UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

# Segmentación de Música EDM con técnicas de Aprendizaje Profundo

**David Tolosa Alarcón**

**Tutor: Jose Javier Lopez Monfort**

Trabajo Fin de Máster presentado en la Escuela Técnica Superior de Ingeniería de Telecomunicación de la Universitat Politècnica de València, para la obtención del Título de Máster en Ingeniería Telecomunicación

Curso 2020-2021

Valencia, 30 de Septiembre de 2021

Escuela Técnica Superior de Ingeniería de Telecomunicación
Universitat Politècnica de València
Edificio 4D. Camino de Vera, s/n, 46022 Valencia
Tel. +34 96 387 71 90, ext. 77190
**www.etsit.upv.es**

VLC/ CAMPUS VALENCIA, INTERNATIONAL CAMPUS OF EXCELLENCE

EMAS Gestión ambiental verificada REG.NO. ES-CV- 000030

## Resumen

El presente trabajo ha realizado una investigación sobre cuáles son las técnicas y modelos más empleados en Deep Learning para la segmentación musical dentro del estado del arte actual. Utilizando una base de datos de 150 canciones del género EDM, desarrollado por el Instituto de Telecomunicaciones y Aplicaciones Multimedia de la UPV, El estudio ha abordado la extracción de características acústicas de las canciones, así como la implementación, entrenamiento y evaluación de redes neuronales para la creación de un modelo que realice la segmentación musical sin la necesidad de una mayor ingeniería.

## Resum

El present treball ha realitzat una investigació sobre quines són les tècniques i models més empleats de Deep Learning per a la segmentació musical dins de l'estat de l'art actual. Utilitzant una base de dades de 150 cançons del gènere EDM, desenrotllat per l'Institut de Telecomunicacions i Aplicacions Multimèdia de la UPV, L'estudi ha abordat l'extracció de característiques acústiques de les cançons, així com la implementació, entrenament i avaluació de xarxes neuronals per a la creació d'un model que realitze la segmentació musical sense la necessitat d'una major enginyeria.

## Summary

The present work has conducted research on what are the most used techniques and models of Deep Learning for music segmentation within the current state of the art. Using a database of 150 songs of the EDM genre, developed by the Institute of Telecommunications and Multimedia Applications of the UPV, the study has addressed the extraction of acoustic characteristics of the songs, as well as the implementation, training and evaluation of neural networks for the creation of a model that performs music segmentation without the need for greater engineering.

# Índice general

# Capítulo 1

# Introducción

Dentro del análisis musical, la determinación de fronteras (ej. *Entre estribillo y verso*) es una de las tareas más importantes para poder comprender y estudiar las estructuras presentes en una pieza musical. La anotación de segmentos musicales (*el inicio y el fin de cierta estructura*) es muy utilizada en aplicaciones que combinan música con vídeo o animaciones, como es en el caso de todo el equipo de visualización e iluminación que se utiliza en eventos de música en directo, ya que para poder coordinar los eventos que ocurren en la canción con las imágenes que se proyectan o el tipo de iluminación que se crea en el escenario **necesitamos conocer de antemano cuándo ocurren estos eventos**.

Pero esta tarea, a simple vista sencilla, presenta cierta ambigüedad que complica la generalización de un método. Por ejemplo, si dos expertos musicales se les asignara la tarea de segmentar una canción en sus estructuras lo más probable es que las anotaciones hechas por uno difieran del otro, dentro de un margen perceptible por ambos. Dicho de otra manera, **no existe un consenso dentro de los oyentes en el campo de la segmentación musical**.

Si partimos de esta premisa comprendemos rápidamente que la **automatización** de la detección de estructuras es un proyecto que tiene ciertas dificultades. Empezando por la misma evaluación, ya que ésta se realizará comparando los resultados con anotaciones hechas a mano y como se ha dicho, no existe una generalización por parte de los profesionales, lo cual va a hacer que tengamos que asumir **un cierto error de base en la precisión de las anotaciones generadas** por dicho software o aplicación. Además cada género musical tiene sus propias estructuras y si nos adentramos dentro de cada subgénero, las estructuras también cambian tanto en su orden como en su composición. Por lo que es complejo realizar un *"software universal"* que cree anotaciones válidas, incluso dentro del mismo género musical.

Todos los algoritmos creados y que son utilizados actualmente para abordar esta tarea precisan de un diseño *"end-to-end"*, es decir, se necesita diseñar y optimizar manualmente todos los procesos necesarios. Por lo que se puede decir que estas aplicaciones requieren de una alta cantidad de ingeniería y tiempo necesario para su implementación, que además por contra no es posible generalizar, **resultando que el desarrollo de estas aplicaciones sea costoso**.

Este trabajo aborda la automatización de detección de fronteras entre estructuras para canciones del **género EDM** (*Electronic Dance Music*). El Grupo de Tratamiento de Audio y Comunicaciones, GTAC [1] , del Instituto de Telecomunicaciones y Aplicaciones Multimedia de la UPV, iTEAM [2], desarrolló para la 144ª Convención de Audio en Milan (2018) un algoritmo para la segmentación online de música EDM [3]. Utilizando para ello un set de 100 canciones de diferentes estilos EDM y con sus respectivas anotaciones de los eventos realizadas por un experto musical. En el diagrama de bloques de la Figura 1.1 se puede ver los pasos realizados para la implementación de esta aplicación. Este trabajo va a utilizar la misma base de datos más una aportación de 56 nuevas canciones, también todas dentro del género EDM, junto con sus anotaciones que nos ha proporcionado el grupo GTAC.

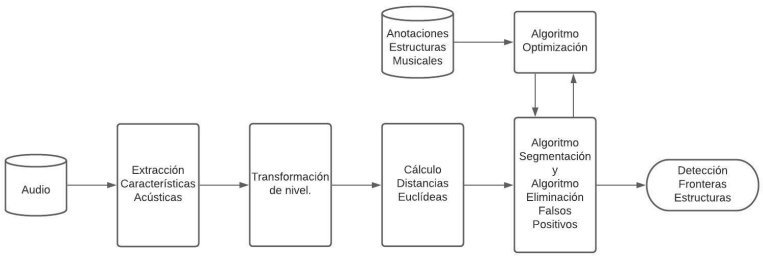


Figura 1.1: Diagrama de Bloques del Algoritmo presentado en [3]

Lo que se va a exponer en este proyecto es **la implementación de la automatización de segmentación musical** utilizando técnicas de **deep learning**. Esto quiere decir que **se ha entrenado y utilizado redes neuronales**, en concreto, redes convolucionales ó CNNs (del inglés: *Convolutional Neural Networks*) **para la detección de fronteras**. Para ello, y como es habitual dentro del campo del *Deep Learning*, se ha acudido a los trabajos realizados que definen el estado del arte actual. Es necesario decir que los resultados obtenidos por el estado del arte **no superan al de las aplicaciones diseñadas *"end-to-end"*** como la desarrollada en [3]. Pero también arrojan otros resultados que son reveladores: Primero, **el tiempo necesario para el diseño es mucho menor** y lo segundo, **los modelos pueden ser adaptados mucho más fácilmente a otros distintos bancos de datos**, para nuestro caso, se pueden entrenar un mismo modelo, o bastante similar, con distintos géneros y éste es capaz de **aprender las estructuras para cada género** sin necesidad de una gran intervención humana, es decir, a partir de una red neuronal, realizando modificaciones mínimas, podemos obtener modelos para cada uno de los géneros o subgéneros musicales sin necesidad de que tengamos que realizar distintas optimizaciones del algoritmo para cada tipo, **dejando que la red aprenda por sí sola estas diferencias**.

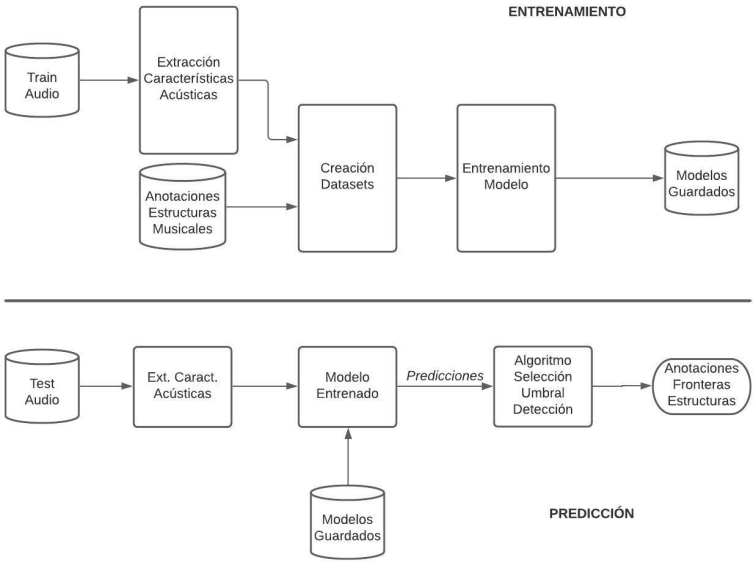


Figura 1.2: Diagrama de Bloques del entrenamiento del modelo y de la obtención de las predicciones de las fronteras de las estructuras musicales.

En la Figura 1.2, el proceso *Entrenamiento Modelo* abordaría lo que en la Figura 1.1 comprenderían a los procesos *Transformación de nivel, Cálculos Distancias Euclídeas, Algoritmo Optimización, Algoritmo Segmentación y Algoritmo Eliminación Falsos Positivos*. Aquí se puede ver cuánto se puede reducir la cantidad de *ingeniería* necesaria mediante el entrenamiento de redes neuronales profundas (del inglés: *deep neural networks*). Se obtendrán predicciones de los modelos entrenados (*Figura 1.2, DB inferior*) a las que se le aplica cierto umbral de verdad para refinar y mejorar las predicciones obtenidas. Los resultados obtenidos en este trabajo se encontrarían dentro de valores similares a otros proyectos desarrollados dentro del estado del arte, por debajo de los mejores modelos conseguidos. Aún así se refuerza la idea de que mediante deep learning se pueden conseguir soluciones que requieren de menor intervención humana y por tanto menos costosas, y dado que se ha entrenado con un dataset completamente distinto que representaba a un género concreto nunca antes entrenado, se puede decir que es posible diseñar métodos más generalizados que nos sirvan para abordar esta tarea y dar soluciones particulares de una forma más rápida.

# Capítulo 2

# Estado del Arte

A continuación se va a realizar un breve comentario sobre cuál es el Estado del Arte actual en segmentación musical y en machine learning, qué redes neuronales se utilizan para el análisis musical y cuales son los mejores resultados obtenidos mediante técnicas de Deep Learning.

