modelo de detección de onsets en lugar de las etiquetas de verdad de campo.

Podemos observar en el cuadro 15 que los resultados se mantienen, aunque decrecen levemente. Esto muestra que los modelos de reconocimiento del instrumento, pueden prescindir de las anotaciones de groundtruth y, en lugar de ellos, depender de un modelo de detección de eventos.

4.5. Resultados del modelo de reconocimiento del instrumento. Comparación con y sin separación de fuentes

En esta sección, compararemos los resultados de entrenar modelos con las pistas completas (incluye batería, bajo, guitarra y voz) con y sin separación de fuentes aplicada previamente. Es decir, en un caso entrenamos el modelo de CNN con la pista completa de modo que aprenda a distinguir sonidos de batería de otros sonidos (clasificados también como 'other'). En el otro, utilizamos Hybrid-Demucs como modelo de separación de fuentes (SF), sobre la pista completa de modo de poder aislar la batería. Luego sobre la pista de batería predicha por el modelo de SF entrenaremos un mismo modelo CNN de reconocimiento del instrumento pero con esta nueva señal que intenta incluir sólo los sonidos de batería. Los resultados se muestran a continuación en la tabla 16.

[27] No obstante cuando tomamos, en lugar de SI-SDR, el SDR clásico se obtiene un promedio de 3.3 y este resulta positivo para todas las canciones de heldout exceptuando la de FreeJazz.

	Resultados de SS, combinado con reconocimiento de instrumento				
Método	Instrumento	Recall	Precision	F1-Score	Macro F1-Score
Sin separación de fuentes	CY	0.093	0.333	0.145	0.705
	HH	0.934	0.655	0.77	
	KD	0.964	0.98	**0.972**	
	SD	0.943	0.924	0.933	
Con separación de fuentes	CY	0.544	0.689	**0.608**	**0.825**
	HH	0.941	0.686	**0.793**	
	KD	0.944	0.982	0.963	
	SD	0.964	0.907	**0.935**	

Cuadro 16: Resultados provenientes del uso del método de CNN. Uno de los métodos utilizó separación de fuentes (Hybrid-Demucs) como parte del preprocesamiento de la señal para quedarse con solamente la pista de batería. El otro método utilizó directamente la pista completa con bajo, batería, guitarra y voz pero sin un método de separación de fuentes como preprocesamiento. En este caso utilizamos las etiquetas de verdad de campo y no un método de detección de onsets.

Notamos que en términos agregados, el modelo de separación de fuentes, mejora los resultados en términos F1-Score a lo largo de los instrumentos. Sobre esto, debemos destacar que la mayor diferencia se encuentra en los CY, siguiendo por HH y SD. Es importante mostrar que el modelo de separación de fuentes resultó peor en el caso del KD y muy levemente mejor en el caso del SD. Cuando no existen amplias diferencias en los resultados como lo es en el SD o KD, resulta mejor no arribar a conclusiones sobre si este método de SF aporta o no para el caso del reconocimiento del instrumento.

4.6. Resultados del modelo de detección de onsets. Comparación con y sin separación de fuentes

En esta sección, compararemos los resultados de entrenar modelos con las pistas completas (incluye batería, bajo, guitarra y voz) con y sin separación de fuentes aplicada previamente. Más precisamente, en un caso entrenamos el modelo de RNN con la pista completa de modo que aprenda a distinguir sonidos de batería de otros sonidos. En el otro, utilizamos Hybrid-Demucs como modelo de separación de fuentes (SF), sobre la pista completa de modo de poder aislar la batería. Luego sobre la pista de batería predicha por el modelo de SF entrenaremos un mismo modelo RNN de detección de eventos pero con esta nueva señal que intenta incluir sólo los sonidos de batería. Los resultados se muestran a continuación en la tabla 17.

