Trabajo Fin de Grado

Separación de fuentes y transcripción musical con deep learning

Source separation and music transcription with deep learning

Autor

Pablo Castillón Ruiz

Director

José Ramón Beltrán Blázquez

Carlos Hernández Oliván

ESCUELA DE INGENIERÍA Y ARQUITECTURA

2022

RESUMEN

El objetivo de este trabajo es conseguir una transcripción MIDI[1] a partir de un archivo MP3 en melodías polifónicas (varias notas sonando al mismo tiempo) y con varios instrumentos sonando a la vez, haciendo uso del *deep learning*[2]. Para ello primero se separa cada instrumento mediante el uso de la librería *Demucs*[3], y luego se entrena un modelo que detecta las notas completas (*frames*). El *deep learning* es una herramienta que ha evolucionado en gran medida estos últimos años gracias al avance de la tecnología. La implementación se realiza mediante Python[4] en el entorno de Anaconda[5], sobre todo, con los programas de Spyder[6] y Jupyter notebook[7]. Además, para facilitar el trabajo con las redes neuronales se utilizará Tensorflow[8].

ABSTRACT

The aim of this work is to obtain a MIDI transcription from an MP3 file in polyphonic melodies (several notes sounding at the same time) and with several instruments sounding at the same time, making use of *deep learning*. This is done by first separating each instrument using the *demucs* library, and then training a model that detects the whole notes (*frames*). The *deep learning* is a tool that has evolved greatly in recent years thanks to advances in technology. It is implemented using Python in the Anaconda environment, especially with the Spyder and Jupyter notebook programs. In addition, Tensorflow will be used to facilitate the work with the neural networks.

Índice

Capítulo 1

Introducción

1.1. Motivación

La inteligencia artificial surge definitivamente a partir de algunos trabajos publicados en la década de 1940 que no tuvieron gran repercusión, pero a partir del influyente trabajo en 1950 de Alan Turing [9], matemático británico, se abre una nueva disciplina de las ciencias de la información.

Una IA[10] es, básicamente, un tipo especial de sistema informático que intenta imitar el complejo proceso mental humano. La diferencia fundamental entre un software común y una IA, radica en que un software actúa según su programación y hace lo que el desarrollador ha indicado, pero una IA aprende a realizar una tarea y crea su propia programación para solucionar problemas. La inteligencia artificial debe pensar por sí sola o al menos simular una forma de razonamiento.

En muy poco tiempo las tecnologías de inteligencia artificial han alcanzado un progreso inimaginable hace unos años. Un ejemplo de esto es el robot Atlas de Boston Dynamics[11], capaz de hacer una voltereta en el aire. La lista de aplicaciones de la inteligencia artificial es ilimitada y cada vez se están desarrollando otras nuevas. Como podemos ver en la figura 1.1 la inversión en este campo no deja de crecer.

La IA no sólo incrementa la productividad a nivel de maquinaria, sino que también hace que incremente la productividad de los trabajadores y la calidad del trabajo que realizan. El poder disponer de mayor información, les permite tener una visión más focalizada de su trabajo y tomar mejores decisiones.

1.2. Objeto y Alcance

Lo que me ha llevado a realizar esta investigación para mi trabajo de final de grado ha sido el poder saber como funciona el deep learning aplicado a la transcripción musical, ya que es una parte importante en el análisis de señales musicales. Con este trabajo se contribuye

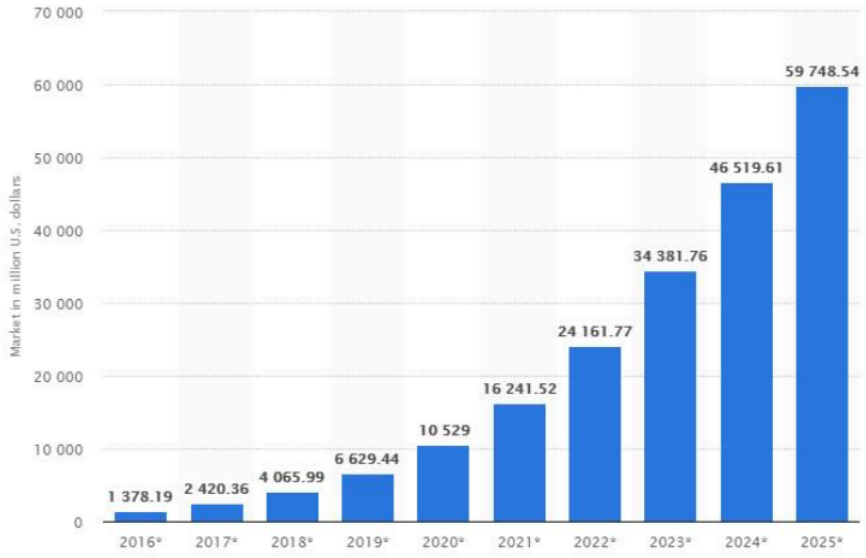


Figura 1.1: Aumento de la inversión capital en proyectos relacionados con la IA [12].

a poder desarrollar nueva música de forma más rápida y eficiente, así como a abrir nuevas puertas a mejorar la educación musical.

Este trabajo se va a centrar en mejorar el entrenamiento de una red neuronal (ya preentrenada) en el análisis musical. Se va a intentar que mejore su análisis a melodías polifónicas (varias notas a la vez) y con varios instrumentos sonando a la vez. De todos los instrumentos que tiene cada canción nos centraremos en el bajo.

El lenguaje de programación usado principalmente es Python, utilizando las herramientas de Spyder y Jupyter Notebook, pero también se han utilizado comandos CMD[13]. Se ha entrenado la red haciendo uso de una base de datos llamada Slakh2100[14], compuesta por 1823 canciones.

Para poder llevar a cabo esta tarea hemos realizado diferentes desarrollos software:

- **Conversión a '.wav' de los archivos '.mp3'**: Se encuentran las funciones necesarias para poder utilizar los archivos de la base de datos.
- **Modificación del módulo de procesamiento de entrada**: El módulo de entrada almacena las funciones que permiten procesar las señales de audio. Se ha modificado esas funciones para adaptarlas a nuestra base de datos.
- **Modificación del módulo de procesamiento de test**: Realizamos un test inicial antes de entrenar la red, para ver de qué punto partimos, ya que ha sido entrenada con otra base de datos.
- **Modificación del módulo de procesamiento de entrenamiento**: Se ha modificado el módulo de entrenamiento, en el cual están las funciones que permiten el entrenamiento

de la red y la visualización de las salidas.

– **Realización de un test**: Para ver el comportamiento final de la red.

Con este trabajo conseguimos que la red neuronal pueda transcribir de forma precisa una melodía multimímbrica y con varios instrumentos sonando simultáneamente, ya que al ser un problema complejo, la red no era capaz de hacerlo. Como resultados final, extraemos un archivo MIDI a partir de uno WAV.

1.3. Organización de la Memoria

La memoria consta de un breve resumen del trabajo realizado y 6 capítulos. El primero es el de introducción 1, que abarca la motivación de realizar este trabajo, así como las diferentes tareas a realizar.

En el capítulo 2 se describen aquellos conceptos necesarios para entender el trabajo, incidiendo en su mayoría en una introducción a las redes neuronales y conceptos relacionados con estas.

En el capítulo 3 se describe el estado del arte que pertenece a este trabajo. Veremos diferentes modelos de separación de fuentes y conceptos relacionados con la transcripción automática de música.

En el capítulo 4 se describen los modelos utilizados para la separación de las canciones, la base de datos utilizada y el entrenamiento y arquitectura del modelo.

En el capítulo 5 se muestran los resultados del trabajo y se hace una breve discusión sobre ellos, y en el capítulo 6 se habla sobre las conclusiones y los proyectos que pueden continuar con este experimento.

Capítulo 2

Introducción al Deep Learning, la transcripción automática de música y la separación de fuentes

En este capítulo se va a hacer un desarrollo de los conceptos más importantes relacionados con el trabajo, para así poder explicar más adelante con más detalle lo que se ha hecho a nivel práctico.

2.1. Inteligencia artificial

La Inteligencia Artificial (IA) es la combinación de algoritmos planteados con el propósito de crear máquinas que presenten las mismas capacidades que el ser humano[15]. Este concepto de Inteligencia Artificial engloba al de Machine Learning, y este a su vez a las redes neuronales, figura 2.1. Dentro de las redes neuronales tenemos el Deep Learning, que es donde entraría este trabajo.

El Deep Learning o aprendizaje profundo sería aquel formado por las redes neuronales que tienen muchas capas (por eso se llama profundo). Cada capa se centra en aprender una cosa nueva y por eso permite resolver problemas tan complejos como la transcripción de musica o la traducción de textos.

2.2. Redes Neuronales Artificiales

La iniciativa de las Redes Neuronales[18] es, como su propio nombre sugiere, intentar de imitar el cerebro humano. Pese a que no se sabe exactamente cómo funciona el cerebro humano, sí que se saben ciertas propiedades como la composición que tiene e inspirándose vagamente en estas visualizaciones emergen unidades llamadas neuronas conectadas entre sí para transmitirse.

Cuando los datos/información de entrada atraviesan esta red de neuronas, se termina

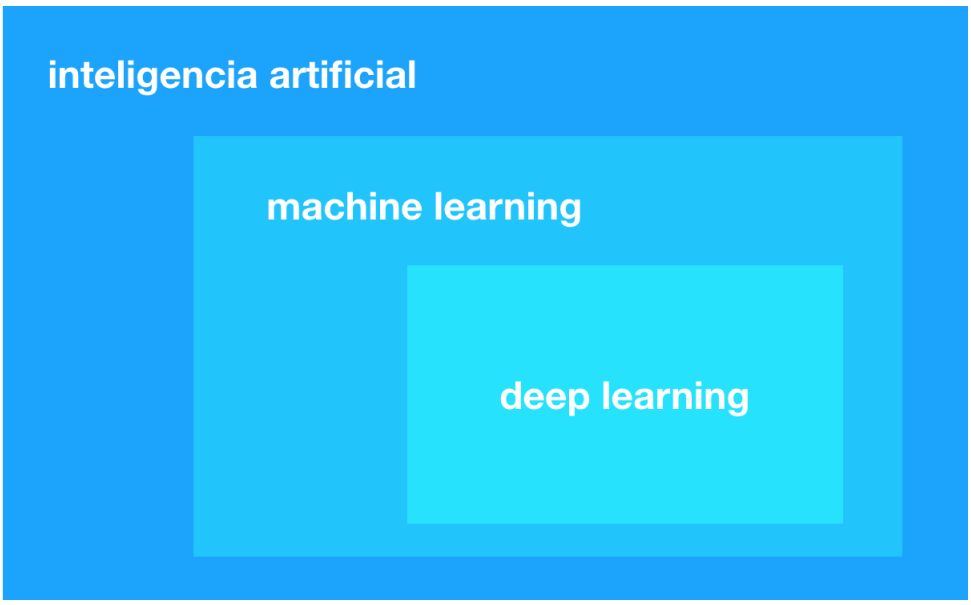


Figura 2.1: Ditribución de la IA[16]

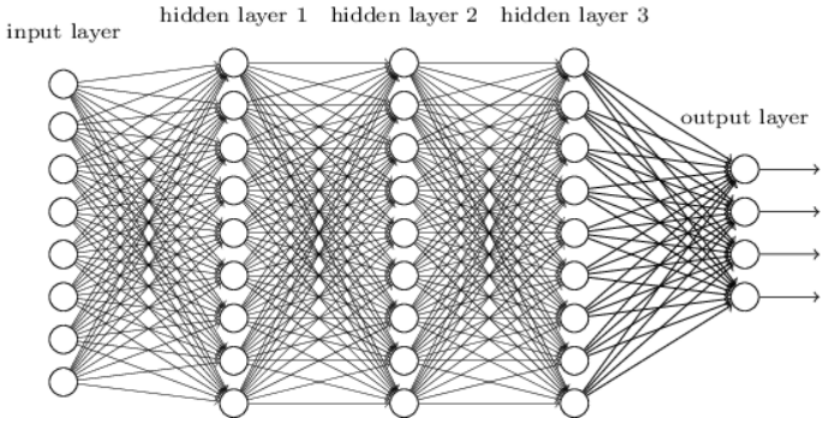


Figura 2.2: Ejemplo de red neuronal con varias capas[17].

produciendo un resultado o unos valores de salida que son los que nos interesan. De este modo se puede entender una red neuronal como una gigantesca función que crea una salida dependiente de unos datos de entrada, esa función para eso se basa en muchas subfunciones que la componen. Es decir, se puede ver una red neuronal como una función de funciones que permiten a la función principal computar y capturar potentes relaciones en los datos.