## 2.1. Segmentación Musical

Como se ha descrito en 1, los algoritmos desarrollados para segmentación musical requieren de un estudio e implementación integral de todas las partes que involucran el proceso. En el trabajo desarrollado por Paulus et al.[4] se realiza un estudio de carácter general sobre análisis de estructuras musicales y concluye que existen tres maneras de acercarnos a este problema:

- **Novelty-based**: Detectando las transiciones entre dos partes contrastantes, es decir, encontrando las fronteras entre dos estructuras que presentan diferencias en sus características musicales (contraste).
- **Homogeneity-based**: Identificando secciones que son más o menos consistentes con sus propiedades musicales. Se busca las estructuras a partir de las posibles propiedades musicales que la forman.
- **Repetition-based**: Determina las estructuras a través de la recurrencia de éstas en la canción.

La mayoría de los algoritmos actuales para segmentación musical están basados en uno de estos tres acercamientos, o utilizan combinaciones de éstos. Todos ellos precisan de extraer características acústicas de las canciones, posibles descriptores de audio pueden ser: el timbre mediante el uso de MFCCs (del inglés: *Mel Frequency Cepstral Coeficients - Coeficientes Cepstrales en las Frecuencias de Mel*), el pitch, características rítmicas ó vectores chroma [5]. De esta manera podemos resaltar diferentes dimensiones perceptibles musicales. Para conseguir una mayor precisión temporal, se puede realizar la acumulación de características síncronamente con el beat, de manera que la resolución temporal de las características es la resolución propia de la canción, en [4] también se encuentra esto explicado.

Los algoritmos basados en la "novedad"(*novelty-based*) normalmente son implementados utilizando matrices de autosemejanza ó SSMs (del inglés *self-similarity Matrix*) y matrices de autodis-

tancia ó SDMs (del inglés *self-distance Matrix*). La técnica consiste en evaluar las características en cierto espacio temporal con el resto de características de la canción, al obtener la matriz los valores más similares o con menor distancia (dependiendo del tipo de técnica) quedarán resaltados de manera que se podrán identificar con cierta estructura de la canción. La Figura 3.8 muestra un ejemplo de SDM. Posteriormente se evalúa estas matrices para seleccionar los posibles bordes de la estructura y la eliminación de posibles falsos bordes mediante un mapeo de probabilidad. Asignando los bordes a los *frames* más probables.

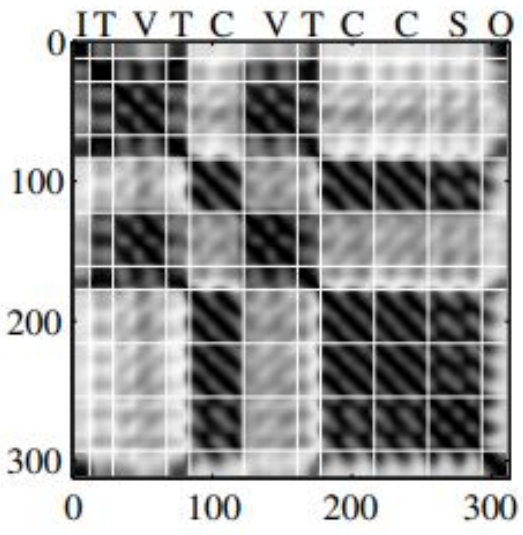


Figura 2.1: Ejemplo de SDM expuesto en [4] obtenido a partir de vectores chroma. Ha sido realizado con una frecuencia de corte baja haciendo que las diferentes estructuras parezcan bloques. La cuadrícula superpuesta indica los bordes de las estructuras obtenidas, indicadas en la parte superior como: Intro (I), Theme (T), Verse (V), Chorus (C), Solo (S) y Outro (O)

Los algoritmos basados en la homogeneidad (*Homogeneity-based*) suelen utilizar *Hidden Markov Models* ó HMMs [6]. Estos modelos ya han sido utilizados satisfactoriamente para aplicaciones de reconocimiento de voz y existe un interés por ellos dentro del campo del análisis musical. La idea consiste en utilizar HMMs para poder obtener un análisis musical basado en las *texturas* presentes en una pieza musical y obtener la segmentación de ésta a partir de la evaluación de las texturas.

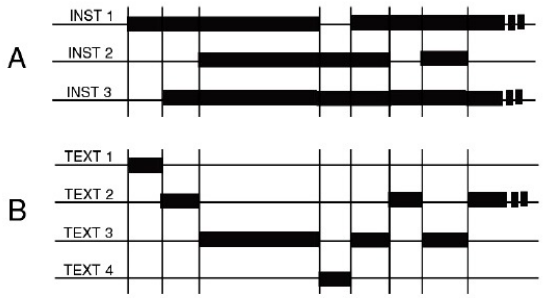


Figura 2.2: Comparación de representación musical en forma de track MIDI (A) y en texturas (B). Imagen obtenida de [6]

Por último, los algoritmos basados en la repetición utilizan SSMs ó SDMs cómo aquellos basados en la novedad (Figura 3.8) que son utilizadas por otro algoritmo para descubrir patrones y sus posibles fronteras [7]. La optimización de estos algoritmos puede realizarse de manera supervisada o no supervisada. Realizar una optimización o entrenamiento *supervisado* quiere decir que se utiliza anotaciones o datos ya clasificados para su optimización. En nuestro caso, selección de fronteras entre estructuras musicales, se utilizan anotaciones hechas por un profesional para comparar con las predichas con nuestro algoritmo. Los algoritmos que utilizan SDMs ó SSMs suelen utilizar optimización supervisada mientras que aquellos que utilizan HMMs no.

## 2.2. Redes Neuronales

Las redes neuronales artificiales existen desde al menos 1960, uno de los primeros avances conseguidos con el entrenamiento de este tipo de redes es el modelo **LeNet-5** conseguido por Yann LeCun et al. [8] (1998) para la detección de dígitos simples en imágenes. En las últimas décadas el interés por el entrenamiento de redes neuronales ha aumentado significativamente y ha culminado en lo que ahora se denomina **Deep Learning**. Las principales razones de este auge son: **La gran cantidad de información existente en el mundo**, **el aumento de la capacidad de computación de los ordenadores** y **el fácil acceso a la información (Internet)**. Dónde más claramente se han podido observar sus avances es dentro del campo de la imagen o en tareas relacionadas con la visión. Un ejemplo puede ser el modelo creado para la detección de caras por Huaizu Jiang et al. [9] o la famosa VGGnet para la clasificación de imagenes desarrollada por el grupo Visual Geometry Group de la Universidad de Oxford [10].

La ciencia o arte de programar modelos computacionales para que ellos puedan aprender de un cierto grupo de datos es conocido como **Machine Learning** (Deep Learning sería la evolución del ML con modelos mucho más grandes y con vólumenes de datos mucho mayores). Una definición más precisa es la indicada por Tom Mitchell en [11](traducida del inglés):

> Se dice que un programa de ordenador aprende de una experiencia $E$ respecto de una tarea $T$ con una actuación medida como $P$, si su funcionamiento en $T$, medido por $P$, mejora con cada experiencia $E$.

Para una mejor comprensión de este concepto se desarrolla el siguiente ejemplo: Queremos realizar un filtro de spam para nuestro correo, para ello vamos a entrenar un modelo que detecte automáticamente que correos son potencialmente spam y deben ser rechazados. Aquí nuestra tarea $T$ es detectar spam del correo entrante, la experiencia $E$ sería todos los correos (spam o no) que disponemos para poder entrenar el modelo ( serían nuestros datos de entrenamiento) y la forma de medir la actuación del programa sería evaluando el porcentaje de correos que ha indicado correctamente como spam. Si quisiésemos implementar el mismo programa pero con técnicas de programación clásicas, tendríamos que elaborar un algoritmo que detectara, por ejemplo, palabras clave que son típicas en estos correos de spam ( ej. *gratis, oferta, para ti, increíble,...*), además de otros patrones. El problema en sí ya es dificil, encontrar patrones dentro de los correos spam es algo subjetivo y además requiere repetir el proceso de optimización durante mucho tiempo hasta que hayamos encontrado los suficientes patrones para que el programa sea robusto, lo cual hace que problablemente tu programa tenga un gran número de reglas lo cual también es dificil de mantener o depurar. Sin embargo, utilizando ML nuestro filtro de spam aprendería automáticamente que palabras o frases son buenos indicadores de spam mediante la detección de patrones inusuales en los

correos de entrenamiento que le hemos dispuesto, aprendiendo a diferenciar entre correo de spam o no. Este programa sería mucho más corto, fácil de mantener y problabemente más preciso que el realizado mediante programación clásica. Además si los patrones del correo de spam cambian, una aplicación basada en ML también sería capaz de aprender a detectar estos cambios, mientras que de la otra manera tendríamos que añadir manualmente nuevas reglas para los nuevos patrones existentes.

Este ejemplo también sirve para exponer las claras ventajas que tiene el Deep Learning para la realización de tareas que requieren de largos procesos de optimización o que precisan de tener en cuenta numerosas reglas a la vez. A continuación se describe brevemente algunas tareas en las que los modelos ML han superado en actuación a los desarrollados por programación clásica o incluso a la propia capacidad humana. *Tanto el ejemplo como las tareas indicadas se han obtenido y traducido de [12] páginas [5-8]*.

- **Clasificación de Imágenes**: *Analizar imágenes de productos en una producción para automáticamente clasificarlos*
- **Clasificación de Texto**: *Detección automática de comentarios ofensivos en foros online.*
- **Segmentación de Imágenes**: *Detectar el número de coches presentes en una carretera.*
- **Segmentación Semántica**: *Detectar tumores en escáneres cerebrales*
- **Procesamiento Natural del Lenguaje (NLP)**: *Asistente Personal (Siri, Alexa,...) o los Chatbot.*
- **Reconocimiento de Voz**: *Hacer que tu aplicación reaccione a comandos de voz.*
- **Sistema de recomendación**: *Recomendar cierto producto a un cliente potencial basado en su historial de compras*
- **Agrupamiento (Clustering)**: *Segmentar clientes basándose en sus compras para poder realizar diferentes estrategias de marketing para cada segmento.*
- **Detección de anomalías**: *Detectar fraudes de tarjetas de crédito*

Cada una de estas tareas requiere de modelos específicos, por tanto las estructuras de la red así como sus componentes varían con respecto a cada tarea. Aunque dentro de una misma tarea es común acudir a los modelos ya implementados para así acercarnos más rápidamente a la solución. Existe un concepto dentro del ML conocido como **Transfer Learning** que consiste en utilizar modelos ya entrenados para una tarea similar específica ( ej. Clasificador de imágenes de perros y gatos) y utilizarlo directamente o con poco entrenamiento en otra tarea similar (ej. Clasificador de imágenes pero entre caballos y ovejas). Con esto se quiere decir que es normal acudir al estado del arte actual en cierta tarea para orientarse y proceder a la hora de entrenar un nuevo modelo, este trabajo no ha realizado *transfer learning*, debido a la imposibilidad de obtener otros modelos entrenados en esta tarea, pero si ha basado gran parte de su modelo en otros ya probados y que se detallarán más adelante.