	Resultados de SS, combinado con DO		
Método	Recall	Precision	F1 Score
Sin separación de fuentes	0.779	0.864	0.819
Con separación de fuentes	0.875	0.891	**0.883**

Cuadro 17: Resultados provenientes del uso del método de RNN para detección de onsets. Uno de los métodos utilizó separación de fuentes (Hybrid-Demucs) como parte del preprocesamiento de la señal para quedarse con solamente la pista de batería. El otro método utilizó directamente la pista completa con bajo, batería, guitarra y voz pero sin un método de separación de fuentes como preprocesamiento.

Puede notarse que el modelo entrenado con separación de fuentes funciona mejor que el modelo entrenado utilizando la mezcla completa: mientras que el primero tiene un F1-Score de 0.883, el segundo posee uno de 0.819. Esto nos lleva a pensar que la separación de fuentes colabora a nuestro modelo de detección de eventos y por ende lo hará también cuando tengamos este mismo combinado con reconocimiento del instrumento (sección 4.7).

4.7. Combinación del modelo de detección de onsets y reconocimiento: Comparación con y sin separación de fuentes

En esta sección, compararemos los resultados de entrenar modelos con las pistas completas (incluye batería, bajo, guitarra y voz) con y sin separación de fuentes aplicada previamente. Es decir, en un

caso entrenamos el modelo de RNN+CNN (detección de eventos y reconocimiento del instrumento entrenados por separado) con la pista completa de modo que aprenda a distinguir sonidos de batería de otros sonidos (clasificados también como 'other'). En el otro, utilizamos Hybrid-Demucs como modelo de separación de fuentes (SF), sobre la pista completa de modo de poder aislar la batería. Luego sobre la pista de batería predicha por el modelo de SF entrenaremos un mismo modelo CNN de reconocimiento del instrumento y otro de detección de eventos RNN; pero ambos con esta nueva señal que intenta incluir sólo los sonidos de batería. Los resultados se muestran a continuación en la tabla 18.

	Resultados de SS, combinado con RI y DO.				
Método	Instrumento	Recall	Precision	F1-Score	Macro F1-Score
Sin separación de fuentes	CY	0.253	0.731	0.376	0.6625
	HH	0.701	0.726	0.713	
	KD	0.849	0.942	0.893	
	SD	0.605	0.746	0.668	
Con separación de fuentes	CY	0.284	0.768	**0.415**	**0.713**
	HH	0.77	0.754	**0.762**	
	KD	0.891	0.958	**0.923**	
	SD	0.734	0.768	**0.75**	

Cuadro 18: Resultados provenientes de la combinación del detector de eventos basado en RNN, y CNN. Uno de los métodos utilizó separación de fuentes (Hybrid-Demucs) como parte del preprocesamiento de la señal para quedarse con solamente la pista de batería. El otro método utilizó directamente la pista completa con bajo, batería, guitarra y voz pero sin un método de separación de fuentes como preprocesamiento. La diferencia con los resultados de la sección anterior subyace en que en este caso utilizamos el método de detección de eventos de RNN mientras que en la otra utilizamos las etiquetas de verdad de campo como base.

En este caso notamos que las diferencias de F1-Score resultan amplias a favor del modelo de separación de fuentes para cada tipo de instrumento. También lo es así para el Macro F1-Score. De aquí se desprende que podemos concluir que el modelo de SS contribuye a la detección del onset y reconocimiento del instrumento.