El objetivo final es que dicha gran función acabe realizando predicciones correctas para los datos de entrada (la salida obtenida por la red sea la esperada). Luego por tanto estamos ante un caso de algoritmo de aprendizaje supervisado en el que necesitamos una función que se adapte a unos datos previamente etiquetados (recibe una serie de salidas esperadas para unas entradas determinadas y necesita ajustarse para producir esas mismas salidas).

2.2.1. Neurona

La unidad más simple de una red neuronal son las neuronas[19], las cuáles se juntan y forman las distintas capas de la red. En el caso de las neuronas artificiales, la suma de las entradas multiplicadas por sus pesos asociados determina el "impulso nervioso" que recibe la neurona. Este valor, se procesa en el interior de la célula mediante una función de activación que devuelve un valor que se envía como salida de la neurona.

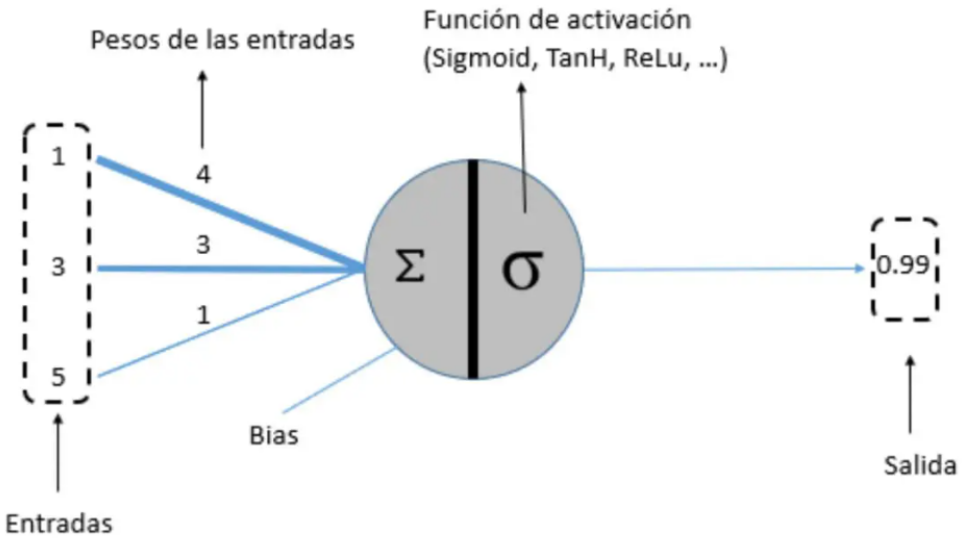


Figura 2.3: Ejemplo de neurona con sus pesos en la entrada [20].

La expresión matemática que daría el resultado de la salida de la red es la siguiente, figura 2.1, siendo y la salida, x_i las entradas, w_i los pesos, b el bias que corresponde a un valor entero y g la función de activación.

$$y = g(b + \sum_{i=1}^{N} x_i w_i) \tag{2.1}$$

Una capa es un conjunto de neuronas cuyas entradas provienen de una capa anterior (o de los datos de entrada en el caso de la primera capa) y cuyas salidas son la entrada de una capa posterior.

Las neuronas de la primera capa reciben como entrada los datos reales que alimentan a la red neuronal. Es por eso por lo que la primera capa se conoce como capa de entrada. La salida de la última capa es el resultado visible de la red, por lo que la última capa se conoce como la capa de salida. Las capas que se sitúan entre la capa de entrada y la capa de salida se conocen como capas ocultas ya que desconocemos tanto los valores de entrada como los de salida.

2.2.2. Entrenamiento de modelos de Deep Learning

Cada neurona tiene una salida, pero para poder obtenerla se necesita una función de activación que dada una entrada o un conjunto de entradas que genere dicha salida. Cada capa de la red neuronal tiene una función de activación [21], además será en la mayoría de los casos una función no lineal, ya que estas permiten que el modelo se adapte mejor a trabajar con la mayor cantidad de datos. Las funciones de activación se dividen en dos tipos como: lineal y no lineal:

Función Lineal

Conocida como identidad, se usa cuando se requiere una regresión lineal y por lo tanto la red va a generar un valor único como salida. Por ejemplo cuando se va a predecir el numero de ventas.

Función Umbral

También conocida como escalón, si la x es menor que 0 la neurona será cero, y cuando sea mayor o igual la salida será 1. Esta función se utiliza cuando se tienen salidas categóricas.

Función Sigmoide

Los valores de salida están entre cero y uno, por lo que se interpreta la salida como una probabilidad. Con valores de entrada negativos la salida tiende a 0, si se evalúa en cero la función dará 0.5 y en valores positivos su valor tiende a 1. Por lo que esta función se usa en la última capa y se usa para clasificar datos en dos categorías. Actualmente la sigmoide no es una función muy utilizada debido a que influye el problema de desaparición de gradiente, esto significa que en los extremos el gradiente se vuelve muy pequeño, lo que hace que el modelo converja lentamente en un mínimo y debido a esto y al *backpropagation*[22] la red no puede obtener un buen aprendizaje.

Tangente Hiperbólica

Función de activación muy parecida a la sigmoide, solo que está si que esta centrada respecto a 0, es decir que en los extremos los valores tienden a 1 y -1 en los valores mas positivos y negativos de x respectivamente. Tiene el mismo problema de desaparición de gradiente.

Función ReLU

Es la función más utilizada ya que el gradiente en el segundo cuadrante es cero y en el primero es uno, por lo que se soluciona el problema de las dos funciones de activación

anteriores.

Función Softmax

Esta función se utiliza para clasificar algo entre varios tipos. Los valores que puede tomar la salida van de -1 en el extremo negativo a 1 en el positivo, pero pudiendo alcanzar cualquier valor en los rangos intermedios. Por ejemplo si queremos clasificar algo entre varias clases de fruta nos dará la posibilidad que hay de que pertenezca a cada clase, siendo la suma de todas ellas la unidad.

2.3. Arquitecturas de Redes Neuronales

Cuando hablamos de la arquitectura de las redes neuronales no solo nos referimos al número de capas neuronales o al número de neuronas en cada una de ellas, también se incluye la conexión entre estas, al tipo de neuronas y hasta la manera de entrenar las redes [23].

Las redes tipo *feed forward* están organizadas en capas y llevan la información desde la capa de entrada (la primera) a la capa de salida (la última).

Estas redes constan de varias capas de neuronas, *completamente conectadas*, como podemos ver en la figura 2.4. Esto quiere decir que cada neurona de una capa esta conectada con todas las neuronas de la siguiente capa y el entrenamiento se realiza por *back propagation*[24] o propagación hacia detrás.

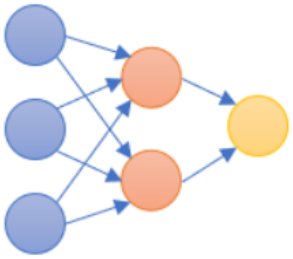


Figura 2.4: Ejemplo de red fully connected [23].

El *back propagation* consiste en una regresión lineal de la forma $ax + b = 0$ para determinar los valores de a y b. La función de coste indicará lo bueno que es el modelo final.

2.3.1. Redes Convolucionales

Las redes neuronales son un tipo de redes que se crearon tomando como modelo el funcionamiento del cerebro humano, capaces de aprender en diferentes niveles de abstracción: en las primeras capas se diferencian formas más simples, colores y bordes o esquinas; en las siguientes ya se pueden distinguir algunas combinaciones de estos y en las últimas capas

ya se pueden distinguir figuras y clasificarlas según nuestras salidas de la red. Estas redes son la última tendencia en procesamiento de imágenes. En la figura2.5 podemos ver una red convolucional con distintas capas.

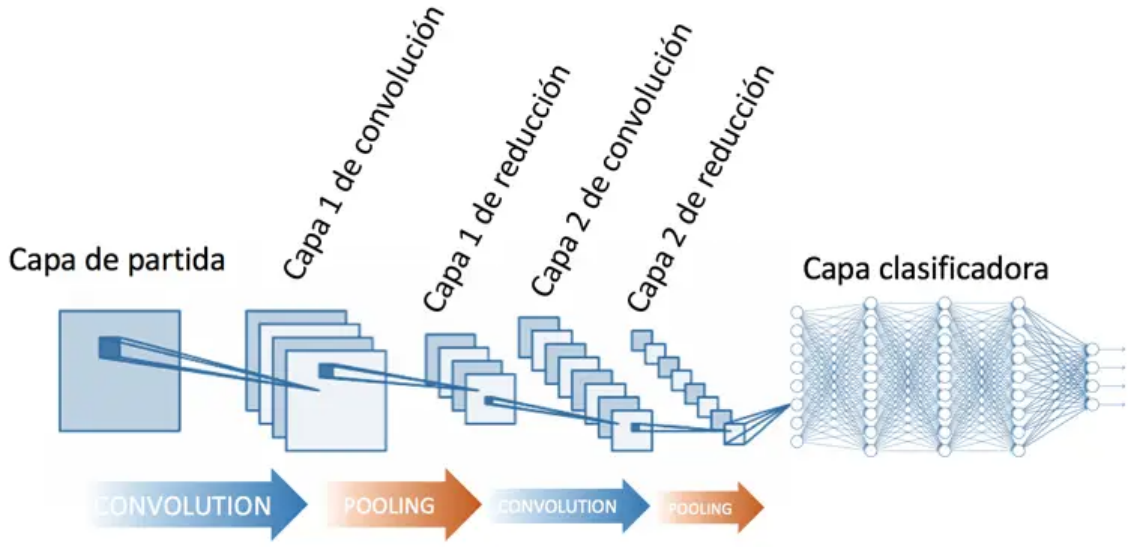


Figura 2.5: Ejemplo de red convolucional [25]

Aunque su función principal va a destinada al procesamiento de imágenes, en este trabajo se van a utilizar para procesar señales de audio. En este caso la imagen utilizada como entrada corresponden a representaciones de tiempo y frecuencia como la transformada de Q constante (CQT) [26]

Estas redes tienen una primera fase de extracción de características, compuesta por neuronas convolucionales, luego una reducción por muestreo y al final se tienen neuronas más sencillas para realizar una clasificación final sobre las características extraídas.

La fase de extracción de características esta formada por capas alternas de neuronas convolucionales y neuronas de reducción de muestreo. A medida que avanzamos en la red, se disminuye la dimensión de las capas, siendo las capas más profundas menos sensibles a variaciones en los datos de entrada, y activándose por características más complejas.

Podemos ver una convolución en la imagen 2.6 y esta se basa en realizar una multiplicación entre una parte de la imagen y un filtro, el cuál tiene varios parámetros que podemos modificar, los cuales son:

- **Tamaño del filtro**: Dimensiones que tiene el filtro que se aplica en toda la imagen, en este caso podemos ver que el tamaño del filtro de la figura es de 3x3. Si la imagen fuera a color tendría que haber 3 filtros iguales, uno para cada color RGB.
- ***Stride***: Es el desplazamiento que utiliza el filtro en la imágenes de entrada. Se mide en píxeles y en esta convolución de la imagen es de 1 píxel.

– ***Padding***: Consiste en añadir ceros alrededor de la matriz de entrada y se utiliza para mantener las dimensiones de la imagen de entrada y de esta forma poder realizar mas convoluciones y hacer la red más profunda y extraer características más profundas. En el filtro que vemos en la imagen el padding es de 1.

– **Número de filtros**: En una misma capa convolucional podemos usar varios filtros.

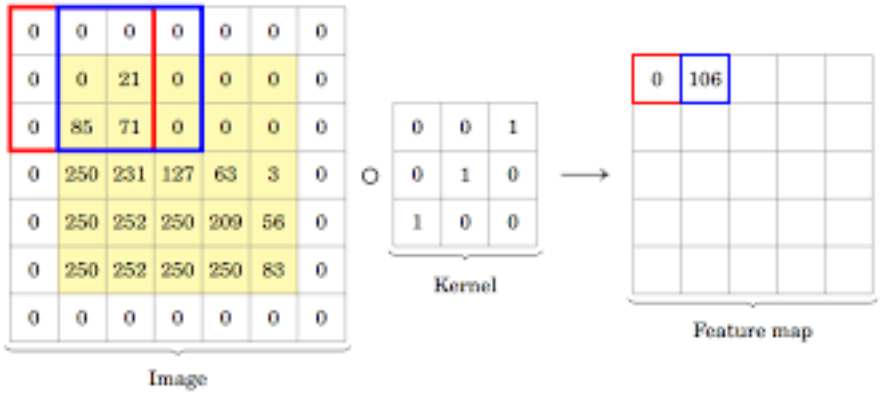


Figura 2.6: Funcionamiento de una convolución [27]

Después de las etapas convolucionales, estas redes suelen utilizar capas de *pooling*, las cuales permiten analizar el contenido de una imagen por regiones para sacar de cada una de ellas la información mas representativa.