## 2.3. Análisis Musical con Deep Learning

En el campo de la extracción de información musical ( del ingles: *Music Information Retrieval - MIR*) existen ya aplicaciones que hacen uso de técnicas deep learning y que su funcionamiento ha sobrepasado ya el de los algoritmos desarrollados de manera clásica. Para la tarea de *Clasificación de música por artista* el trabajo desarrollado por Zain Nasrullah et al.[13] utilizando redes convolucionales recursivas ó CRNNs refuerza la idea de que estos métodos establecen la nueva forma a proceder para desarrollar este tipo de aplicaciones. Otro ejemplo podría ser en la tarea de *Detección de inicio musical* (detectar dentro de un segmento de audio si existe contenido musical y dónde empieza) el trabajo realizado por Jan Schlüter et al.[14] indica que resulta más sencillo y rentable entrenar redes convolucionales ó CNNs que diseñar directamente una solución de inicio a fin. Otros trabajos que utilizan también CNNs supervisadas cuyos resultados superan al de los algoritmos clásicos son [15] para la *Clasificación musical por género* y [16] para el *Reconocimiento de acordes musicales*.

Los datos con los que se alimentan estas redes neuronales son características acústicas extraídas de los audios originales, dentro de todos los descriptores de audios los espectrogramas Mel ó MFCCs suelen funcionar bien para muchas de las tareas dentro del análisis musical con redes neuronales y son utilizados mayoritariamente. También se han probado combinaciones de MFCCs y SSMs como en [17] y últimamente se han utilizado espectrogramas obtenidos mediante STFT (*Short-Time Fourier Transform*) en tareas de clasificación con buenos resultados[18].

Dentro de la segmentación musical, el trabajo realizado por Karen Ullrich, Jan Schlüter y Thomas Grill[19] utilizando redes convolucionales arroja grandes resultados y este trabajo en gran parte ha basado sus procedimientos en los utilizados por aquél. Concluyen que a mayor información temporal en las muestras para alimentar la CNN, ésta es capaz de detectar con mayor precisión las fronteras de las estructuras, incluso si se reduce la cantidad de información total por muestra (lo cual permite poder entrenar más rápidamente una gran cantidad de datos). Ciertos métodos desarrollados en [19] serán utilizados y explicados posteriormente en este trabajo, ya que establecen una base para proceder en el análisis de la segmentación musical utilizando redes convolucionales.

Por último cabe destacar la actuación del ISMIR(*International Conferences on Music Information Retrieval*)[20] por la cantidad de investigaciones y trabajos expuestos, y en particular, de la comunidad MIREX(*Music Information Retrieval Evaluation eXchange*)[21] coordinada y dirigida por IMIRSEL(*International Music Information Retrieval Systems Evaluation Laboratory*)[22] que ofrece la necesaria estructura para la evaluación científica de técnicas desarrolladas por investigadores dentro del campo de la extracción de información musical. Desde 2004 que se fundó se publican anualmente trabajos sobre tareas del análisis musical, de aquí surgen numerosos trabajos realizados con técnicas deep learning que han servido de guía y orientación para la realización de este proyecto.

# Capítulo 3

# Método

El objetivo de este trabajo es entrenar una red neuronal convolucional (CNN) para que sea capaz de detectar las fronteras entre las estructuras musicales existentes en una canción. Para ello se utilizará una base de datos compuesta por canciones de tipo EDM, de las cuales se extraerán características acústicas que serán los datos a analizar por la red, además de anotaciones hechas a mano indicando el segmento exacto dónde inician y terminan las diversas estructuras. Estas anotaciones o etiquetas nos servirán para entrenar el modelo a que detecte estas fronteras. La red neuronal resuelve un problema de *Clasificación Binaria*. Si tenemos un array que representa las características acústicas de una canción de manera que cada muestra representa un instante temporal, para cada muestra en una canción, el modelo las clasificará entre potenciales fronteras (con un valor positivo) o las descartará (valor cero). La utilización de CNNs para la segmentación musical tiene una serie de problemas que se tendrán que resolver previamente para poder entrenar correctamente los modelos y obtener buenos resultados. Todas estas cuestiones se explicarán en las secciones siguientes.

## 3.1. Arquitectura Red Neuronal

Varios trabajos han utilizado redes convolucionales con espectrogramas para realizar ciertas tareas del análisis musical ([14, 19, 23] ya comentados en Sección 2.3). Se conoce los buenos resultados que ofrecen estos modelos sobre todo en el campo de la imagen [10].En deep learning una red neuronal convolucional o CNN es un tipo de red típicamente utilizada para el reconocimiento de patrones presentes en imágenes pero también son usadas para análisis de representación de datos o procesado de señal entre otros. El funcionamiento de las CNN para la detección de fronteras en espectrogramas es bastante parecido a como funcionan en el campo de la imagen para la detección de bordes, por lo que la arquitectura de los modelos para esta tarea es heredada de los trabajos realizados en la segmentación y clasificación de imágenes, Si tuviésemos que definir la topología básica que define una red neuronal convolucional tenemos que acudir al modelo VGGnet[10]. Muchas aplicaciones y modelos actuales que se implementan con redes convolucionales tienen como base esta arquitectura.

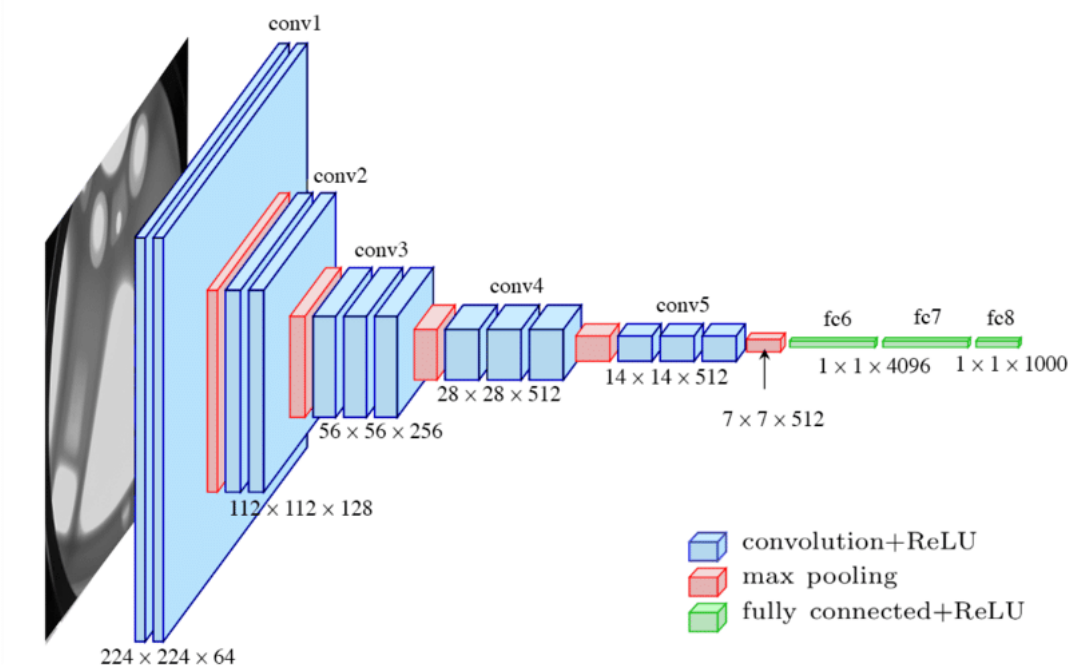


Figura 3.1: Topología de la red neuronal convolucional VGG16 propuesta en [10].

Este modelo presenta tres tipos de capas o *layers* distintos y en un orden concreto: **Convolutional layers, MaxPooling Layers y Fully Connected Layer**. Los distintos modelos que derivan de éste pueden diferir en el número de capas, el tamaño de los kernels, el número de neuronas en las capas FC... Pero todos presentan estos componentes y en la misma estructura (FC siempre al final, combinación de capas convolucionales seguidas de capas Maxpooling). A continuación se describirá brevemente las distintas capas que forman una CNN.

### 3.1.1. Convolutional Layer

La primera capa de una CNN siempre es una capa convolucional. Estas capas realizan una operación de convolución sobre los datos de entrada pasando los resultados a la siguiente capa. La convolución es realizada por filtros o kernels siempre de un tamaño menor a las dimensiones de los datos de entrada, de manera que el filtro pasa por muchas partes de la entrada extrayendo información útil. El concepto de convolución del kernel con los datos de entrada es el mismo que el de filtrado en procesado digital de imagen. En las CNNs los valores de los filtros son aleatorios, pero se van optimizando según es entrenada la red. La utilidad de estas capas es que con kernels de tamaño pequeño (respecto a la dimensión del dato de entrada) aunque sus valores sean aleatorios resulta que son capaces de detectar bordes, lo cual resulta muy útil para nuestra tarea ya que las fronteras entre estructuras musicales representadas en un espectrograma muestran bordes abruptos que pueden ser detectados por estas capas (Figura 3.3).

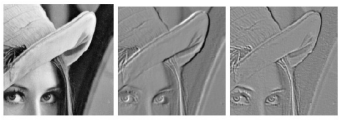

Figura 3.2: Resultado de convolucionar una imagen (*izquierda*) con un kernel aleatorio de tamaño 5x5 detecta bordes orientados(*centro y derecha*). *Imagen obtenida de [14]*.

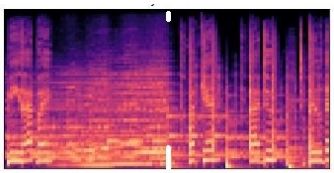

Figura 3.3: Muestra de un espectrograma con frontera marcada por la línea blanca.

Estas capas precisan saber la dimensión del dato de entrada ( el tamaño de la imagen), el tamaño del kernel y también el número de filtros a emplear. Cada filtro se inicializará con valores aleatorios distintos lo cual ofrece a la red distintos procesados de la entrada que resalten distinta información.

### 3.1.2. MaxPooling Layer

Las capas convolucionales en una CNN extraen una serie de características de los datos de entrada, un problema existente con el mapa de características que se obtienen a la salida de estas capas es que **son sensibles a la localización de la característica en la entrada**. Un enfoque para abordar este problema consiste en **reducir el número de características del mapa** resultante (*downsampling*). Con esto lo que se consigue es que el mapa resultante sea más robusto a cambios en la localización de la característica en los datos de entrada. Este concepto se conoce técnicamente como *local translation invariance*.

Las capas de agrupación ó *Pooling layers* proporcionan una forma para reducir el número de muestras en un mapa de características. Dos métodos comunes de agrupación son: Max Pooling y Average Pooling. Estas capas también hacen uso de un kernel, éste agrupa los datos de la entrada (*por ejemplo, un kernel de 3x3 recoge datos de un mapa de entrada, 9 datos en total, y obtiene a su salida un único valor*) a través de la media de los valores (*average pooling*) o obteniendo el máximo valor para ese grupo (*max pooling*). Al igual que en las capas convolucionales el tamaño del kernel ha de ser menor al tamaño de los datos de entrada. Si se utilizan conjuntamente capas convolucionales y capas de agrupación lo que obtenemos son mapas de características más robustos y que además reducen la cantidad total de parámetros entrenables lo cual también ayuda a evitar que la red neuronal se sobreajuste demasiado a los datos del entrenamiento (*overfitting*), este concepto se explicará más adelante.