4.8. Resultados de reconocimiento del instrumento desagregados por canción

También, resulta de interés evaluar el rendimiento de los modelos a nivel canción o pista, para el modelo de CNN binario, tal como se muestra en el cuadro 19.

annotation_path	Género	Complejidad	F1-Score				Onsets count			
			KD	SD	HH	CY	KD	SD	HH	CY
MusicDelta_Hendrix	Rock	5	1	1	1		32	32	64	0
MusicDelta_SwingJazz	Jazz	8	0.995	0.941	0.748	0.333	82	200	85	4
MusicDelta_FreeJazz	Jazz	10	0.976	0.946	0.229	0.606	129	329	7	211
MusicDelta_Beatles	Rock	6	0.979	0.795			47	45	0	0
MusicDelta_Country1	Otro	4	1		1		32	0	37	0
MusicDelta_SpeedMetal	Rock	7	0.966	0.953	0.803	0.222	114	67	105	4
MusicDelta_Punk	Rock	8	0.904	0.944	0.665	0.505	73	42	50	33
MusicDelta_ModalJazz	Jazz	9	0.941	0.99	0.825	0.605	27	241	177	371
MusicDelta_Gospel	Otro	7	0.982	0.946	0.808	0.279	140	137	212	11
MusicDelta_LatinJazz	Jazz	9	0.988	0.956	0.839	0.708	114	199	179	99
MusicDelta_Grunge	Rock	5	0.977	0.849	0.793	0.5	88	58	120	2

Cuadro 19: Resultados del modelo de reconocimiento del instrumento basado en CNN desagregados a nivel pista (out of sample). Las celdas vacías en el F1-Score de algunos instrumentos representan casos donde la ocurrencia del mismo es nula.

Donde:

- **annotation_path** representa el nombre de la anotación (abreviado) de la pista.
- **Género**, constituye el género de la pista.
- **Complejidad**, representa el nivel de dificultad percibida al reconocer manualmente (sin el uso de un sistema de TAB) los onsets e instrumentos presentes en la pista de batería. Es muy importante aclarar que este nivel fue anotado manualmente por el autor, con base en su percepción personal, por lo que se reconoce que es de naturaleza subjetiva y puede variar según la persona que lo evalúe.

Podemos notar que el F1-Score no se comporta homogéneamente para todas las pistas, es decir que posee cierta variabilidad a lo largo de estas. Para el KD, la diferencia de F1-Score entre el mínimo y el máximo es inferior a 0.1; mientras que para el SD es superior a 0.2; y mayor aún, resulta para el HH y el CY. Es importante notar que se incluye la cantidad de eventos (en el campo onset_counts del cuadro 19) puesto que el bajo tamaño muestral en algunos casos hacen que la métrica no tenga sentido para esa canción e instrumento. Existen también casos extremos como, por ejemplo, el del HH en MusicDelta_Beatles, donde este no ocurre y es por ello que su respectiva celda se encuentra vacía.

Por otro lado, notamos que no existe relación clara entre el nivel de complejidad asignado y el desempeño en la tarea de reconocimiento del instrumento, tal como se observa en la figura 12.