Esta capa permite reducir el tamaño de la matriz de entrada en cada capa, lo que facilita el procesamiento de las imágenes manteniendo la información más relevante.

En la figura 2.7 se puede ver un filtro de *Max-Pooling* de tamaño 2x2, en el que de cada región de 2x2 de la imagen se queda con el píxel que tenga el valor máximo. Dicho valor máximo se asigna al píxel que corresponda en la imagen de salida.

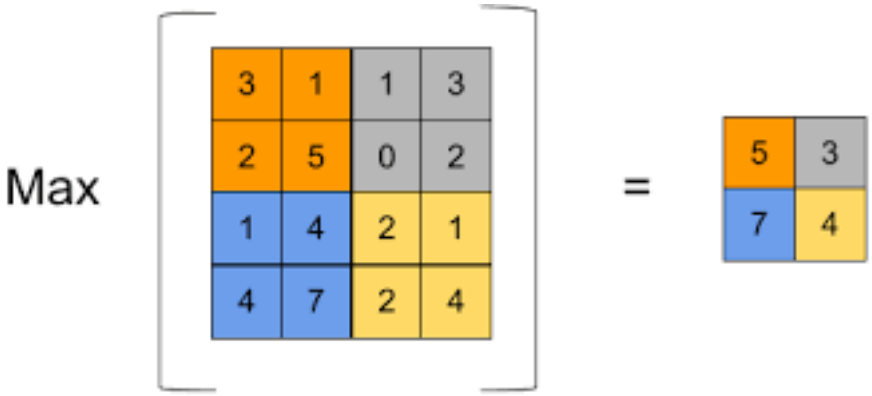


Figura 2.7: Capa de Max-Pooling [28]

2.3.2. Transformer

Vamos a nombrar también un tipo de arquitectura bastante importante en transcripción musical, y es en el que se basa Google para hacer algunos de sus modelos. Esta arquitectura pertenece al modelo Transformer[29].

Este modelo como podemos ver en la figura 2.8 esta compuesto por codificadores y decodificadores conectados entre sí. El bloque codificador esta compuesto por una pila de 6 codificadores uno encima de otro (se podría probar con otro número diferente ya que el número 6 no tiene nada de especial en este caso). Y el bloque de decodificación es una pila de decodificadores del mismo número que el bloque anterior.

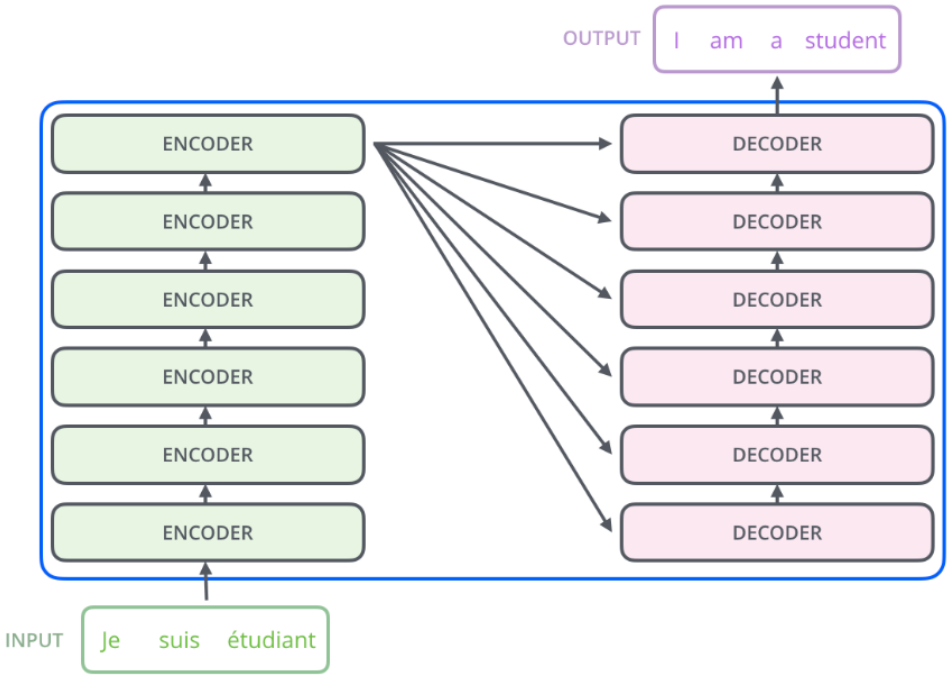


Figura 2.8: Bloques codificador y decodificador del modelo Transformer [30]

Los codificadores y decodificadores tienen la misma estructura, solo que el decodificador tiene un capa extra que le ayuda a centrarse en las partes más importantes de la entrada. En la figura podemos ver como está formado un codificador, aplicándose una última capa de normalización. A continuación vemos una imagen 2.9 de la estructura de cada bloque.

A la salida del bloque decodificador tenemos una última capa lineal y *softmax* para acabar de clasificar y resolver nuestro problema. Por último tenemos una figura 2.10 donde se puede ver la arquitectura de todo el modelo entero.

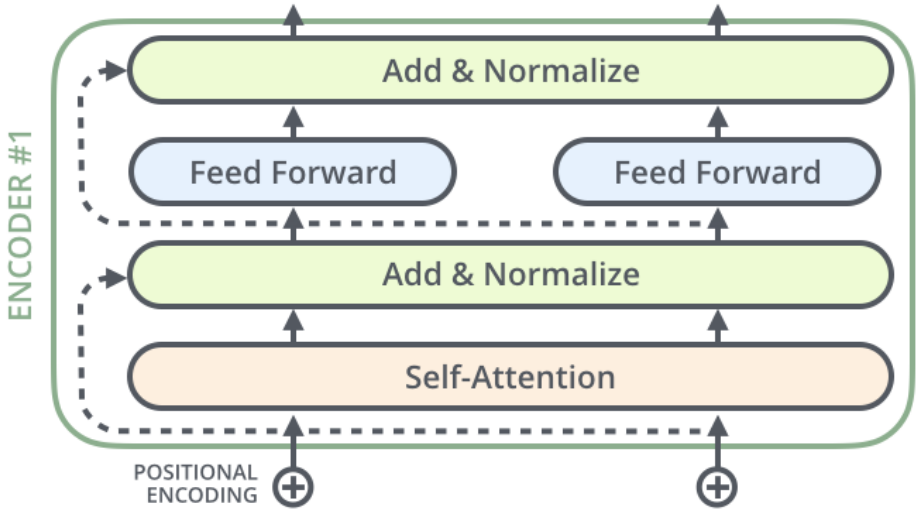


Figura 2.9: Estructura de un codificador y decodificador del modelo Transformer [30]

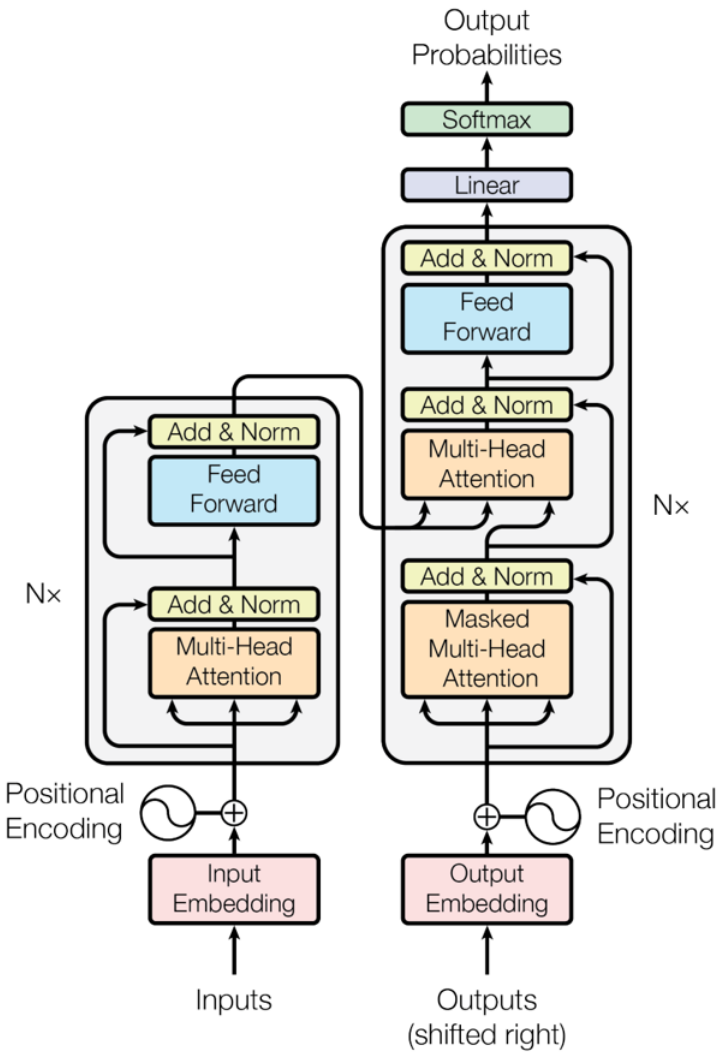


Figura 2.10: Arquitectura del modelo Transformer [30]

Capítulo 3

Estado del Arte

En este capítulo se van a desarrollar todos aquellos conceptos necesarios para entender el trabajo y aquellas tecnologías y estructuras en las que se fundamenta el mismo.

Se hablará de que es la separación de fuentes y los diferentes tipos de modelos que hay, así como de las métricas de esta. También se explicará que es la transcripción automática de música y se profundizará en lo necesario para llevarla acabo, así como de los principales modelos utilizados.

3.1. Separación de Fuentes

Ya sea en cualquier situación cotidiana, como hablar con otras personas, o en una melodía musical, el oído humano recibe sonidos y ruidos de varias fuentes sonoras y es capaz de filtrarlos, reconocerlos y seleccionar aquellos que le interesan (en la mayoría de ocasiones). Esto es precisamente lo que se quiere conseguir con la separación de fuentes, en este caso musicales.

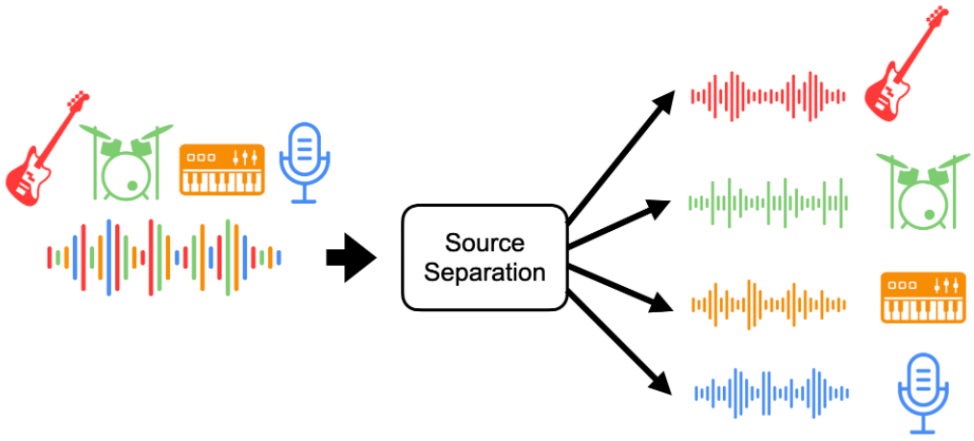


Figura 3.1: Separación de una fuente musical en varias pistas de instrumentos [31].

La separación de fuentes consiste en separar un sonido formado por varias fuentes en cada una de ellas individualmente, haciendo referencia a la separación musical, se estaría haciendo

alusión a separar de una canción cada uno de los instrumentos y la vocal. Un ejemplo sería la figura3.1, donde separa los instrumentos de l canción en 4 pistas diferentes.

Matemáticamente hablando decimos que la señal mezclada $y(t)$ se compone de varias fuentes $x_i(t)$, con $n = 1...N$ de la forma:

$$y(t) = \sum_{i=1}^{N} x_i(t) \tag{3.1}$$

Dada la señal mezclada $y(t)$ el objetivo es conseguir todos las fuentes separadas $x(t)$. Pero con frecuencia hay menos señales de mezcla que fuentes dentro de la mezcla, y por eso la separación de fuentes se considera un problema indeterminado, es decir, que se tienen más fuentes separadas que mezclas. Si en una melodía se tiene un violín, una viola y un piano, habría que separar tres instrumentos, pero solo poseemos dos canales. En este caso la separación de fuentes se podría utilizar para separar solamente el piano.