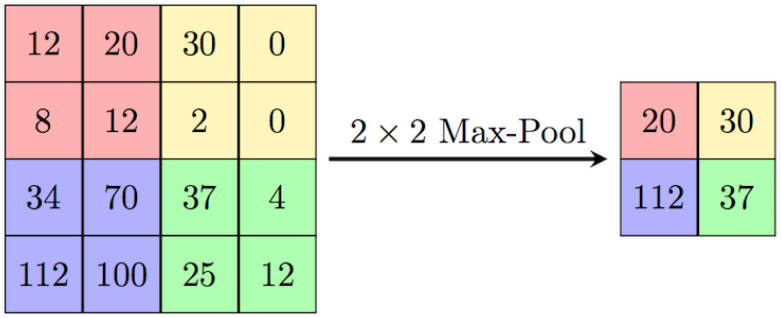


Figura 3.4: Ejemplo de Max Pooling con un kernel 2x2.

### 3.1.3. Fully Connected Layer

En una red neuronal una capa completamente conectada (*Fully Connected - FC*) es aquella en el que los datos de entrada provenientes de la capa anterior **están conectados a cada una de las neuronas de esta capa**. Éstas capas, que se suelen encontrar en la parte final de la estructura en una red convolucional, **compilan todos los datos de las capas anteriores**. Están formadas por una serie de celdas de activación ó neuronas, En redes convolucionales se utilizan estas capas para resumir y analizar las características extraidas previamente por las capas convolucionales y de agrupación. También corresponde a este tipo de capa **tomar la decisión a partir de lo aprendido de los datos de entrada**, en nuestro caso decidir si el dato de entrada corresponde a una frontera (1) o no (0). Como es un problema de clasificación binaria solo se utiliza una única neurona en la última capa FC, ésta a partir de su activación y con los datos previos obtenidos ofrecerá un valor entre 0 y 1 que será la decisión tomada por el modelo.

### 3.1.4. Modelo

La arquitectura desarrollada en este trabajo es casi totalmente heredada del modelo creado en [19]. Se realizó un estudio del tamaño de los kernels para las capas convolucionales y de agrupación, así como el número de filtros sin llegar a resultados más óptimos que los descritos por el trabajo de Karen Ullrich et al. Sólamente consiguiendo mejorar el modelo al utilizar más capas convolucionales (*hacer la red neuronal más profunda*) pero que debido a limitaciones de tiempo y de capacidad computacional no se ha implementado finalmente (las capas convolucionales son aquellas dónde la red neuronal consume más tiempo, al incrementarlas aumentamos considerablemente el tiempo de procesado del modelo). La arquitectura del modelo entrenado tiene la siguiente forma (desde el inicio de la red hasta su final):

- **Capa Convolucional**. Tamaño kernel : (8,6). Número de filtros: 16
- **Max Pooling Layer**. Tamaño kernel : (6,3).
- **Capa Convolucional**. Tamaño kernel: (6,3). Número de filtros: 32
- **Dropout**. Porcentaje: 0.25
- **Fully Connected Layer**: Número activaciones o neuronas: 128.
- **Dropout**. Porcentaje: 0,5
- **Fully Conected Layer**: Capa de decisión, una sola celda de activación.

Dropout no es una capa en sí, si no una técnica utilizada a menudo en conjunto con las capas FC. Lo que realiza es que desactiva de manera aleatoria una serie de celdas dentro de una capa FC. Se indica para ello el porcentaje de celdas a desactivar. Esto tiene un objetivo y es ayudar al modelo a que **generalice**. Desactivando aleatoriamente neuronas en una capa FC estamos obligando el modelo a que tenga en cuenta más características a la hora de tomar su decisión, evitando que solo se fije en unas pocas que pueden ser válidas para ciertos datos de entrada pero no representan una solución general. El hecho de que un modelo se fije en características particulares a ciertos datos para obtener sus predicciones se conoce como sobreajuste o **overfitting**. Debido a como son los

tipos de datos para la segmentación musical el overfitting es común a la hora de entrenar modelos, por ello y para poder obtener resultados fiables se emplea la técnica dropout. Esta técnica solo es utilizada durante el entrenamiento, cuando se realizan pruebas con el modelo se deja que éste emplee el total de sus neuronas.

Si nos fijamos en la arquitectura vemos que tras la segunda capa convolucional no existe una capa de agrupación, se realizó también un estudio incluyendo otra capa Max Pooling tras la segunda capa convolucional como en la topología descrita en [14] sin obtener mejores resultados, lo que se concluye es que resulta contraproducente reducir la cantidad de información tras la segunda convolución, siendo los mapas de características suficientemente robustos para tomar una decisión tras esta capa. Otra cuestión relevante es si nos fijamos en las dimensiones de los kernels de las capas convolucionales vemos que éstos son mayores en la primera dimensión respecto a la segunda. Posteriormente se explicará con mayor detalle pero para una mejor comprensión ahora se puede indicar que la dimensión primera corresponde con valores frecuenciales en los datos de entrada mientras que la segunda se corresponde a información temporal. La conclusión sacada, también descrita en [19] y en [13], es que **la red convolucional es más sensible a información temporal que frecuencial** de aquí que se busque tamaños de kernel no simétricos, con este concepto también se explica el por qué de la dimensión del kernel de la capa de agrupación ya que se busca reducir más la información frecuencial que la temporal.

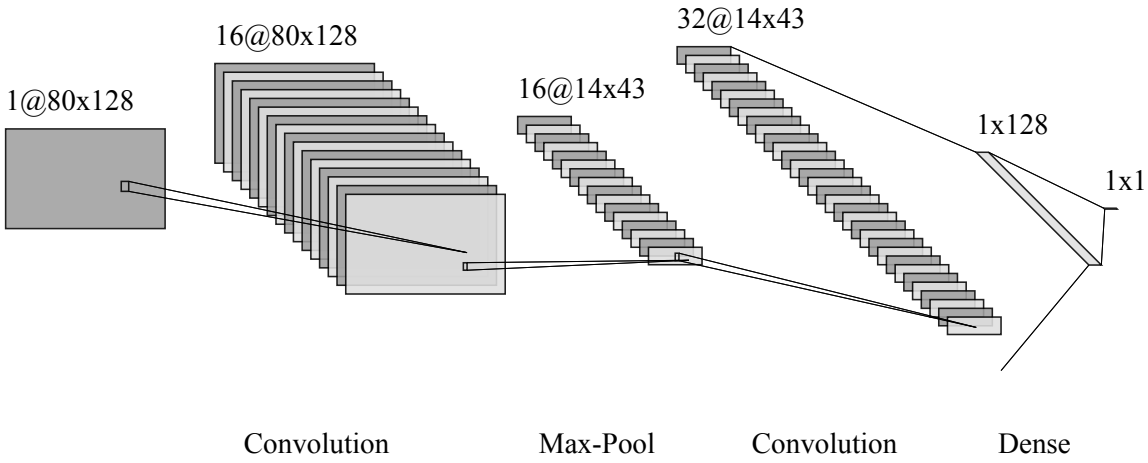


Figura 3.5: Visualización de la topología del modelo para un dato de entrada de dimensiones (80,128,1).

Por último las activaciones en todas las capas menos en la última FC son del tipo ReLU (*Rectified Linear Unit*), sin entrar mucho en detalle porque no es necesario para la comprensión de este trabajo, las activaciones son una parte de todas las neuronas en una red deep learning (*todas las neuronas hacen al menos dos operaciones básicas: procesamiento de datos de entrada + activación*), su función es añadir no linealidad a la salida de cada una de las neuronas en una red. Gracias a ello diferentes neuronas en una misma capa son capaces de procesar funciones distintas, si no añadiésemos estas activaciones las neuronas tendrían un funcionamiento lineal y se podría resumir la complejidad de toda una red en una sola neurona ya que podríamos obtener una ecuación lineal que se aproximase al funcionamiento de todas las neuronas en conjunto. En la última capa de la red neuronal (última capa FC) se incluye una activación de función sigmoide, este tipo de activación es común en esta capa cuando se trabaja en un caso de clasificación binaria.

## 3.2. Dataset

El banco de datos que se dispone es el ofrecido por el grupo GTAC de iTEAM UPV. Está compuesto por un total de 146 canciones pertenecientes al género EDM además de las respectivas anotaciones de los eventos en las canciones. El género EDM presenta una diversidad de formas y estructuras en las que se construyen sus canciones, nuestra biblioteca de canciones tiene una variedad de subgéneros dentro del EDM, Siendo muy distintos en algunos casos un subgénero de otro. Las anotaciones realizadas por un profesional tienen la forma y estructura de la Figura 3.6, se han anotado el frame o muestra dónde ocurre cierto evento, las canciones utilizadas están en formato mp3 y sampleadas a 44100Hz.

| **Frontera** | **Frame** |
|---|---|
| VoiceIn | 22528 |
| Layer | 589824 |
| BreakIn | 691712 |
| Layer | 1345536 |
| Layer | 1554944 |
| Layer | 1693184 |
| Instant | 1977344 |
| Layer | 2344448 |
| Instant | 2324992 |
| Layer | 2705408 |
| Layer | 2873344 |
| Instant | 2954752 |
| BreakOut | 3013632 |
| VoiceOut | 3027968 |

Figura 3.6: Ejemplo de las anotaciones hechas para la canción *Summer Air* del artista Hardwell.

Los diferentes segmentos que ocurren en una canción se han anotado de la siguiente forma:

- VoiceIn/VoiceOut: Inicio y fin de la voz.
- BreakIn/BreakOut: Inicio y fin de la parte con más energía de la canción, llamado break y típico de canciones EDM.
- Layer. Se anota cuando se introduce un nuevo instrumento melódico o percusivo a la canción que añade una nueva *capa* a la melodía.
- Instant. Detalle que ocurre en la canción por un periodo breve y que se utiliza para adornar ciertos momentos de la canción.

Las etiquetas “Instant” no han sido utilizadas en este trabajo ya que no aportan información útil para la segmentación musical debido a que no representan el inicio o fin de cierta estructura.

## 3.3. Extracción Características Acústicas

La extracción de características acústicas de las canciones se ha realizado mediante el uso de los paquetes de la famosa librería para Python *librosa* en su versión 0.8 [24]. Ésta ofrece todos los bloques y funciones necesarias para realizar tareas relacionadas con la extracción de información musical (*Music Information Retrieval*). El proceso desarrollado en este trabajo ha sido inspirado en [19, 14].