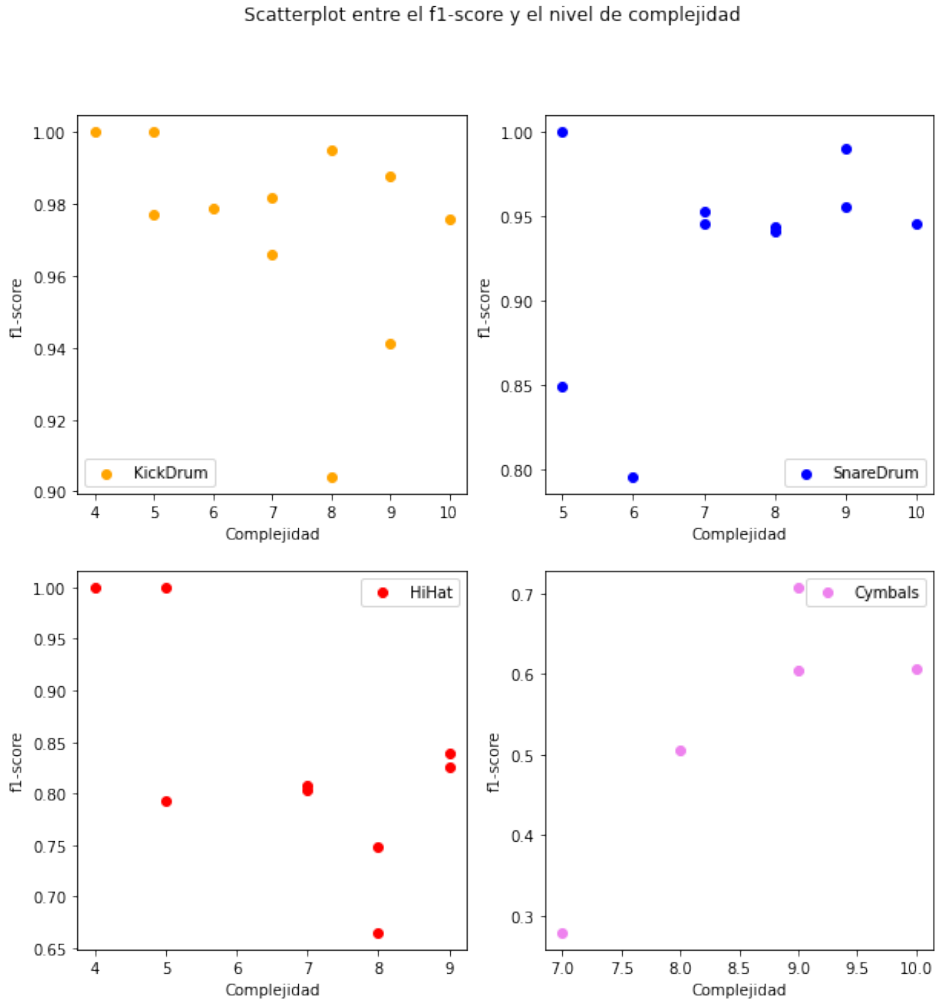

Figura 12: Diagrama de dispersión entre el F1-Score y el nivel de complejidad (asignado subjetivamente) para los distintos tipos de batería. Cada punto en el diagrama representa una canción. Las canciones con menos de 10 onsets para el tipo de instrumento no se tienen en cuenta.

En el único caso donde puede considerarse alguna relación es para el caso de los platillos o Cymbals (CY), aunque dicha relación no necesariamente puede ser robusta debido a la baja cantidad de

observaciones (canciones) disponibles.

También, se observan diferencias entre los F1-Scores promedio a lo largo de los géneros para los diferentes instrumentos tal como se muestra en el cuadro 20, pero no consideramos que sean lo suficientemente grandes como para arribar a alguna conclusión. Debemos recordar que para calcular estos promedios contamos con cuatro observaciones de Jazz, cuatro de Rock y dos que calificamos bajo "Otro", además de omitir las pistas que contaban con menos de diez onsets para el tipo de instrumento en cuestión (resultando, en un número de pistas menor aún, dependiendo del tipo de instrumento).

Género	F1-Score Promedio			
	KD	SD	HH	CY
Jazz	0.975	0.958	0.804	0.64
Otro	0.991	0.946	0.904	0.279*
Rock	0.965	0.908	0.815	0.505

Cuadro 20: F1-Score promedio entre pistas (CNN-Reconocimiento del instrumento) para los distintos tipos de instrumentos de batería. Aquellas pistas que cuenten con menos de 10 onsets para el instrumento en cuestión no son consideradas. Debido a lo mencionado anteriormente, en el caso del CY para el género "Otro" contábamos con sólo una observación para calcular el promedio, es por eso que se marca en el cuadro con un asterisco.

5. Discusión

En este trabajo, indagamos comparativamente sobre el rendimiento de los modelos de reconocimiento del instrumento, utilizando el conjunto de datos "MDB Drums". Notamos que el rendimiento del mejor modelo (CNN) no decrecía significativamente tras incorporar un detector de eventos en lugar de usar las anotaciones de groundtruth.