Este trabajo esta centrado en las señales musicales, las cuáles debido a algunas de sus características y la forma en la que son compuestas son de las más difíciles de separar. Algunas de estas son:

- **Procesamiento de forma no física y no lineal**: Las técnicas de procesamiento no lineales (filtrado, reverberación, etc) que se utilizan para componer las melodías dificultan su separación, y hoy en día las utilizan todos los compositores. Así como que las mezclas debido a la forma de grabarse nunca podrían darse en el mundo real.
- **Cambio de fuente**: En una canción o melodía frecuentemente todas las fuentes varían a la vez, es decir, que cuando una cambia la nota, el resto suelen cambiarla también, por ejemplo al principio de un compás.

A continuación se explican algunos modelos que se se pueden utilizar para la separación de fuentes musicales.

3.1.1. Modelos

Demucs

La red neuronal Demucs[3] fue programada por Facebook, con la intención de separar canciones en cuatro diferentes fuentes, la batería, el bajo, la voz y el resto de instrumentos. Esta red creada en 2019 esta hecha en Pytorch y esta disponible en GitHub para cualquiera que quiera utilizarla.

Demucs se basa en el modelo de red Wave-U-Net[32], por lo que su arquitectura es muy parecida a las U-Net, posee primero algunas capas codificadoras que comprimen la entrada y luego capas decodificadoras que la descomprimen. Tanto la entrada como la salida de esta red

son las ondas de sonido directamente, y como se puede ver en la figura3.2 tenemos a la entrada de la red la onda mezclada con todos los instrumentos y a la salida las cuatro pistas de audio comentadas anteriormente. Esta red tiene 6 capas codificadoras y 4 decodificadoras, ya que tiene que ser simétrica. En el apartado 4 nos centraremos más en esta red y estará detallada más a fondo su arquitectura, ya que es la que hemos usado para separar las canciones de nuestra base de datos.

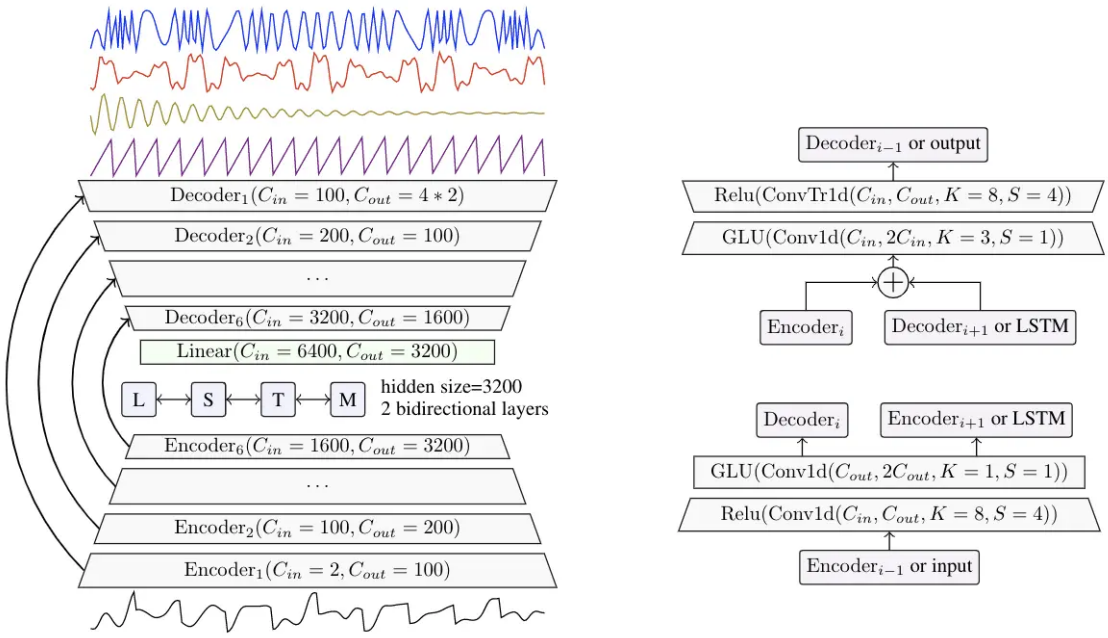


Figura 3.2: Arquitectura de la red nueronal Demucs [3]

El entrenamiento de este modelo se hizo con la base de datos MusDB[33], la cual esta formada por 150 canciones separadas por pistas. Cada una de ellas se dividió en trozos de 11 segundos y se agruparon por lotes de 16 para el entrenamiento. Con diferentes técnicas se consiguió aumentar los datos: se invirtieron las ondas, se desplazaron las pistas temporalmente, se intercambiaron los canales de estéreo y hasta se intercambiaron varias pistas entre canciones de un mismo lote.

Spleeter

Spleeter[34] es una red neuronal desarrollada por el equipo de Deezer R&D (es una plataforma de música parecida a Spotify) y se basa, al igual que Demucs, en la red neuronal U-Net y adapta la arquitectura de esta a la tarea de separación de fuentes. Ya que la función inicial de estas redes basadas en esta arquitectura era la de mejorar la localización y precisión microscópica de imágenes biomédicas de las estructuras neuronales.

Spleeter contiene modelos pre-entrenados para distintas separaciones de fuentes:

- **Separación de 2 fuentes**: vocal y acompañamiento.

– **Separación de 4 fuentes**:vocal, batería, bajo y acompañamiento.

– **Separación de 5 fuentes**:vocal, batería, bajo, piano y acompañamiento.

Fue el primer modelo en permitir separación de 5 fuentes diferentes.

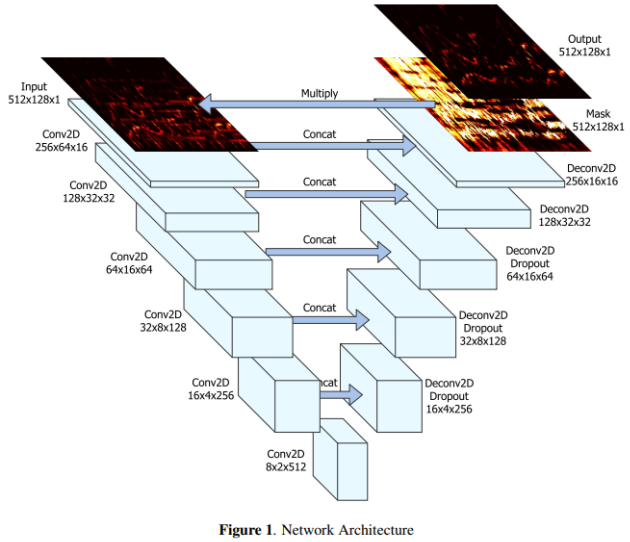


Figure 1. Network Architecture

Figura 3.3: Arquitectura de la red nueronal Spleeter [35]

La U-Net es una red neuronal convolucional con arquitectura codificador/decodificador. En este caso, como podemos ven en la figura3.3 se usan 12 U-Nets, 6 para codificar y 6 para decodificar. Se utiliza una U-net para estimar una máscara blanda para cada fuente. El modelo fue entrenado con una base de datos interna de la propia empresa Deezer, por lo que es privada y no se revelaron más datos, durante una semana aproximadamente en una sola GPU. Fue entrenada en una en TensorFlow, lo que hace posible poder ejecutar el codigo en la CPU(Central Processing Unit) o en la GPU.

Cerberus

La principal contribución de este trabajo es que esta red Cerberus[36], en lugar de tener que separar una canción y luego transcribirla o hacer solo una de las dos cosas, es capaz de realizar las dos tareas a la vez usando una sola red neuronal en un escenario multitarea. Fue posible aprovechar las similitudes entre ambas tareas y hacer un modelo compartido para poder realizar las dos a la vez.

La arquitectura de la red neuronal esta basada en la red Chimera[37], la cual era utilizada para separación de fuentes. En la red Chimera se toma como entrada un espectrograma de magnitud TxF y se estiman dos salidas. Una de ellas es la típica salida de los algoritmos de separación de fuentes, una máscara con las mismas dimensiones que la entrada. La otra

es una compresión de la dimensión TFx20, donde cada fila representa una proyección del bin(valor discreto) tiempo-frecuencia del espectrograma en un espacio dimensional de 20.

En la red Cerberus, se añadió una nueva capa de salida a estas ya presentes en la arquitectura de la red Chimera. Se le llamó red Cerberus haciendo referencia al perro de 3 cabezas de la mitología griega. Esta nueva salida es un piano-roll de cada uno de los instrumentos, con un total de 88 posibles notas de MIDI[1]. En la figura3.4 podemos ver su arquitectura.

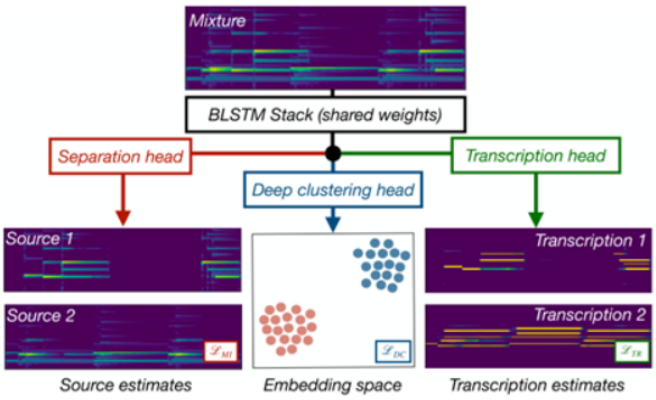


Figura 3.4: Arquitectura de la red nueronal Cerberus [36]

La base de datos utilizada en el entrenamiento de esta red fue la Slakh2100[14], creando grupos de mezclas sintetizadas. Las combinaciones de instrumentos para cada grupo fueron: (1) guitarra y piano; (2) guitarra, bajo y piano; (3) guitarra, bajo, piano y batería; (4) guitarra, bajo, piano, batería y cuerda.

3.1.2. Métricas

La separación de fuentes es un problema muy difícil de evaluar, por eso existen una serie de métricas que ayudan a cuantificar este resultado.

Las dos principales categorías utilizadas para evaluar los resultados de una separación de fuentes son: objetiva y subjetiva. Las medidas objetivas califican la calidad de la separación a través de un conjunto de cálculos que comparan las fuentes aisladas reales con las señales de salida del sistema de separación. Las medidas subjetivas se llevan a cabo mediante unos calificadores humanos que puntúan los resultados del sistema de separación.

Ambas formas de evaluar tienen sus inconvenientes. En las medidas objetivas hay muchos aspectos de la percepción humana que son muy difíciles de captar mediante medios informáticos. En cambio, las medidas subjetivas son mucho más caras y lentas, además dependen de cada evaluador, por lo que no hay un sistema real de referencia.

Las medidas subjetivas pueden ser más fiables ya que hay un evaluador humano, pero

son mucho más utilizadas las subjetivas debido a su coste computacional y económico. Las medidas objetivas más importantes son:

- **Source-to-Artifact Ratio (SAR)**: Cantidad de ruidos no deseados que hay en el audio separado en relación con la fuente real separada.
- **Source-to-Interference Ratio (SIR)**: Cantidad de otras fuentes de audio que se escuchan en la separación de la fuente que corresponda.
- **Source-to-Distortion Ratio (SDR)**: Se considera la medida global de la calidad del sonido de una fuente separada. Cuando solo se proporciona una cifra suele ser esta.

Estos son los tres principales términos utilizados para definir nuestras medidas, las cuáles se indican en decibelios. A mayor es el valor, mejor es el resultado de la separación. Y para poder calcularlas se necesita disponer de las diferentes fuentes reales del audio separadas, es decir sin ningún ruido grabas por separado.

3.2. Transcripción Automática de Música

Lo que se busca con la transcripción automática de música es obtener a partir de un archivo de audio, ya sea *.wav, .mp3*, etc, una representación de este en algún formato de notación musical, como por ejemplo el .mid (MIDI).

Este es un proceso muy complejo y existen muchos problemas a la hora de intentar transcribir audios, ya que hay que tener en cuenta un montón de características, no solo las notas. Para llevar a cabo estas tareas se suele utilizar como entrada a nuestra red los espectrogramas (imagen del espectro de la señal en ventanas de tiempo), así como la *Constant Q-Transform* (CQT), de la cuál se hablara a continuación.

3.2.1. Constant Q-Transform

La transformada de Q constante (CQT)[38] es un algoritmo que se fundamenta en el uso de una escala de frecuencia logarítmica, cuyo objetivo es mejorar la resolución espectral a bajas frecuencias y la resolución temporal a altas frecuencias. Para conseguir esto, la CQT toma el mismo número de muestras en frecuencia por octava. Gracias a esto es tan utilizada la CQT en procesado de señales musicales.

Un principio en el que se fundamenta la CQT es que, para sonidos formados por componentes armónicos de frecuencia, las posiciones relativas entre el fundamental y sus armónicos son las mismas, independientemente de la frecuencia fundamental, siempre y cuando su representación sea logarítmica. De esta forma es mucho más fácil identificar los instrumentos o detectar la F0 [39].