Para cada archivo de audio se ha realizado la obtención de su espectrograma de Mel mediante la realización de la FFT (*Fast Fourier Transform*) con una ventana de 46ms (2048 muestras a 44.1KHz) y con un 50 % de sobrelapamiento entre ellas (salto de 1024 muestras entre FFTs), a esta transformación se ha aplicado un banco de filtros mel desde 80Hz hasta 12KHz. La resolución del espectrograma se ha restringido a 80 unidades o compartimentos (del inglés: *80 mel-frequency bins*). El espectrograma mel es convertido posteriormente a escala logarítmica y normalizado a 0 dB.

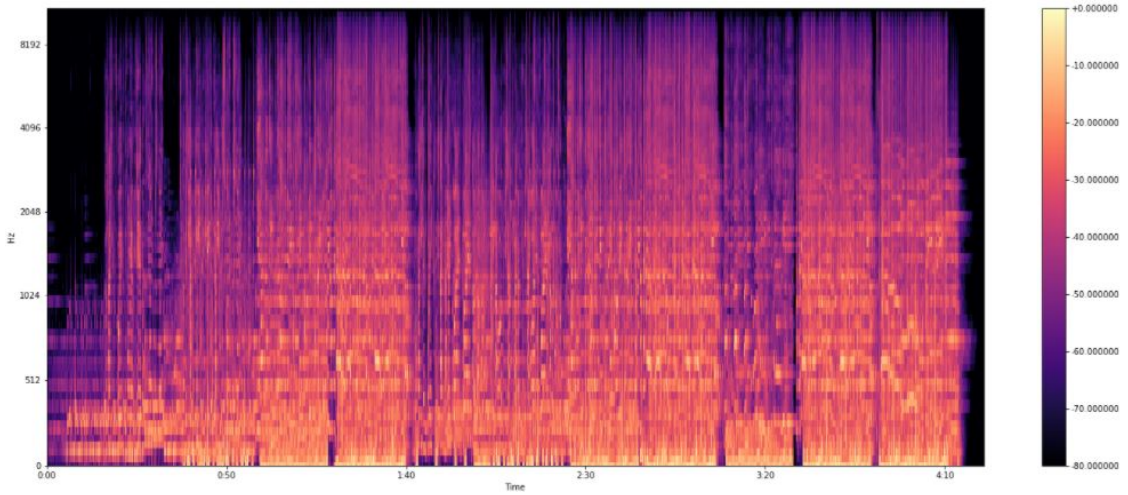


Figura 3.7: Espectrograma mel normalizado de la canción *Bad Liar* de Imagine Dragons

Como queremos estudiar la canción en cada muestra del espectrograma, para cada una se va a obtener una extracción del espectro. A cada muestra se añadirá un **contexto** consistente en las muestras vecinas, esto sirve para proporcionar más información acerca de su situación pasada y futura (muestras del espectrograma anteriores y posteriores). La idea es que cada dato a utilizar en la red neuronal es una "*imagen*" extraída del espectrograma, con la muestra a analizar en el centro de la imagen y su contexto a los lados, La cantidad de contexto a añadir es un **hiperparámetro** a estudiar por lo que se probarán con distintos valores: 4,8,16 y 32 segundos de contexto basándose en [19]. Para las muestras al inicio y final de una pieza musical se ha añadido muestras de relleno (*padding*) para poder dotarlas de contexto. Este relleno consiste en un espectrograma a -120dB (silencio), se ha decidido por este tipo de relleno debido a su fácil implementación. En [19] se ha utiizado como relleno ruido rosa a -70dB para parecerse al fondo que existe en las canciones y así evitar problemas durante el entrenamiento del modelo pero con un relleno de silencio la red no ha tenido problemas a la hora de clasificar estas muestras.

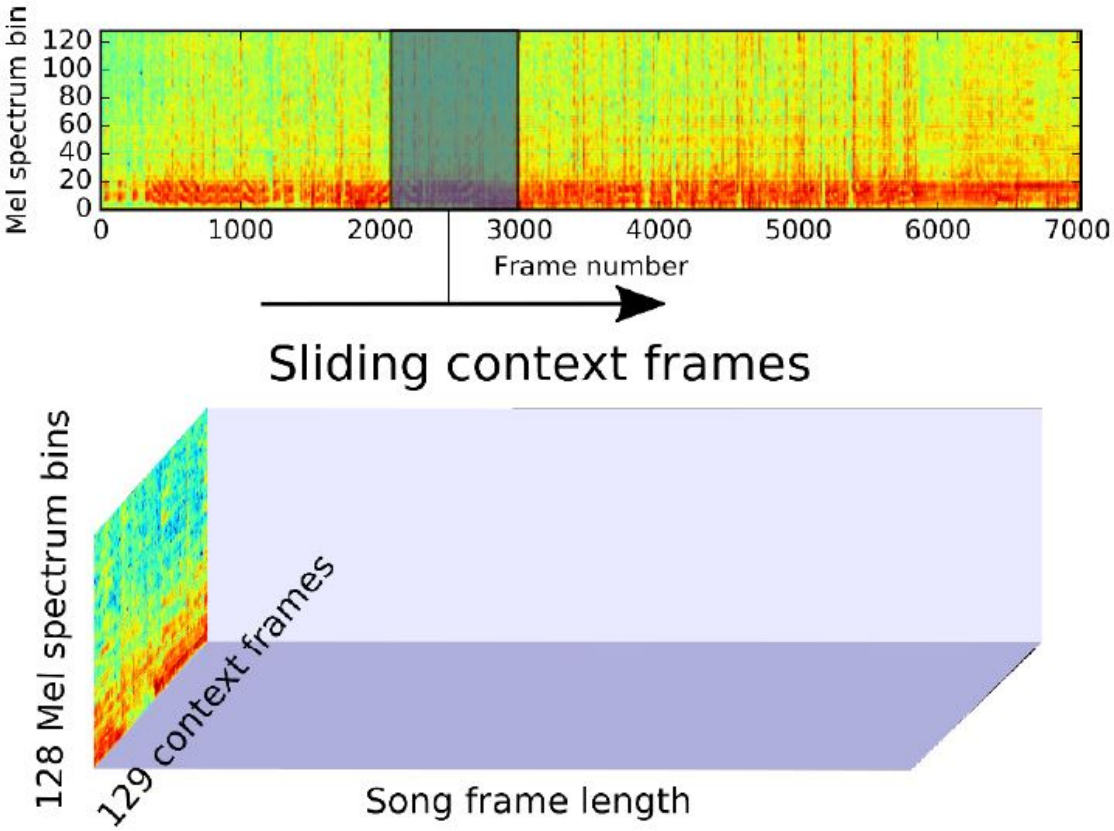


Figura 3.8: Ilustración de la obtención de extracciones del espectrograma para una canción. Este ejemplo obtenido de [23] utiliza 3 segundos de contexto resultando en extracciones de dimensiones 129 frames temporales por 128 frames frecuenciales.

El espectrograma mel normalizado de una canción de unos 3 minutos contiene más de 7000 muestras, si por cada muestra hacemos una extracción del espectrograma con su contexto nos encontramos con un banco de datos de 7000 imágenes de dimensiones (*tamaño contexto + 1, 80 unidades mel*). Esto va a suponer un problema a la hora de querer entrenar el modelo, ya que los datasets creados cuando se utilicen múltiples canciones serán gigantescos ocupando demasiado espacio en la memoria y ralentizando todo el proceso o volviéndolo imposible de poner en funcionamiento. Para ello se propone la solución desarrollada en [19] consistente en reducir el número total de muestras en un espectrograma (**subsampling**). Se ha obtenido el máximo por cada 3, 6 y 12 muestras adyacentes, sin sobrelapamiento, resultando en espectrogramas de distintas resoluciones, cada tipo de espectrograma resultante se ha agrupado en los datasets llamados ***High, Mid y Low*** correspondientes con el número de muestras reducidas. La cantidad de muestras por segundo para cada dataset es: 14,35fps, 7,18fps y 3,59fps. Esto nos servirá para poder convertir a su resolución original los resultados obtenidos y así poder analizarlos correctamente.

## 3.4. Preprocesado Etiquetas

Para poder utilizar las etiquetas como datos de entrada en la red neuronal se requiere de un preprocesado previo. A cada extracción del espectrograma hecha en una canción le corresponde una etiqueta. Esta etiqueta indica si existe una frontera en esa extracción, indicado con un 1, o si la imagen no presenta éste evento, indicado por un valor cero. Para ello crearemos para cada canción un array del mismo número de muestras que el generado en la extracción de características acústicas , de esta manera tendremos pares de datos (*Extracción espectrograma, etiqueta*) en el que la etiqueta es el ítem del array dicho anteriormente correspondiente a la extracción que indica si existe en ésta un evento de frontera o no.

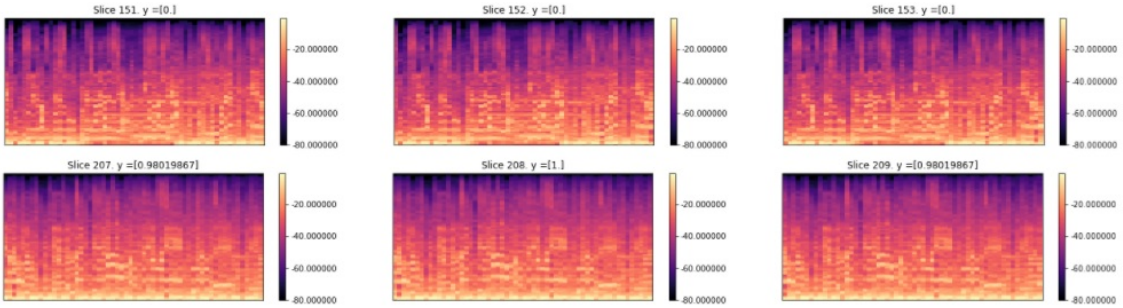


Figura 3.9: Ejemplo de pares de datos (Extracción, etiqueta), el valor de las etiquetas está indicado en el título de cada extracción. Las imágenes de la parte superior corresponden a extracciones de la clase no evento y las inferiores a la clase que presenta evento de frontera en su imagen.

El principal problema que existe es que si vemos el número de extracciones que contienen un evento respecto al total el porcentaje es muy bajo. De manera general para todas las canciones utilizadas el porcentaje de extracciones que muestran fronteras es en torno al 5 % o menor. Esto es normal ya que en una canción no existen muchas transiciones y aunque añadiésemos más canciones al dataset este porcentaje continuará parecido. Un banco de datos que contiene dos clases (extracciones de espectrograma con evento de frontera y no evento) en la que una clase es más representada que la otra se conoce como **datos desiquilibrados** (del inglés: *Unbalanced Data*). En deeplearning esto es un problema a la hora de entrenar una red con un dataset de este tipo. El modelo tenderá a omitir la clase menos representada ( si el caso es exagerado como es el nuestro, en el que la representación de una clase es del 5 % del total de ejemplos) y aprenderá a clasificar la mayoría de muestras como la clase superior. Esto tiene sentido ya que un modelo que predice que no existe evento de frontera para todas las muestras en una canción obtendría una precisión muy alta (95 % del total de extracciones estarían predichas correctamente). Por tanto el modelo no sería capaz de aprender a detectar esta clase menos representada que es en realidad nuestro objetivo.