Posteriormente, probamos si al incorporar la pista con todos los instrumentos y no solo la batería aislada impactaba en el rendimiento del sistema. Se encontró que este hecho influenciaba sobre todo en el modelo de detección de eventos como puede verse en la sección 4.6 aunque no tanto sobre el modelo de reconocimiento del instrumento como podemos observar en la sección 4.5.

Mostramos que aplicar un modelo de separación de fuentes contribuía al rendimiento del sistema de TAB compuesto por el modelo de detección de eventos y reconimiento del instrumento. Es decir, que entrenar un modelo de detección de eventos y de reconocimiento del instrumento con las pistas completas otorgaba peores resultados que entrenar un modelo igual pero con la predicción de la señal de batería otorgada por el algoritmo de separación de fuentes sobre la pista completa.

Observamos tanto mediante visualización del espectrograma (sección 3.1.3) como con escucha del sonido, que el algoritmo de separación de fuentes permitía aislar la señal de batería con claridad en la mayoría de los casos y esto explica parte de su éxito.

Cuando vamos a la comparativa de los modelos de reconocimiento, encontramos que los modelos basados en CNN exhiben mejor desempeño que el resto. Aquí es importante destacar el desempeño de tres modelos: El primero es el modelo de CNN vectorial, que permite hacer la inferencia con menores requisitos de memoria y mayor velocidad que los de CNN binarios. El segundo, el modelo de CLAP, que permitió realizar inferencia de manera zero-shot y resulta un área prometedora de investigación, puesto que permite hacer inferencias sin entrenamiento previo; este presento los mejores resultados sobre el CY cuando se realizó fine-tuning del mismo, pero no en su versión zero-shot. El tercero es el modelo de CNN basado en las features de MERT, que resultó ganador en algunos de los instrumentos (HH y KD), si bien no fue el que tuvo mayor macro F1-Score.

Quedará para trabajos futuros, plantear diferentes arquitecturas para la red que utiliza MERT y verificar los resultados subyacentes, tal como se señala en la sección 3.2.4.5. Nos queda también realizar comparaciones con y sin normalización de la señal de entrada para el caso de los modelos de CNN, aunque como mencionamos en la sección 3.2.2.3, no pensamos que esto último lleve a mayores cambios en la performance de nuestros modelos.

Para los instrumentos de KD, SD, HH el F1-Score redondeado del mejor modelo (CNN-Reconocimiento del instrumento) fue de .98, .94 y .81 respectivamente, lo que se considera aceptable; mientras que para CY el F1-score ha sido de .63 y el resto de los modelos (OT, TT) han exhibido un rendimiento nulo o casi nulo. Se lo considera el mejor modelo puesto que es aquel que tiene mayor Macro F1-Score (0.838) y es el lider en el instrumento de redoblante con un F1-Score de 0.94 como mencionamos anteriormente.

Muy posiblemente las causas del bajo rendimiento en CY sean por un lado, que estos sonidos tienen mucho menor decay; y por otro, que tienen pocas observaciones, que tienen que ser divididas a su vez en entrenamiento y validación. Este problema de escasez de datos, ocurre más pronunciadamente en el caso de TT (toms) y de pandereta/side stick (OT), contando en cada caso con menos de 100 registros, que tienen que ser divididos en desarrollo y out of sample. Un posible camino de acción sería reforzar la cantidad de datos de entrenamiento con técnicas de data augmentation o bien utilizando otros conjuntos de datos tales como ENST Drums [33] para entrenar e incrementar por ende la cantidad de datos de entrenamiento, manteniendo constante el tamaño de out of sample.

En lo que respecta a la desagregación de resultados, obtuvimos los F1-Scores separados por canción junto con su recall y precision para el modelo de reconocimiento del instrumento basado en CNN. Notamos que no se observan relaciones claras entre el F1-Score y (a) el género de la pista o (b) el nivel de complejidad asignado subjetivamente. Resulta dificultoso arribar a conclusiones significativas cuando se trata de 10 pistas, es por ello que al igual que otros trabajos, recalcamos que el tamaño muestral - conseguir anotaciones manuales - todavía es un desafío pendiente no culminado en esta área.