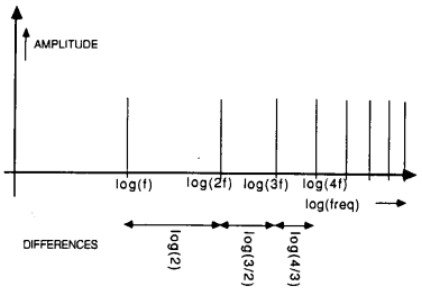


Figura 3.5: Representación de la transformada de Fourier en relación con el logaritmo de la frecuencia [38]

El principal problema a la hora de hacer la transformada corta de Fourier (STFT) [40] en un dominio del logaritmo de las frecuencias, es, que para bajas frecuencias no tenemos casi información y para altas frecuencias hay demasiada. Si se toma una ventana fija de 1024 muestras y suponemos que la frecuencia de muestreo es de 32000 muestras cada segundo, la resolución es de 32000/1024 = 31,3Hz. En el caso de la escala más baja del violín que es de 196Hz, esto es casi un 16 % de la frecuencia (bastante más elevado que el 6 %, que es la distancia que hay entre dos notas adyacentes, por eso esta resolución frecuencial no sería suficiente). Por el contrario, en el rango alto de un piano que sería de 4186Hz, esta resolución frecuencial de 31,3Hz supone un 0,7 % y, por tanto, sería demasiado grande.

Si queremos que la resolución sea suficiente, para poder distinguir correctamente un semitono, la resolución a esas frecuencias tendrá que ser alrededor de un 3 % de la frecuencia de la nota que se esté analizando. Este valor corresponde a la mitad de 6 %, $2^{\frac{1}{12}} - 1 \approx 0{,}06$. El ratio que relaciona la frecuencia con la resolución se llama Q (factor de calidad) y para el ejemplo anterior valdrá f/0.03f = 34 y se mantendrá constante durante el cálculo. [38]

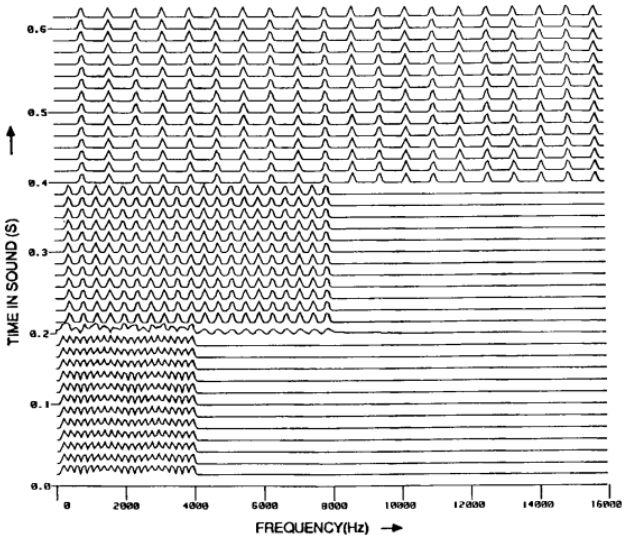


Figura 3.6: Transformada de Fourier de tres sonidos con frecuencias de G3 (196Hz), G4 (392Hz) y G5 (784Hz) [38]

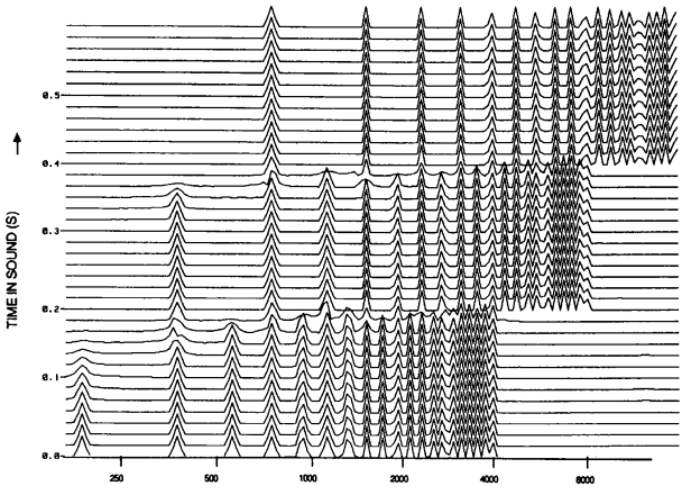


Figura 3.7: CQT de tres sonidos con frecuencias de G3 (196Hz), G4 (392Hz) y G5 (784Hz) [38]

Es necesario modificar el tamaño de la ventana en relación a la frecuencia para que se mantenga este factor de calidad, o lo que es lo mismo, que se calcule la frecuencia con el mismo número de periodos para cualquier valor de nota. La ecuación 3.2, permite calcular el tamaño de la ventana N[k], donde f_s es la frecuencia de muestreo, Q es el factor de calidad (34 es el ideal para diferenciar correctamente semitonos) y f_k es la frecuencia que queremos detectar.

$$N[k] = \frac{f_s}{f_k} Q \tag{3.2}$$

Si miramos las figuras 3.6 y 3.7 se puede ver que la representación es mucho más clara en la figura 3.7, la de la CQT, ya que pese a ser frecuencias distintas estan todas a la misma distancia y no perdemos información.

En este trabajo la CQT se ha calculado a través de la librería de Python librosa [41].

3.2.2. Archivos MIDI

Las siglas MIDI[1] hacen referencia a la abreviatura *Musical Instrument Digital Interface*, que es un protocolo de interacción entre diferentes señales que provienen de varios instrumentos y dispositivos de creación musical.

Un archivo MIDI es un archivo que utiliza este protocolo y almacena los sonidos musicales en un archivo de gran calidad, que no ocupa mucha memoria y que permite ser modificado. Este archivo supuso una mejora muy grande en la creación de musica de manera digital, siendo mucho mejor que sus archivos de audio predecesores, como eran el *.mp3* y *.wav*.

Este archivo contiene una serie de instrucciones y de números que se envían a un instrumento musical digital (controladores, sintetizadores, instrumentos electrónicos, etc.),

y este leyendo esas instrucciones hace sonar una nota. Las principales instrucciones enviadas por este archivo son:

- ***Onset***: Indica el comienzo de una nota, es decir, el momento en que se pulsa el aparato MIDI[1] empleado.
- ***Offset***: Momento de finalización de esa misma nota.
- **Número de nota o *pitch***: Tono de la nota musical.
- **Velocidad**: Indicador de la potencia con la que ha sido pulsada la nota.
- **Tempo**: Los golpes por minuto (*BPM*) a los que va la melodía y que nos indican la rapidez de esta. La duración de un BPM equivale a la duración de la nota llamada *negra*, que el en caso de una melodía con 60 BPM, esta duración será de un segundo. Mientras que si la melodía va el doble de rápido, 120 BPM, esta duración será de medio segundo.
- **Volumen** de la pieza o archivo MIDI[1].

Los datos en formato MIDI[1] los utilizaremos para realizar una representación de la música en un formato *pianoroll*[42] que será útil para el entrenamiento de nuestra red. Un *pianoroll* es una matriz en la que en el eje Y se encuentran las notas musicales, y en el eje X el tiempo. La nota que está sonando tiene un valor equivalente a la velocidad y las notas que no están sonando su valor es 0. En la figura **??** se puede ver un ejemplo de *pianoroll*.

3.2.3. Pianoroll

Es otro de los conceptos que es importante entender, ya que se va a usar para poder representar los archivos MIDI[1]. El pianoroll[42] es una forma de representación de las notas, que consiste en una matriz donde en el eje Y se encuentran las notas musicales, y en el eje X el tiempo. La nota que esta sonando tiene un valor dependiendo de cual sea, y las que no suenan son 0.

De esta forma como se puede ver en la figura 3.8 queda muy claro en que momento suena una nota o no, así como cuando empieza, cuando acaba y que nota es.

3.2.4. Modelos

Al estimar múltiples frecuencias fundamentales (multi-F0) es posible detectar en un mismo frame varias notas sonando al mismo tiempo. Sin embargo, esto es muy complicado ya que es posible que los armónicos de las notas se solapen, por lo que la extracción de estas es muy difícil. Esto es lo que se busca con los siguientes modelos de redes neuronales basadas en transcripción multi-F0.

Figura 3.8: Pianoroll de una Orquesta [42]

DeepSalience

El objetivo del modelo Deep Salience[43] es estimar las notas en una melodía polifónica. Este modelo esta entrenado con una base de datos de 240 canciones, 108 recogidas de la base de datos MedleyDB[44] y otras 132 pertenecientes a la música pop occidental de los años 80 hasta la actualidad.

Se procesan las señales de audio de la entrada mediante una CQT implementada con librosa[41], y se les aplica una normalización de los valores para encontrarse en el intervalo [0,1].

Como entrada a la red no se utiliza una CQT, sino 6 armónicos apilados, y como salida se tiene una representación tiempo-frecuencia de un tamaño igual que cada uno de los armónicos de entrada.

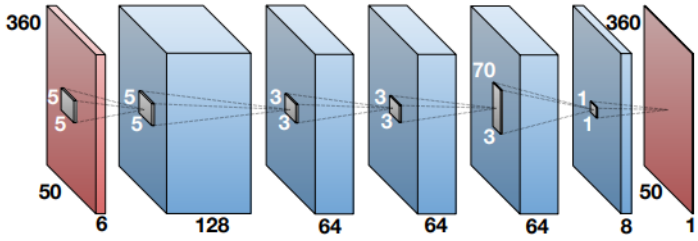


Figura 3.9: Arquitectura de la red neuronal Deep Salience [43]

Como podemos ver en la figura3.9, la arquitectura del modelo consiste en 5 capas convolucionales. Donde la primera capa y la segunda tienen 128 y 64 filtros respectivamente, de tamaño 5x5. Las dos siguientes tienen 64 filtros de 3x3. y la última capa tiene 64 filtros de 70x3. En toda la red se mantiene el mismo tamaño de frecuencia y tiempo (360, 50), para evitar pequeños cambios que podría haber en la frecuencia.

Onsets and Frames

Otro modelo que vamos a ver es el de Onsets & Frames[45], esta red está incluida en el proyecto Magenta de Google y a diferencia de la Deep Salience, la salida de esta red son las notas en formato MIDI, en lugar de la frecuencia fundamental, es decir las frecuencias que están sonando en una ventana de tiempo pequeña.

Se utiliza una red convolucional profunda y recurrente que está entrenada para detectar los *onsets* (el inicio de una nota) y la duración de esta, o lo que es lo mismo, los *frames* que esa nota esta sonando. Se entrenan conjuntamente los *frames* y los *onsets*, ya que los primeros solo actúan si el segundo ha dicho que hay una nota activa.

Esta red solo ha sido entrenada para piano, con la base de datos llamada MAPS[45], donde hay tanto anotaciones con las notas aisladas como el audio.

La arquitectura del detector de *onsets* está formada por un modelo acústico basado en la arquitectura presentada en [46] con algunas modificaciones, seguida de 128 unidades LSTM[47] bidireccionales y por último una capa *Fully Conected* con una sigmoidea de 88 neuronas que se corresponden con las 88 notas que tiene el piano.

Mientras que la arquitectura que se encarga de detectar los *frames* es el modelo acústico, seguido de una *Fully conected* con una sigmoidea de 88 salidas. Estas salidas se concatenan con la salida del detector de *onsets* y a continuación hay 128 unidades LSTM bidireccionales. En último lugar hay una capa *Fully conected* con una sigmoidea de 88 neuronas. Para poder saber si el detector de *onsets* esta activo, se usa un umbral o *threshold* de 0.5. A continuación podemos ver en la figura3.10 la arquitectura de esta red.

MT3

El modelo MT3[48] ha sido utilizado en este trabajo para comparar como resulta la transcripción final de un archivo MIDI, con nuestra red entrenada por bajos separados y sin separar, en relación a este modelo actual en el mercado, que es uno de lo más avanzados en el tema de separación de fuentes y transcripción musical.

Este modelo no solo transcribe, sino que también se encarga de hacer separación de fuentes. Para entrenar este modelo se utilizaron diferentes pistas de 6 bases de datos distintas. Con canciones de varios estilos musicales, diferentes duraciones e instrumentos y métodos de grabación distintos. Las bases de datos utilizadas para el entrenamiento fueron: MAESTROv3[49], Slakh2100[14], Cerberus[50], GuitarSet[51], MusicNet[52] y URMP[53].