Para solucionar este problema y aumentar la representación de la clase que nos interesa se implementa la solución desarrollada en [19] conocida como **target smearing**. La idea consiste en *manchar* las muestras cercanas a una extracción que presente frontera, convirtiéndolas en muestras positivas. Esto tiene sentido ya que las extracciones cercanas a aquélla que presenta frontera también presenta este evento, pero no se encuentra centrado en la imagen. Aún así podemos categorizar esta muestras como de la clase frontera. Para ello lo que se ha realizado es convolucionar los arrays con etiquetas con un kernel gaussiano de cierto tamaño. **El ancho del filtro gaussiano es otro hiperparámetro**, se utilizarán los siguientes anchos: 1, 1.5 y 3 segundos.

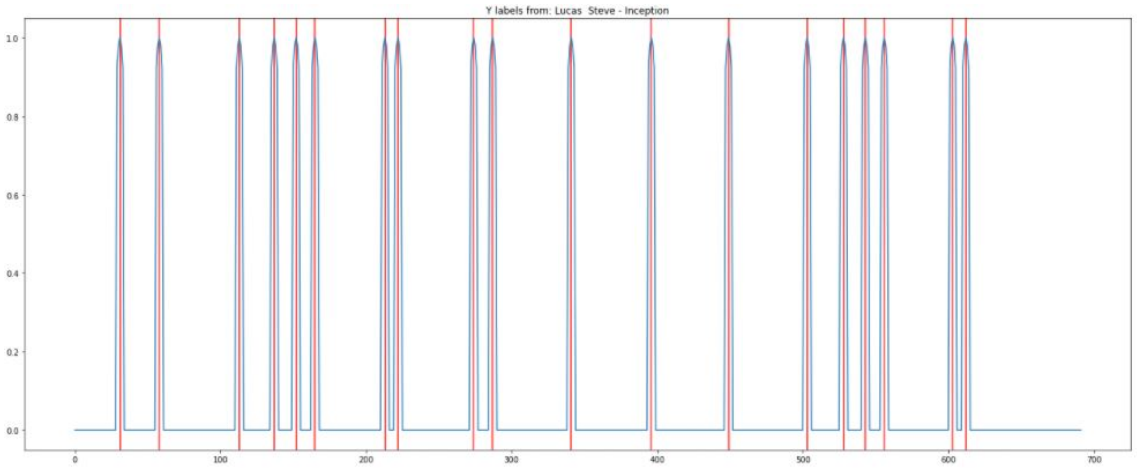


Figura 3.10: Etiquetas de la canción *Inception* de Lucas Steve. Las líneas rojas indican los eventos de fronteras obtenidos de las anotaciones, la línea azul son las nuevas etiquetas convolucionadas con un kernel gaussiano de 1,5 segundos.

Como se puede observar en la Figura 3.10, las nuevas etiquetas presentan máximos dónde ocurren exactamente los eventos mientras que los frames vecinos se convierten en positivos pero con valores menores. De esta manera aumentamos el número de etiquetas positivas en torno a un 15-30 % dependiendo del ancho del kernel gaussiano y de la resolución de los espectrogramas pero resulta aún insuficiente. Para poder equilibrar aún más las clases, durante el entrenamiento del modelo se aumentará la probabilidad de que se utilice aleatoriamente extracciones con etiquetas positivas por un factor entre 2 y 3. Esta técnica también es utilizada en [19, 23] y se consigue añadiendo aleatoriamente extracciones positivas dentro del dataset del entrenamiento. Es preciso decir que todo este tratamiento de las etiquetas para aumentar el número de muestras positivas **solo es utilizado durante el entrenamiento del modelo**. A la hora de evaluar y testear se utilizarán las etiquetas obtenidas de las anotaciones, lo que sería las líneas rojas en la Figura 3.10.

## 3.5. Entrenamiento

Inicialmente se ha creado un entorno para el entrenamiento que utilice la GPU en vez de la CPU para realizar el procesamiento. Esto consigue acelerar la velocidad de computación hasta 17 veces más. Para ello se ha utilizado Anaconda[25], un entorno especializado en *big data science* que permite utilizar muchos de los paquetes y herramientas necesarios para deep learning. Todo el programa y funciones han sido desarrollado en Python 3.8 utilizando como IDE el software Jupyter[26], bastante común en trabajos de machine learning. La computación en GPU se ha conseguido con el uso de la librería tensorflow-gpu 2.4 junto con el uso del software CUDA 11.2 de Nvidia. La tarjeta gráfica utilizada es una Nvidia GTX 1070 que nos ha permitido entrenamientos bajo tiempos razonables dada la extensión de este tipo de trabajo. Para la implementación, entrenamiento y evaluación de la red neuronal se ha utilizado el backend de Keras[27].

En cada entrenamiento se ha creado previamente un dataset compuesto por las extracciones del espectrograma y las etiquetas, seleccionando el tipo de resolución del espectrograma, el tamaño del contexto y el tamaño del kernel gaussiano (*target smearing*). Se ha realizado un estudio para optimizar los parámetros del entrenamiento, se ha buscado conseguir un compromiso entre el ratio de aprendizaje (*learning rate*) y el número de generaciones entrenadas de manera que no demorase mucho el entrenamiento. Como función de pérdidas se ha utilizado la función *binary crossentropy* utilizada principalmente en tareas de clasificación binaria con un optimizador SGD (*Stochastic Gradient Descent*) cuyo ratio de aprendizaje es de 0.05 con un momento de 0.45. El número total de generaciones ó epochs entrenadas por cada modelo ha sido de 20. Con estos parámetros se han conseguido tiempos de entrenamiento no superiores a la media hora, lo cual ha resultado adecuado para realizar distintas combinaciones de los hiperparámetros indicados anteriormente (*resolución, contexto, target smearing*).

Los resultados obtenidos en este trabajo se han conseguido entrenando modelos con 116 canciones del banco de datos disponible, dejando 10 canciones para evaluar el modelo durante el entrenamiento y las 20 restantes para realizar los test y obtener resultados. Estas 30 canciones no han sido utilizadas para entrenar el modelo, por tanto éste no ha aprendido de ellas, lo cual nos sirve para ver y comprobar la efectividad del entrenamiento y cuán robusto es el modelo a la hora de analizar canciones que no ha visto anteriormente. La red ha sido alimentada mediante mini-lotes ó *mini-batches* de 64 muestras cada uno. En vez de dar a la red todas los ejemplos disponibles a la vez, la red ha sido entrenada poco a poco introduciendo estos lotes, de esta manera conseguimos tiempos de entrenamiento mucho más rápidos ya que los pesos del modelo son actualizados por cada lote en vez de por cada generación si utilizamos todas las muestras a la vez. También se hizo un estudio del tamaño de estos lotes concluyendo que 64 es un valor adecuado, un tamaño menor resulta en insuficientes pruebas por lote y un tamaño mayor ralentizaba el aprendizaje teniendo que entrenar por más generaciones. Además tras todas las capas convolucionales se ha incluido *batch normalization* ó normalización de los lotes. Esta técnica consiste en normalizar el grupo de datos de entrada para hacerlos más consistentes, reescalando y distribuyendo uniformemente la distribución de datos, y por tanto haciendo que la red trabaje de manera más rápida. El tipo de normalización utilizada es la llamada normalización L2, bastante utilizada en machine learning.

# Capítulo 4

# Evaluación y Resultados

En este capítulo se describirá el procesado necesario que se ha realizado sobre las salidas del modelo para su correcta evaluación así como los resultados obtenidos. El modelo de evaluación es inferido del propuesto por la organización MIREX[21] para la segmentación musical. También se mostrarán los resultados de otros trabajos aceptados y presentados por MIREX para poder realizar una comparación, no tan fidedigna como se quisiese ya que este trabajo ha utilizado un banco de datos distinto, pero que sirva al menos de referencia para la evaluación de éste. Los resultados se han obtenido con un set de 20 canciones de nuestro banco de datos que no han sido utilizadas para el entrenamiento, por tanto hablamos de que el modelo *predice* en que muestras existe un evento de frontera a partir de lo aprendido en el set de entrenamiento.

## 4.1. Evaluación

Las predicciones que obtenemos con un modelo entrenado requieren de cierto procesamiento para ultimar la decisión tomada por él. Éstas no muestran una clasificación perfecta si no más bien muestran cierta ambigüedad, a veces las fronteras que han sido detectadas se muestran con predicciones dispersas alrededor de las muestras cercanas y no todas han sido predichas con el mismo nivel de verdad (del inglés: *ground-truth level*) por lo que tenemos que emplear técnicas que nos permitan decidir que valores son los que realmente predice y obtiene el modelo.

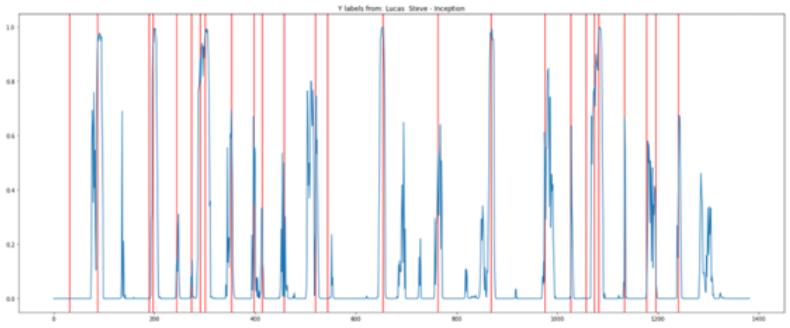

Figura 4.1: Ejemplo de las predicciones realizadas por un modelo (línea azul) sin realizar ningún tipo de procesado.

La primera técnica que se emplea es la conocida como **bagging**, consiste en utilizar varios modelos entrenados sobre el mismo dataset con los mismos hiperparámetros y realizar la media sobre sus predicciones. De esta manera reducimos la posible dispersión de valores y variabilidad existente en cada modelo particular. Se han entrenado para cada dataset cinco modelos y se ha obtenido una salida mas suavizada que nos facilita el posterior procesado y nos mejora los resultados obtenidos.

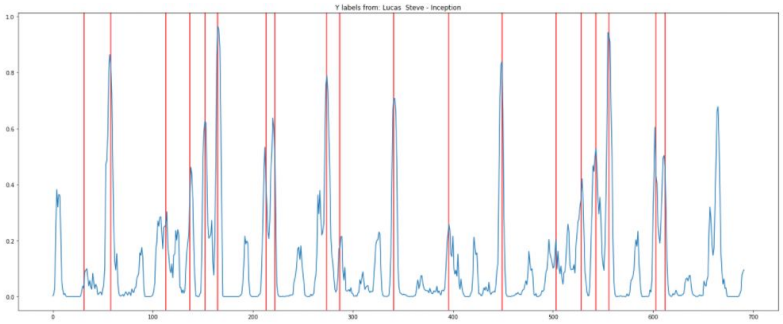


Figura 4.2: Predicciones obtenidas tras hacer bagging de 5 modelos.