Referencias

[1] Chih-Wei Wu et al. «A review of automatic drum transcription». En: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* PP (abr. de 2018), págs. 1-1. DOI: 10.1109/TASLP.2018.2830113.

[2] Sebastian Böck y Gerhard Widmer. «MAXIMUM FILTER VIBRATO SUPPRESSION FOR ONSET DETECTION». En: *Proceedings of the 16th International Conference on Digital Audio Effects (DAFx-13)*. Maynooth, Ireland, sep. de 2013, págs. 55-61.

[3] Lucas Thompson, Simon Dixon y Matthias Mauch. «Drum Transcription via Classification of Bar-Level Rhythmic Patterns». En: *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*. 2014.

[4] Ryoto Ishizuka et al. «Tatum-Level Drum Transcription Based on a Convolutional Recurrent Neural Network with Language Model-Based Regularized Training». En: *2020 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*. 2020, págs. 359-364.

[5] Céline Jacques y Axel Roebel. «Automatic drum transcription with convolutional neural networks». En: *Proceedings of the 21st International Conference on Digital Audio Effects (DAFx-18), Aveiro, Portugal, September 4–8, 2018*. Aveiro, Portugal, sep. de 2018. URL: https://hal.science/hal-02018777.

[6] Carl Southall, Ryan Stables y Jason Hockman. «Automatic Drum Transcription Using Bi-Directional Recurrent Neural Networks». En: *International Society for Music Information Retrieval Conference*. 2016. URL: https://api.semanticscholar.org/CorpusID:2891003.

[7] Keunwoo Choi y Kyunghyun Cho. «Deep Unsupervised Drum Transcription». En: *International Society for Music Information Retrieval Conference*. 2019. URL: https://api.semanticscholar.org/CorpusID:139087497.

[8] Benjamin Elizalde et al. «CLAP Learning Audio Concepts from Natural Language Supervision». En: *ICASSP 2023 - 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2023, págs. 1-5. DOI: 10.1109/ICASSP49357.2023.10095889.

[9] Yizhi LI et al. «MERT: Acoustic Music Understanding Model with Large-Scale Self-supervised Training». En: *The Twelfth International Conference on Learning Representations*. 2024. URL: https://openreview.net/forum?id=w3YZ9MSlBu.

[10] Sebastian Böck y Gerhard Widmer. «Maximum Filter Vibrato Suppression for Onset Detection». En: *Proceedings of the 16th International Conference on Digital Audio Effects (DAFx-13)*. Maynooth, Ireland, sep. de 2013, págs. 55-61.

[11] Alexandre Défossez. *Hybrid Spectrogram and Waveform Source Separation*. 2022. arXiv: 2111.03600 [eess.AS].

[12] Carl Southall et al. «MDB Drums: An annotated subset of MedleyDB for automatic drum transcription». En: *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*. Oct. de 2017. URL: https://musicinformatics.gatech.edu/wp-content_nondefault/uploads/2017/10/Wu-et-al_2017_MDB-Drums-An-Annotated-Subset-of-MedleyDB-for-Automatic-Drum-Transcription.pdf.

[13] Eric Battenberg. «Techniques for Machine Understanding of Live Drum Performances». Tesis doct. EECS Department, University of California, Berkeley, dic. de 2012. URL: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2012/EECS-2012-250.html.

[14] Yizhi Li et al. *MAP-Music2Vec: A Simple and Effective Baseline for Self-Supervised Music Audio Representation Learning*. 2022. arXiv: 2212.02508 [cs.SD].

[15] Jacob Devlin et al. «BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding». En: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, jun. de 2019, págs. 4171-4186. DOI: 10.18653/v1/N19-1423.