La arquitectura de este modelo se basa en la arquitectura de Transformer[29], T5 [54] en este caso. Esta arquitectura consiste como se puede ver en la figura 3.11 en un modelo Transformer codificador-decodificador, con dos bloques de cada uno de ellos. Se utiliza una T5

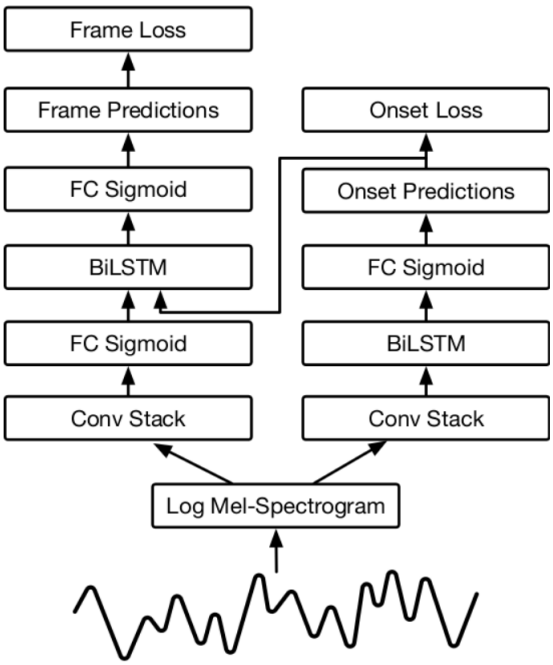


Figura 3.10: Arquitectura de la red neuronal Onsets & Frames [45]

'pequeña', ya que solo consta de 60 millones de parámetros, cuando comúnmente se utilizan modelos mucho mas grandes en este tipo de tareas de clasificación.

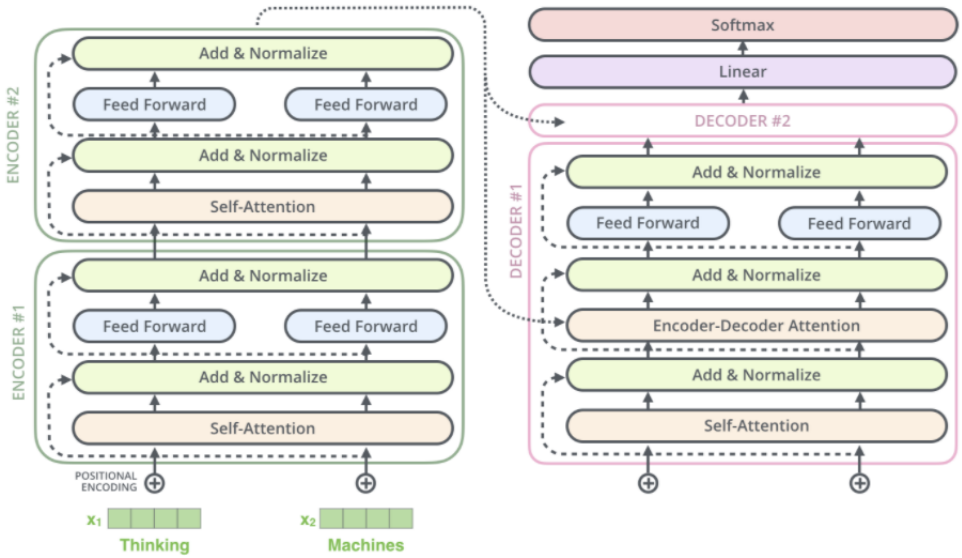


Figura 3.11: Arquitectura de la red neuronal MT3 [55]

3.2.5. Métricas

Para evaluar algún modelo siempre se utilizan métricas, las cuáles cambian en función de lo que se quiera evaluar o a que va destinado el modelo. En este caso, como nuestro objetivo es ver el nivel de acierto del modelo al predecir el tiempo en el que suena una nota y su valor,

se van a utilizar una serie de métricas utilizadas en clasificación.

Para calcular estas métricas vamos a utilizar los positivos verdaderos (TP), los positivos falsos (FP), los negativos verdaderos (TN) y negativos falsos (FN). Las métricas utilizadas son:

- **Precisión**: Es el porcentaje de veces que se acierta un positivo cuando el modelo dice que es positivo. En este caso sería el porcentaje de las notas que ha acertado respecto al total de las notas que ha dicho que eran una nota. Si el modelo ha dicho que había 5 *onsets* de notas y solo ha acertado 4 de ellos, sería un 0,8 la precisión, pero si dice que solo hay una nota y esa nota existe la precisión sería un 1 aunque se hubiese dejado 10 notas. En la ecuación3.3 podemos ver que es aciertos dividido entre aciertos más fallos.

$$Precisión = \frac{TP}{TP + FP} \tag{3.3}$$

- ***Recall***: Es el porcentaje de veces en las que el modelo ha predicho una nota y ha acertado, pero esta vez, frente a todas las notas que hay en la pista y no ha conseguido predecir. Si el modelo dice que hay 4 notas y en realidad hay 5, el *recall* sería un 0.8, el problema es que si dice que hay 2000 notas cuando en realidad hay 5, si esas 5 están entre esas 2000, el *recall* será 1. Podemos ver la ecuación del *recall*3.4 a continuación.

$$Recall = \frac{TP}{TP + FN} \tag{3.4}$$

- ***F-Score***: Debido a los problemas comentados en la precisión y el *recall* se hace una relación entre los dos y se saca el *F-Score*3.5

$$L = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{3.5}$$

- ***Accuracy***: Porcentaje de casos en los que el modelo ha acertado en el tiempo total, es decir, cuando acierta en que momentos hay notas o en que momento no. Podemos ver en la ecuación 3.6 que utilizamos tanto los *True positive* como los *True negative* para calcular el porcentaje total.

$$Recall = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.6}$$

- ***Chroma Accuracy***: Es una métrica utilizada solamente en la música, donde mapeamos todas las notas las mapeamos a la misma octava antes de calcular la *accuracy*.

Capítulo 4

Modelo

Este trabajo se centra en la separación de los instrumentos de una canción y su posterior transcripción automática y para ello hemos escogido como punto de partida la librería Demucs[3] y la red de un artículo publicado por Carlos Hernández[56]. En este artículo se utilizaba una red con la misma arquitectura que la red neuronal Deep Salience[43] añadiendo una capa de salida convolucional de (72, 50). Se han adaptado los scripts de pre-procesamiento de la base de datos para poder entrenar y hacer tests con otros archivos y de esta forma indagar más en la separación de fuentes y posterior transcripción de estas.

En el estudio de este trabajo nos vamos a centrar en que notas de cada instrumento suenan en cada momento en una melodía con varios instrumentos y cada uno de estos haciendo sonar varias notas a la vez.

4.1. Separación de Bajos

Para poder realizar los entrenamientos con la red neuronal que permitiese conseguir unos pesos aceptables de esta para poder transcribir las canciones correctamente y posteriormente hacer los tests que indican como de preciso es este modelo necesitábamos separar los bajos de las canciones de nuestra base de datos (Slakh100[14]), ya que nos hemos centrado en este instrumento. Como las canciones enteras estaban compuestas por varios instrumentos polifónicos (varias notas sonando a la vez) se utilizó Anaconda y la librería Demucs, que ya se ha nombrado anteriormente.

Demucs es una red neuronal cuya entrada es una pista de audio en estéreo mezclada y su salida son cuatro fuentes de audio en estéreo divididas en: bajo, voz, batería y otros (incluye el resto de instrumentos de la canción). La arquitectura de la red la podemos ver en la figura 4.5 y consiste en un codificador convolucional, una red BLSTM[47] y un decodificador convolucional, y tanto el codificador como el decodificador están conectados mediante una red U-net y están diseñados de la siguiente forma:

– Codificador: Está compuesto por 6 bloques convolucionales, donde cada uno de ellos tiene un stride = 4, un tamaño de Kernel = 8 y una activación de de los canales de salida ReLu seguido de una convolución con tamaño de Kernel = 1 y una función de activación GLU(Gated linear units). Se aumenta la profundidad del modelo con bajo costo computacional gracias a las convoluciones con tamaño de Kernel y posteriormente se mejora el rendimiento de la red usando GLU. Se utilizan 2 canales para la entrada del codificador y 100 para la salida. Se duplica el número de canales en cada bloque codificador, por lo que a la salida del último bloque hay 3200 canales. El último se encuentra conectado a una red BLSTM de dos capas de tamaño 3200. Posteriormente esta se conecta a una capa lineal que pasa de 6400 a 3200 el tamaño de salida y que esta conectada al primer bloque decodificador.

– Decodificador: Posee la misma estructura que el codificador pero en sentido inverso. Comienza con una convolución de tamaño de la unidad y un Kernel de anchura 3 y una activación ReLU. Para acabar, se usa una convolución transpuesta con ancho de núcleo 8 y paso 4 y activación ReLU. Las fuentes S se sintetizan solo en la capa final, después de todos los bloques decodificadores. La capa final es lineal con canales de salida S·C0, uno para cada fuente (4 canales estéreo en nuestro caso), sin ninguna función de activación adicional. Cada uno de estos canales genera directamente la forma de onda correspondiente.

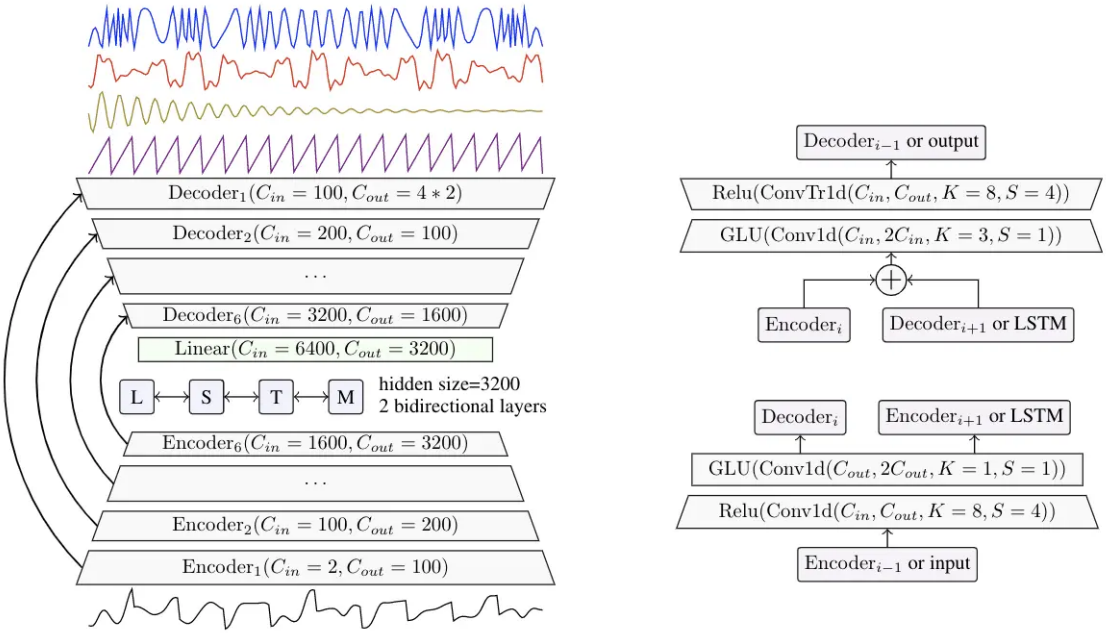


Figura 4.1: Arquitectura de la red neuronal Demucs, utilizada para separar los bajos de las canciones[3]

De las cuatro fuentes de salida se eligió el bajo, ya que en la investigación en la que nos

estamos basando(y que se quiere completar más) se dedujo que el timbre de los instrumentos era muy importante y los instrumentos de percusión no pueden hacer sonar notas musicales.

4.2. Base de datos

Tanto para el entrenamiento, como para la validación y los diversos test hemos utilizado la misma base de datos, la Slakh2100[14].

Esta base de datos esta compuesta por 2100 pistas mezcladas y archivos MIDI adjuntos. Esta dividida en 3 carpetas: test (225 pistas), train (1496 pistas), y validation (375 pistas).