### 4.1.1. Peak-Picking

Para poder comparar las predicciones de los segmentos con las anotaciones que tenemos de base en el banco de datos tenemos que obtener los máximos locales de las predicciones (Figura 4.1, Figura 4.2) . Para ello se han obtenido el máximo sobre muestras contiguas en un total de un segundo (el número de muestras dependerá de la resolución del espectrograma). Este valor ha sido escogido basándonos en los anchos de los picos en los resultados obtenidos por los modelos y nos parece razonable ya que las etiquetas utilizadas durante el entrenamiento presentan muestras positivas contiguas en mínimo un segundo debido al target smearing.

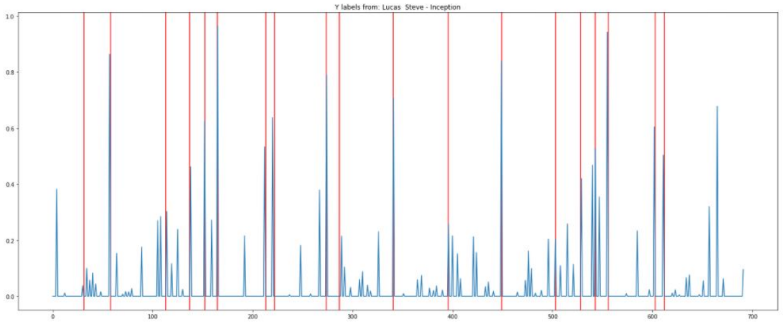


Figura 4.3: Predicciones de la Figura 4.2 tras realizar peak-picking

### 4.1.2. F-score

La evaluación de la segmentación realizada por el modelo presenta un problema. Debido a que la mayoría de muestras son negativas (no presentan frontera) la precisión, entendida como el número de muestras predichas correctamente sobre el total de muestras, no es una medida útil en

nuestro estudio. Se ha seguido el método de evaluación propuesto por MIREX para la segmentación musical y que se puede ver en [19], consistente en el uso de la medida f-score como valor que realmente nos indica cuán correctamente predice nuestro modelo.

En el análisis estadístico de una tarea de clasificación binaria, el f-score es obtenido a partir de los valores de *recall* y *precision*. Estos términos, no traducidos del inglés, representan:

- **Recall**: Porcentaje que muestra cuántos elementos relevantes se han sustraido del total de elementos relevantes
- **Precision**: Fracción que indica cuántos elementos son relevantes respecto al total de elementos sustraídos.

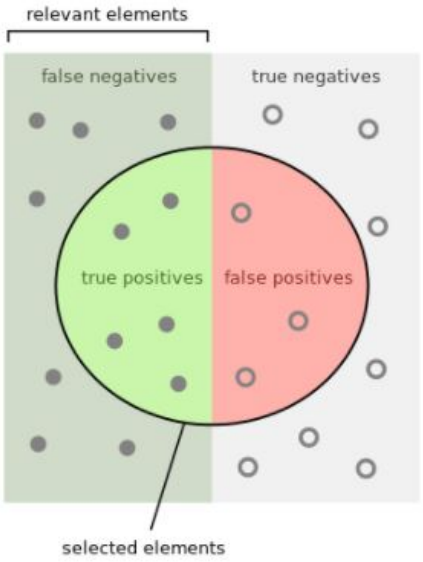


Figura 4.4: Clasificación de todos los posibles elementos.Para nuestro caso los falsos negativos son aquellas fronteras no detectadas. Verdaderos positivos son las fronteras detectadas. Falsos positivos son las predicciones incorrectas. Verdaderos negativos son las muestras predichas correctamente como no frontera.

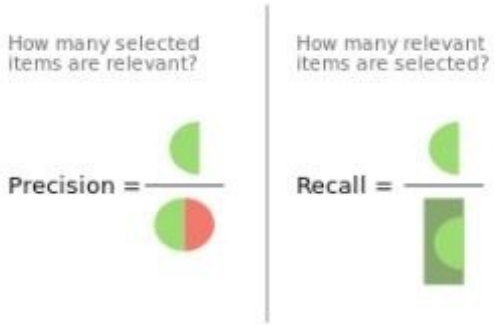


Figura 4.5: Precision y recall basándose en la Figura 4.4. La precisión en nuestro caso es el número de predicciones correctas sobre el total de predicciones. El recall sería respecto al total de fronteras cuántas han sido predichas.

El f-score es la media armónica entre recall y precision. Y es obtenido mediante la siguiente fórmula:

$$F_1 = 2 * \frac{recall * precision}{recall + precision} = \frac{tp}{tp + \frac{1}{2}(fp + fn)} \tag{4.1}$$

*dónde tp:true positive;fp: false positive;fn: false negative*

### 4.1.3. Selección umbral de verdad

Con el objetivo de obtener los mejores resultados es necesario establecer cierto umbral de verdad sobre las predicciones seleccionadas tras el *peak-picking*. Para ello se ha desarrollado un algoritmo que evalue el f-score, recall y precision obtenido por el modelo para cada umbral y obtenga aquel óptimo que obtiene mejor f-score.

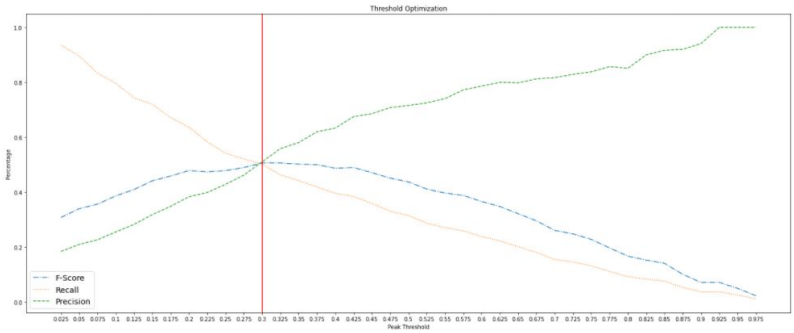


Figura 4.6: Gráfica que representa los distintos valores de f-score,recall y precision según el umbral de verdad seleccionado. La línea roja indica el umbral óptimo calculado.

Como se puede observar en la Figura 4.6 si seleccionamos un umbral bajo obtenemos un mayor recall, lo que quiere decir que muchas de las fronteras han sido detectadas pero a costa de un valor de precision bajo. Si en contra seleccionamos un umbral mucho más alto, la precisión es muy alta, lo que nos dice que la mayoría de predicciones son correctas pero a costa de un bajo recall (no se han predicho muchas de las fronteras). Al final el f-score es el valor que nos dice bajo que umbral se han obtenido los mejores resultados.

## 4.2. Resultados

Siguiendo los pasos de evaluación dictados por el MIREX se han obtenido resultados dejando cierta tolerancia, Como se dijo al principio de este trabajo, es muy difícil que dos profesionales de la música realizan anotaciones exactas sobre una misma canción. Con los modelos entrenados tenemos la misma situación, va a ser complicado que el modelo detecte las fronteras en la misma muestra que la anotación. Por ello vamos a dar válidos fronteras predichas bajo cierta tolerancia. Se han obtenido resultados bajo dos tolerancias siguiendo el procedimiento del MIREX: 3 segundos y 1 segundo de tolerancia. Los resultados mostrados en Tabla 4.1 y Tabla 4.2 se han obtenido haciendo bagging de 5 modelos para cada tipo de dataset. Los valores de contexto y target smearing se encuentran en segundos.

| Resolución | Contexto | Smearing | F-score | Recall | Precision |
|---|---|---|---|---|---|
| LOW | 16 | 1.5 | 0.5345 | 0.5016 | 0.5796 |
| LOW | 16 | 1 | 0.5167 | 0.5625 | 0.4793 |
| LOW | 32 | 1.5 | 0.5155 | 0.6157 | 0.4433 |
| LOW | 32 | 3 | 0.5134 | 0.5995 | 0.4489 |
| HIGH | 4 | 1.5 | 0.5108 | 0.5630 | 0.4812 |
| LOW | 16 | 3 | 0.5020 | 0.4851 | 0.5331 |
| MID | 16 | 1 | 0.4978 | 0.5244 | 0.4738 |
| MID | 8 | 1,5 | 0.4972 | 0.4502 | 0.5386 |
| HIGH | 4 | 3 | 0.4874 | 0.5794 | 0.4115 |
| MID | 16 | 1,5 | 0.4691 | 0.4661 | 0.4557 |
| HIGH | 8 | 1,5 | 0.4545 | 0.6250 | 0.3571 |
| MID | 8 | 3 | 0.4402 | 0.4311 | 0.4570 |
| HIGH | 8 | 1 | 0.4111 | 0.3935 | 0.4304 |

Tabla 4.1: Tabla resultados con TOL = 3s

| Resolución | Contexto | Smearing | F-score | Recall | Precision |
|---|---|---|---|---|---|
| HIGH | 4 | 1.5 | 0.3284 | 0.3451 | 0.3112 |
| MID | 8 | 1.5 | 0.3157 | 0.2907 | 0.0.3415 |
| HIGH | 4 | 3 | 0.3044 | 0.2675 | 0.3618 |
| LOW | 16 | 1 | 0.2989 | 0.2593 | 0.3314 |
| MID | 16 | 1 | 0.2984 | 0.2343 | 0.4106 |
| HIGH | 8 | 1,5 | 0.2961 | 0.2631 | 0.3373 |
| MID | 16 | 1,5 | 0.2982 | 0.2553 | 0.3488 |
| MID | 8 | 3 | 0.2887 | 0.2509 | 0.3428 |
| LOW | 16 | 1,5 | 0.2834 | 0.2801 | 0.3312 |
| HIGH | 8 | 1 | 0.2709 | 0.2593 | 0.2835 |
| LOW | 32 | 3 | 0.2647 | 0.2454 | 0.2873 |
| LOW | 32 | 1.5 | 0.2593 | 0.2755 | 0.2449 |
| LOW | 16 | 3 | 0.2451 | 0.2460 | 0.2582 |

Tabla 4.2: Tabla resultados con TOL = 1s

Para una mejor comprensión de los resultados obtenidos se incluye el mejor modelo entrenado dentro de otros modelos entrenados para segmentación musical realizas por las varias campañas MIREX (Tabla 4.3).

| **Algoritmo** | **F-score** | **Recall** | **Precision** |
|---|---|---|---|
| KTC1(2014) | 0.6164 | 0.7059 | 0.5944 |
| KTC2(2014) | 0.5726 | 0.6675 | 0.5648 |
| **LOW_16_1.5** | 0.5345 | 0.5016 | 0.5796 |
| MP2(2013) | 0.5213 | 0.6443 | 0.4793 |
| MP1(2013) | 0.5188 | 0.5849 | 0.5040 |
| **LOW_16_1** | 0.5167 | 0.5625 | 0.4793 |
| CF5(2013) | 0.5052 | 0.7862 | 0.3990 |
| SMGA1(2012) | 0.4985 | 0.7258 | 0.4021 |
| RBH1(2013) | 0.4920 | 0.7482 | 0.3922 |
| SP1(2012) | 0.4891 | 0.7842 | 0.3854 |
| KSP1(2012) | 0.4888 | 0.7838 | 0.3850 |

Tabla 4.3: Resultados segmentación musical con deep learning con tolerancia de 3 segundos. Nuestros resultados son los resaltados y son comparados con resultados de la campaña MIREX para esta tarea.