[16] Wei-Ning Hsu et al. «HuBERT: Self-Supervised Speech Representation Learning by Masked Prediction of Hidden Units». En: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 29 (2021), págs. 3451-3460. DOI: 10.1109/TASLP.2021.3122291.

[17] Alexandre Défossez et al. «High Fidelity Neural Audio Compression». En: *Transactions on Machine Learning Research* (2023). Featured Certification, Reproducibility Certification. ISSN: 2835-8856. URL: https://openreview.net/forum?id=ivCd8z8zR2.

[18] Alec Radford et al. *Learning Transferable Visual Models From Natural Language Supervision*. 2021. arXiv: 2103.00020 [cs.CV].

[19] Qiuqiang Kong et al. «PANNs: Large-Scale Pretrained Audio Neural Networks for Audio Pattern Recognition». En: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 28 (2020), págs. 2880-2894. DOI: 10.1109/TASLP.2020.3030497.

[20] D. Jurafsky y J.H. Martin. *Speech and Language Processing. An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Youcanprint, 2023. ISBN: 9791221476842. URL: https://books.google.com.ar/books?id=b3PuzwEACAAJ.

[21] Jeffrey L. Elman. «Finding Structure in Time». En: *Cognitive Science* 14.2 (1990), págs. 179-211. DOI: https://doi.org/10.1207/s15516709cog1402_1. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1207/s15516709cog1402_1. URL: https://onlinelibrary.wiley.com/doi/abs/10.1207/s15516709cog1402_1.

[22] Sebastian Böck, Florian Krebs y Markus Schedl. «Evaluating the Online Capabilities of Onset Detection Methods». En: *International Society for Music Information Retrieval Conference*. 2012. URL: https://api.semanticscholar.org/CorpusID:7379180.

[23] E. Vincent, R. Gribonval y C. Fevotte. «Performance measurement in blind audio source separation». En: *IEEE Transactions on Audio, Speech, and Language Processing* 14.4 (2006), págs. 1462-1469. DOI: 10.1109/TSA.2005.858005.

[24] Jonathan Le Roux et al. «SDR – Half-baked or Well Done?» En: *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2018), págs. 626-630. URL: https://api.semanticscholar.org/CorpusID:53246666.

[25] Brian McFee et al. «librosa: Audio and music signal analysis in python». En: *Proceedings of the 14th python in science conference*. Vol. 8. 2015.

[26] Sebastian Böck et al. «madmom: a new Python Audio and Music Signal Processing Library». En: *Proceedings of the 24th ACM International Conference on Multimedia*. Amsterdam, The Netherlands, oct. de 2016, págs. 1174-1178. DOI: 10.1145/2964284.2973795.

[27] Fabian Pedregosa et al. «Scikit-learn: Machine learning in Python». En: *Journal of machine learning research* 12.Oct (2011), págs. 2825-2830.

[28] Adam Paszke et al. «PyTorch: An Imperative Style, High-Performance Deep Learning Library». En: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, págs. 8024-8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[29] Francois Chollet et al. *Keras*. 2015. URL: https://github.com/fchollet/keras.

[30] Thomas Wolf et al. *Transformers: State-of-the-Art Natural Language Processing*. Oct. de 2020. URL: https://github.com/huggingface/transformers.

[31] Wes McKinney et al. «Data structures for statistical computing in python». En: *Proceedings of the 9th Python in Science Conference*. Vol. 445. Austin, TX. 2010, págs. 51-56.

[32] Charles R. Harris et al. «Array programming with NumPy». En: *Nature* 585 (2020), págs. 357-362. DOI: 10.1038/s41586-020-2649-2.

[33] Olivier Gillet y Gaël Richard. «ENST-Drums: an extensive audio-visual database for drum signals processing». En: *International Society for Music Information Retrieval Conference*. 2006. URL: https://api.semanticscholar.org/CorpusID:14692293.