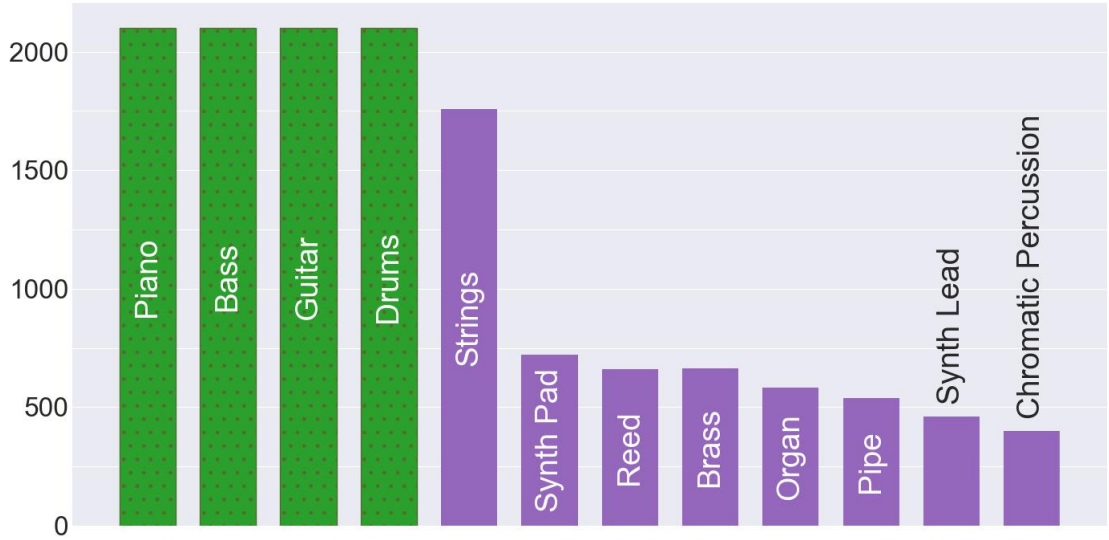


Figura 4.2: Número de canciones en las que aparece cada instrumento [57].

Como podemos ver en la figura 4.2 no todas las canciones tienen todos los instrumentos, o alguna tenía más de un bajo y falseaban el resultado de la red, por lo no se seleccionaron todas. Para esto se tuvo que adaptar un script para seleccionar solo las que tenían un solo bajo, quedando la selección final de esta forma: test (213 pistas), train(1284) y validation(326 pistas). Todas las canciones de esta base de datos son íntegramente instrumentales, en ninguna canción hay un formato vocal.

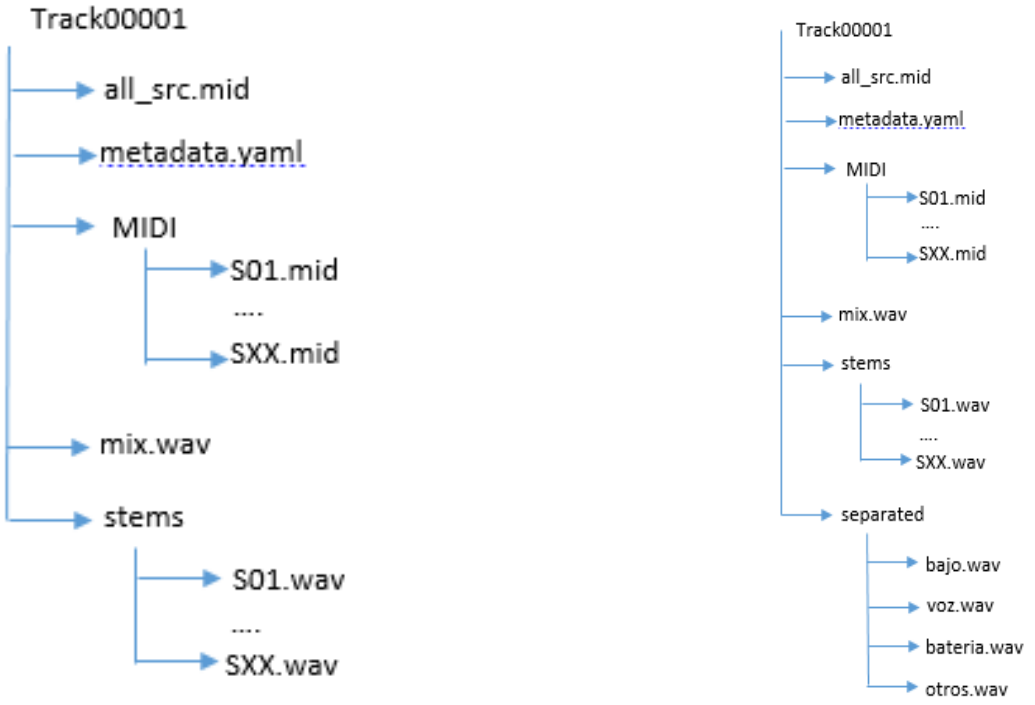
En la imagen 4.3a hay un ejemplo de lo que sería cada carpeta, cada una de ellas corresponde con una canción:

- mix.wav: el archivo de audio en el que está la canción completa con todos los instrumentos.
- all-src.mid: archivo MIDI en el cual está la transcripción original y perfecta de la canción completa con todos los instrumentos.
- directorio MIDI: contiene los archivos .mid separados de cada instrumento que participa

en la canción original. Son estos los que luego se usan para poder comparar el MIDI resultante de hacer los tests con esta red neuronal.

– directorio stems: contiene los archivos .wav separados de cada instrumento que participa en la canción original.

– metadata.yaml: archivo de texto en el que se indica que archivo .wav de la carpeta stems y que archivo .mid de la carpeta MIDI corresponde a cada instrumento, así como: si el audio esta renderizado o no, si es batería, si el midi del instrumento existe y el número de programa al que pertenece(cada instrumento tiene su número de programa).



(a) (b)

Figura 4.3: Organización base de datos: a) Organización de carpetas de la base de datos Slakh 2100, b) Organización de carpetas de la base de datos Slakh 2100 una vez aplicada la red Demucs (canciones separadas).

A cada archivo mix.wav de cada canción de esta base de datos se le aplica la red Demucs mediante una función que se ha realizado, por lo que se separa la pista entera de las cuatro clases de instrumentos que hemos mencionado antes (bajo, voz, batería y otros). Esto crea una subcarpeta, como vemos en la figura 4.3b complementaria a las justo antes descritas dentro de la carpeta de cada canción, la cual se llama separated y contiene los archivos .wav de cada una de las cuatro pistas que hemos separado.

4.3. Modelo de Red Neuronal para la Transcripción Automática

La red propuesta en este trabajo está diseñada para utilizarse en la transcripción automática de música, con el fin de plasmar la canción original en una copia exacta, es decir, saber que nota suena en cada momento de la melodía que timbre tiene. Para este objetivo se escogió una red basada en la arquitectura de la red Deep Salience, a la cual se le agregó a la salida una capa convolucional de dimensiones (72, 50) para facilitar el entrenamiento con archivos MIDI.

4.3.1. Pre-procesamiento: CQTs

Para poder entrenar nuestra red necesitamos haber obtenido la CQT[38] de cada archivo .wav, por eso tenemos que procesarlas antes de entrenar. Para sacar esta CQT de un archivo .wav utilizamos la librería Librosa[41].

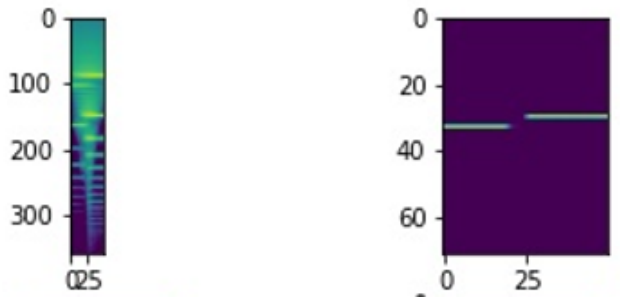


Figura 4.4: CQT obtenida de una parte de un archivo WAV y su pianoroll[42]

Mediante Librosa sacamos la CQT de una parte de la canción (11 ms), como podemos ver en la figura 4.4 (también hemos sacado su pianoroll), y conseguimos almacenar todos los armónicos de un trozo de canción elegido de forma aleatoria, asegurándonos que no es ni el principio ni el final de la canción para evitar que haya trozos vacíos. Para poder hacer esto necesitaremos algunos elementos:

- Ubicación de la canción: para poder calcular la CQT necesitamos el .wav de la canción.
- Pianoroll: lo necesitamos por dos motivos. El primero de ellos es que hay que saber su longitud porque este es más corto que los archivos de audio (.wav), ya que estos tienen el silencio del final, el cual no esta en el archivo MIDI. Y como hemos mencionado antes no queremos coger estos silencios, por lo que conocer esta longitud nos permite coger correctamente la parte de la canción para calcular la CQT. El segundo motivo es porque cuando calculamos la CQT de la parte de la canción hay que ver si el pianoroll esta vacío o no. Ya que el pianoroll es una representación de las notas que han sonado y en que

tiempo, si en un instante de tiempo no esta sonando ninguna nota (lo cuál sucede mucho) el pianoroll está vacío. Necesitamos que haya silencios en nuestro entrenamiento, pero si hay demasiados, la red se pensará que es todo el rato así y fácilmente generalizará a dejar todo en silencio toda la canción, por lo que evitamos esto poniendo un porcentaje máximo a los pianorolls que vacíos que pueda coger.

- Margen: cuando calculamos la CQT de una parte de la canción dejamos un margen a los lados, y luego calculamos la CQT en medio de este margen. Esto lo hacemos porque para calcular un trozo de CQT es necesario mas parte de canción antes y después, por eso dejamos más canción a los lados.

- Tamaño del batch y cantidad de huecos vacíos: Con esto conocemos las ventanas vacías que habrá en el batch y el tamaño que tiene este en cada momento, por lo que podremos conocer el porcentaje de ventanas vacías que habrá en cada batch y si este supera una cantidad, descartaremos el resto de ventanas vacías que le lleguen.

Tenemos una variable 'error' la cual se activa si sucede una de estas 2 condiciones. Si en la lista de elementos de las amplitudes muestreadas por librosa es 0, esto sucede cuando algún archivo de audio no se puede leer (porque esta corrompido), entonces lo descartamos para evitar errores en el entrenamiento. El segundo caso ya comentado es que nos llegue un trozo de canción vacío y ya se haya alcanzado el número máximo de trozos vacíos en ese batch.

Esta función nos devuelve el valor de los coeficientes CQT procesados, el índice del trozo de canción procesado, el error y la cantidad de trozos de canción vacíos en ese batch.

4.3.2. Arquitectura de la Red

La red Deep Salience[43] modificada que se usa en este trabajo tiene unas dimensiones de entrada de (360, 50, 6), las cuales son: la dimensión de frecuencia (360), que son 6 octavas divididas cada una de ellas en 60 partes o bins; el número de frames (50), cada uno de ellos son unos 11 milisegundos de la canción; los armónicos (6), el fundamental, C1 a C7, un subarmónico y 4 armónicos superiores.

Tiene 6 capas convolucionales intermedias organizadas de la siguiente manera:

- Dos capas de 128 y 64 filtros respectivamente de tamaño 5x5 que cubren un semitono, ya que cada octava se divide en 60 bins(cada semitono son 5 bins).

- Dos capas de 64 filtros de tamaño 3x3.

- Una quinta capa que tiene un filtro de 70x3, para cubrir mas de una octava(14 semitonos).

– Una última capa convolucional con un filtro de 5x1 y un stride de 5x1, cuya función es la de hacer una convolución de las 5 frecuencias mas cercanas a las notas reales.

Como se puede ver en la figura 4.5 el modelo tiene una salida de (72, 50), por lo que se consiguen 12 semitonos por octava, que corresponde a las notas reales en el intervalo fundamental (C1 a C7).

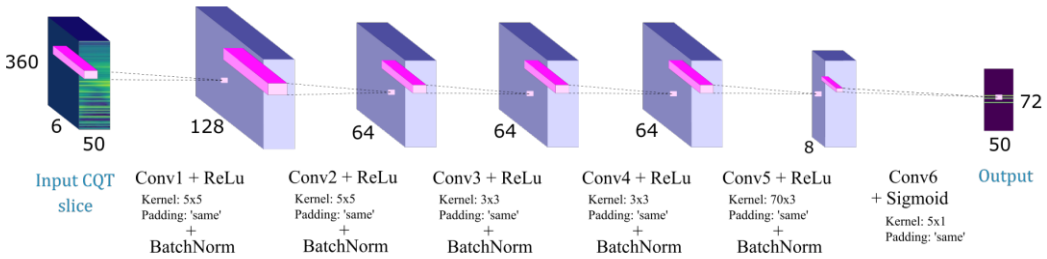


Figura 4.5: Arquitectura de la red neuronal Deep Salience con una última capa convolucional añadida[56]

Se ha empleado un optimizador Adam con un ratio de aprendizaje de 0.001, β_1 de 0.9, β_2 de 0.999 y ϵ de 10^{-7}, para reducir el error. La función de pérdida es la de reducción de la entropía cruzada (ecuación 4.1). Donde y_{pred} es la predicción del modelo, y_{real} es el *ground truth* y L es el error que devuelve la función.

$$L(y_{real}, y_{pred}) = -y_{real} \log(y_{pred}) - (1 - y_{real}) \log(1 - y_{pred}) \tag{4.1}$$

4.4. Entrenamiento del Modelo

Una vez descrita la arquitectura que posee la red, se van a describir los pasos que se llevaron a cabo para poder entrenar el modelo y sacar los pesos de la red necesarios que luego se utilizarían para hacer los tests necesarios. En la figura 4.6 se puede ver un esquema con los pasos que hemos llevado a cabo.