Los resultados obtenidos con la resolución LOW han resultado ser los mejores para una tolerancia de 3 segundos, concluyendo al igual que en [19] que si mantenemos suficiente información temporal (*contexto*) podemos reducir significativamente la información frecuencial sin que esto afecte al modelo. Los modelos que con más precisión han detectado fronteras han sido los que tienen mayor resolución como se puede ver en la Tabla 4.2. Respecto al *smearing* de las etiquetas positivas, resulta que con anchos de 1,5 segundos se suelen obtener los mejores resultados, empeorando ligeramente al aumentar o disminuir el tamaño del kernel.

# Capítulo 5

# Conclusión y discusión de resultados

El desarrollo de este trabajo ha tenido una dinámica cíclica, primero se han entrenado modelos bajo ciertos parámetros, después se han evaluado los resultados y comparado con otros obtenidos para finalmente llegar a una conclusión que nos orienta para el entrenamiento de los siguientes modelos. Este proceso es común en machine learning ya que nosotros introducimos los datos al modelo y vemos los resultados, dejando a éste todo el proceso de optimización de la función de error. Nosotros tenemos que encontrar que características de los datos son las más utiles para la tarea que estamos presentando al modelo y también descubrir que partes de la estructura del modelo son más críticas y ver si requieren de mayor complejidad o no.

Una de las primeras conclusiones obtenidas es respecto a las características acústicas utilizadas, los espectrogramas mel. Han resultado ser muy buenos **descriptores rítmicos**, Las fronteras predichas con mayor seguridad por el modelo (mayor nivel de verdad) han sido aquellas que presentan un cambio rítmico fuerte (*empieza o acaba la batería, existe un cambio rítmico notable entre una estructura u otra...*). Por contra también se ha observado que todos los modelos han presentado problemas a la hora de detectar ciertos eventos melódicos, esto es debido también en parte a los espectrogramas mel. Éstos no son capaces de registrar tan bien cambios melódicos en tiempos largos, como puede ser el cambio de los acordes de fondo en una canción. Esto ocurre porque estos cambios vistos en el espectrograma ocurren en un rango de frecuencias más pequeño que los rítmicos y por tanto tienen una menor representación en el espectro.

Otra cuestión relevante es respecto a las diferencias existentes entre generos EDM. La base de datos utilizada no representa un género en particular si no más bien una diversidad de subgéneros y canciones que pueden ser asociadas al EDM. Esto implica que no existe una estructura general para todas las canciones, caso que si se da en otros géneros musicales. Por ejemplo, el evento conocido como *break in*, que es dónde la canción aporta la máxima energía, no se realiza de la misma manera en todos los subgéneros. En unos puede ser cuando todas los sonidos presentes se utilizan a la vez, en otros puede ser identificado por una composición rítmica particular o también se destaca por un refuerzo del *lead* o de la voz del cantante. Esto ha producido en los resultados una disparidad de predicciones según el tipo de canción. Teniendo canciones que sus fronteras han sido predichas casi a la totalidad mientras en otras sus resultados o bien presentan mucha ambigüedad, teniendo muchas predicciones erróneas o con poco nivel de verdad, o incluso no detectando casi ninguna frontera. En trabajos futuros puede ser interesante realizar un cambio en la distribución de las bases de datos, seleccionando específicamente canciones dentro de un subgénero.

Hubiese sido interesante obtener los modelos entrenados por otros investigadores en esta tarea para poder obtener una mejor comparación. Realizar transfer learning de estos modelos hubiese servido para ver cuán bueno son esos modelos con la base de datos que se ha dispuesto para este trabajo y así obtener una comparación más real.

Como conclusión decir que las redes convolucionales son buenos modelos para la identificación de estructuras musicales en espectogramas, debido a su similitud con la segmentación de imágenes, existe potencial para utilizar estas redes como herramientas para la extracción de información musical. Para poder hacer una descripción más general de todos los eventos en el género EDM los modelos precisan de un mayor volumen de información así como de procesamiento al utilizado en este trabajo. Por un lado más información ya que los espectrogramas mel, aunque muy buenos en ciertas tareas del análisis musical, parece que no son completamente suficientes para ésta. Esta falta de caracterización de cambios melódicos hace que el modelo sea menos robusto frente a estos eventos, una posible solución es la utilización conjunta de otras características acústicas junto con los espectrogramas mel, como las matrices SDM o vectores chroma. Una idea podría ser obtener un espectrograma mel con un banco de filtros menor en un rango de frecuencias en torno a [500-2000]Hz que se añada al realizado en este trabajo, de esta manera obtendríamos una mayor resolución en dónde ocurren los cambios melódicos. También indico que el modelo utilizado precisa de mayor capacidad computacional, un añadido de capas convolucionales ofrecería mejores mapas de características al modelo, pudiendo refinar aquellas partes dónde no es tan resolutivo actualmente. Los experimentos realizados con modelos de mayores capas convolucionales han dado siempre mejores resultados que sus hermanos. Ésto no indica que tenemos un rango infinito de mejora a mayor aumentemos la complejidad del modelo pero sí que éste no ha alcanzado su complejidad máxima para ésta tarea. Una posible manera de proceder para aumentar el número de capas convolucionales sería observar los mapas de características a cada salida de cada capa convolucional, de esta manera identificaríamos que características son extraídas en cada convolución para orientarnos a la hora de aumentar la estructura de la red neuronal, pudiendo concretar más la acción de estas capas en la segmentación musical.

.

# Bibliografía


[1] *Grupo de Tratamiento de Audio y Comunicaciones, iTEAM UPV*. http://www.iteam.upv.es/?group=gtaclang=es.

[2] *iTEAM UPV*. http://www.iteam.upv.es/?lang=es.

[3] Pablo Gutierrez-Parera. Carlos Hernández Jose Javier López. «Low Cost Algorithm for online segmentation of Electronic Dance Music». En: $144^{th}$ *Convention of Audio, Milan* 113 (2018).

[4] Anssi Klapuri Jouni Paulus. «Music Structure Analysis Using a Probabilistic Fitness Measure and an Integrated Musicological Model». En: *ISMIR - Session 3b - Computational Musicology* (2008).

[5] Anssi Klapuri Jouni Paulus. «Acoustic Features for Music Piece Structure Analysis». En: *11th International Conference on Digital Audio Effects (Espoo, Finland)* (2008).

[6] Mark Sandler Jean-Julien Aucouturier. «Segmentation of Musical Signals Using Hidden Markov Models». En: *110th Convention Audio Engineering Society (Amsterdam, Netherlands)* (2001).

[7] Hong-Jiang Zhang Lie Lu Muyuan Wang. «Repeating Pattern Discovery and Structure Analysis from Acoustic Music Data». En: *Proceedings of the 6th ACM SIGMM international workshop on Multimedia information retrieval* (2004), págs. 275-282.

[8] Yoshua Bengio Yann LeCun Leon Bottou. «Gradient-Based Learning Applied to Document Recognition ». En: *Proceedings of the IEEE* 86.11 (1998), págs. 2278-2324. DOI: 10.1109/5.726791.

[9] Erik Learned-Miller Huaizu Jiang. «Face Detection with the Faster R-CNN». En: *Proceedings of the 12th IEEE International Conference on Automatic Face & Gesture Recognition (FG)* (2017), págs. 650-657. DOI: 10.1109/FG.2017.82.

[10] Andrew Zisserman Karen Simonyan. «Very Deep Convolutional Networks for Large-Scale Image Recognition ». En: *CoRR* abs/1409.1556 (2015).

[11] Tom M. Mitchell. *Machine Learning*. McGraw-Hill Education, 1997. ISBN: 0070428077.

[12] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn, Keras and Tensorflow*. O'REILLY, 2020. ISBN: 1492032649.

[13] Yue Zhao Zain Nasrullah. «Music Artist Classification with Convolutional Recurrent Neural Networks ». En: *Proceedings of the International Joint Conference on Neural Networks (IJCNN)* (2019).

[14] Sebastian Böck Jan Schlüter. «Improved Musical Onset Detection with Convolutional Neural Networks ». En: *Proceedings of the IEEE International Conference on Acoustics, Speech and SIgnal Processing (ICASSP)* (2014), págs. 6979-6983.

[15] A. B. Chan T. LH. Li y A. HW. Chun. «Automatic Musical Pattern Feature Extraction using Convolutional Neural Network ». En: *Proceedings of the International MultiConference of Engineers and Computer Scientists (IMECS), Hong Kong* (2010).

[16] E.J. Humphrey y J.P. Bello. «Rethinking automatic chord recognition with convolutional neural networks ». En: *Proceedings of the 11th International Conference on Machine Learning and Applications (ICMLA), Boca Raton, FL, USA* 2 (2012).

[17] Jan Schlüter Thomas Grill. «Structural Segmentation with Convolutional Neural Networks Mirex Submision ». En: 2015.

[18] Keunwoo Choi, György Fazekas y M.Sandler. «Automatic Tagging Using Deep Convolutional Neural Networks ». En: *ArXiV* abs/1606.00298 (2016).

[19] Jan Schlüter y T.Grill Karen Ullrich. «Boundary Detection in Music Structure Analysis Using Convolutional Neural Networks ». En: *Proceedings of the 15th International Society for Music Information Retrieval Conference (ISMIR)* (2014).

[20] *International Society for Music Information Retrieval (ISMIR)*. https://www.ismir.net/.

[21] *Music Information Retrieval Evaluatio eXchange (MIREX)*. https://www.music-ir.org/mirex/wiki/MIREX$_{HOM}$

[22] *IMIRSEL: A secure music retrieval testing environment*. https://experts.illinois.edu/en/publications/imirsel-a-secure-music-retrieval-testing-environment.

[23] Tim O'Brien. «Musical Structure Segmentation with Convolutional Neural Networks ». En: *Proceedings of the 17th International Society for Music Information Retrieval Conference* (2016).

[24] *Librosa Python Library*. https://librosa.org/doc/latest/index.html.

[25] *Anaconda Software*. https://www.anaconda.com/products/individual.

[26] *Project Jupyter*. https://jupyter.org/.

[27] *Keras: the Python Deep Learning API*. https://keras.io/.