Cuando ya se tiene la red predefinida y la base de datos separada en test, train y validation, y en cada una de las pistas aplicada la red Demucs, es decir, que se tiene el archivo .wav de todos los bajos separados de las canciones, se procede a guardar en dos arrays las rutas de cada archivo del bajo .mid y .wav que tiene cada canción.

Lo primero será elegir el tamaño de batch, siendo esta vez 32. A continuación se selecciona aleatoriamente una pista de la lista, de la cual se escogerá de forma aleatoria una parte de la pista y se procesará su señal de audio, calculando la CQT. De este trozo de la pista se sacaran los 50 frames que corresponden a la entrada de la red. Además se obtiene el pianorol que corresponda a ese trozo de la entrada para obtener el ground truth. Y se repite el proceso las

veces que se le haya indicado en el tamaño del batch. Y por último se entrena la red tantas veces como se le indique en el número de épocas, que lo hemos definido como 10000.

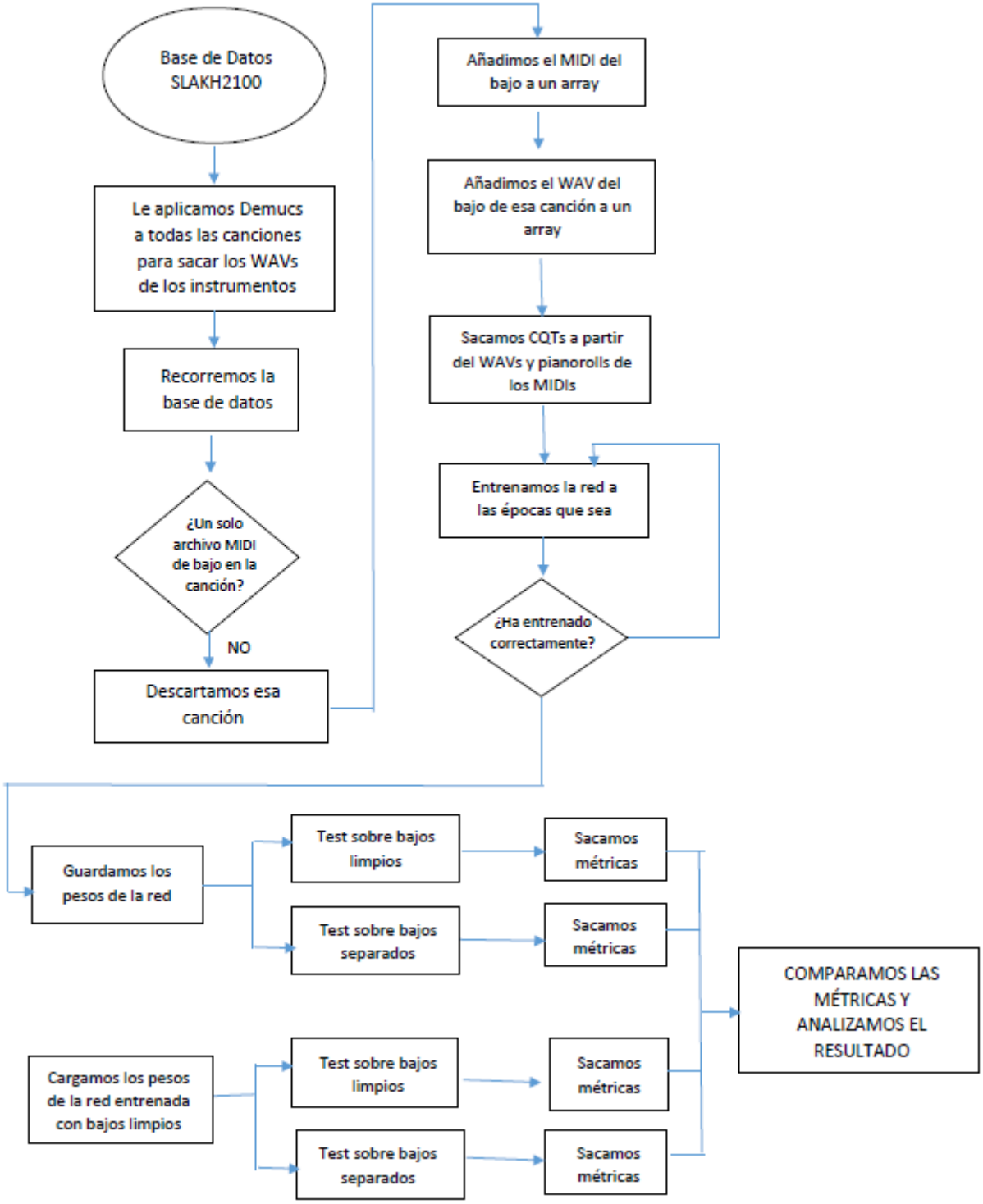


Figura 4.6: Diagrama de flujo del trabajo con todos los pasos que se han llevado a cabo.

Capítulo 5

Experimentos y Resultados

Nuestros resultados se basan en detectar las notas en un frame, es decir, si en ese momento de tiempo está sonando una nota o no. Hemos dividido estos experimentos en 5 apartados distintos:

- **Entrenamiento con bajos separados**.
- **Test con red preentrenada con bajos limpios sobre bajos limpios**: para ver el punto de partida de esta red y saber si mejoramos las mértricas o no.
- **Test sobre bajos separados con red preentrenada con bajos limpios**.
- **Test sobre bajos limpios con red preentrenada con bajos separados**.
- **Test sobre bajos separados con red preentrenada con bajos separados**.
- **Test sobre modelo de transcripción multi-instrumento(MT3)**.

Compararemos los resultados de cada uno de estos tests y veremos cómo los resultados mejoran al entrenar con los bajos separados.

5.1. Test de Bajos limpios con Red Preentrenada con Bajos limpios (baseline)

El primer test que realizamos es para poder ver el punto de partida. Con los pesos inicializados que se obtuvieron en el trabajo que se utiliza como baseline[56] por la red entrenada con bajos limpios, es decir sin haber sido separados por la librería demucs a partir de la canción conjunta.

Los .wav de estos bajos se encuentran grabados por separado en una carpeta en la base de datos de Slakh. Se transcribe el .mid a partir de estos archivos de audio y se compara con el .mid real que también nos proporciona Slakh.

Las métricas analizadas con las que se ha cuantificado la efectividad de la red han sido: *Chromma accuracy* (CAcc), *Accuracy* (Acc), *Precision* (P), *Recall* (R) y *F-score* (F), valorando sobre todo el F-score, ya que es una mezcla entre la precisión y el recall (para el resto de tests será igual).

Como se puede ver en la tabla 5.1 se consigue un porcentaje en el F-Score superior a 90, lo cuál es un resultado muy aceptable. Debajo tenemos la figura 5.1, en la que se puede ver el diagrama de cajas del test de las métricas obtenidas.

Tabla 5.1: Métricas del test con bajos limpios de la red preentrenada con bajos limpios

Base de datos	Instrumento	CAcc	Acc	R	P	F
Slakh2100	Bajo	87.83	87.69	88.97	91.54	90.16

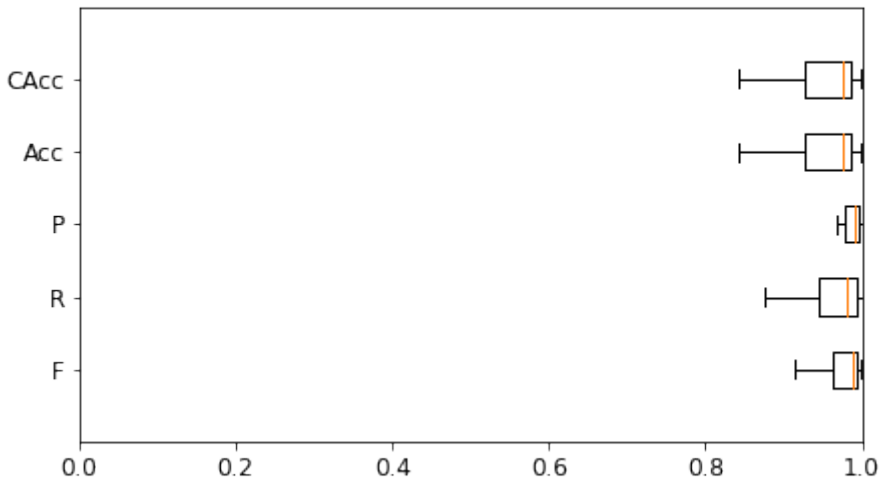


Figura 5.1: test en bajos limpios (no separados).

5.2. Test de Bajos separados con Red Preentrenada con Bajos limpios

En este segundo test, pero todavía para la red preentrenada con bajos limpios, hemos utilizado como entrada de audio los archivos de audio, de aquellos que hemos separado para el test, de los bajos que hemos obtenido separando las canciones en pistas mediante la libreria Demucs.

Estos resultados son los que utilizaremos como iniciales para compararlos con los que salgan de hacer el test de la red ya entrenada con estos mismos audios (con los audios de entrenamiento).

Como podemos ver en la tabla 5.2 y en la figura 5.2 nos ha salido una F-score con un

porcentaje aproximado de 71, por lo que ya apreciamos que hay una caída notable del F-score debido al ruido que se introduce en estos archivos de audio al ser separados.

Tabla 5.2: Métricas del test con bajos separados de la red preentrenada con bajos limpios

Base de datos	Instrumento	CAcc	Acc	R	P	F
Slakh2100	Bajo	65.25	59.7	70.97	72.22	71.55

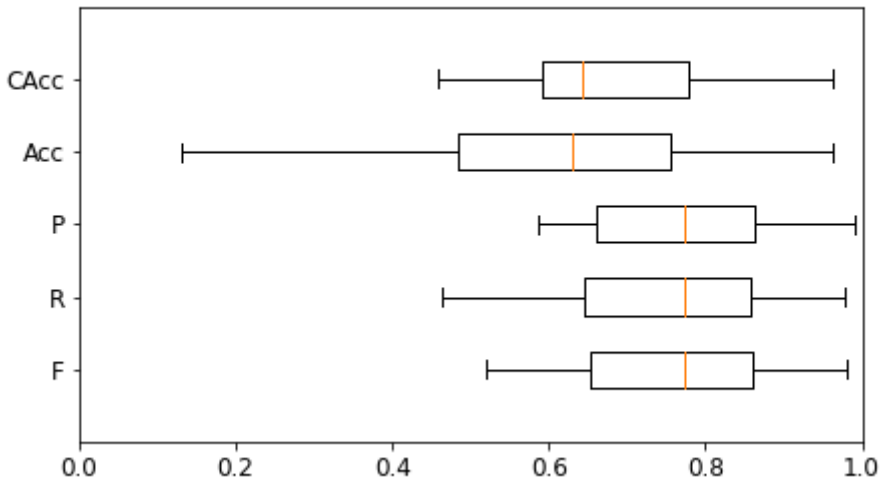


Figura 5.2: test en bajos separados por librería Demucs.

5.3. Test de Bajos limpios con Red Preentrenada con Bajos separados

En la segunda parte de este trabajo ya se realizan los tests con la red que hemos entrenado con los bajos separados de Slakh2100 mediante la librería Demucs, así veremos si la red es capaz de mejorar resultados con respecto a la primera parte, la cuál daba unos resultados buenos con audios limpios, pero no tan buenos con audios separados debido al ruido introducido en estos al separar en varias pistas.

Los resultados de esta parte van a estar divididos en dos. El primero que es realizar un test con las canciones de los bajos limpio. Este test lo cuantificaremos de la misma forma que el anterior, así podremos comparar resultados y ver si ha mejorado, empeorado o no ha afectado, lo cuál no es el objetivo principal de este trabajo, ya que es sobre audios ya separados sin haberse introducido ruido en ellos por la librería Demucs (ya que venían separados).

Pese a que no sea nuestro objetivo principal, también es interesante ver si sobre los audios limpios no ha empeorado mucho. Ya que si conseguimos mantener resultados respecto al otro entrenamiento, se podría usar esta red preentrenada con estos archivos para los mismos usos que la otra y añadiendo aquellos casos en los que se tengan archivos separados.

Como se puede ver en la tabla 5.3 y en la figura 5.3 no solo no empeora, sino que mejora bastante los resultados, subiendo el F-score hasta un porcentaje cercano al 97.

Tabla 5.3: Métricas del test con bajos limpios de la red preentrenada con bajos separados

Base de datos	Instrumento	CAcc	Acc	R	P	F
Slakh2100	Bajo	95.75	95.69	96.58	99.02	97.13

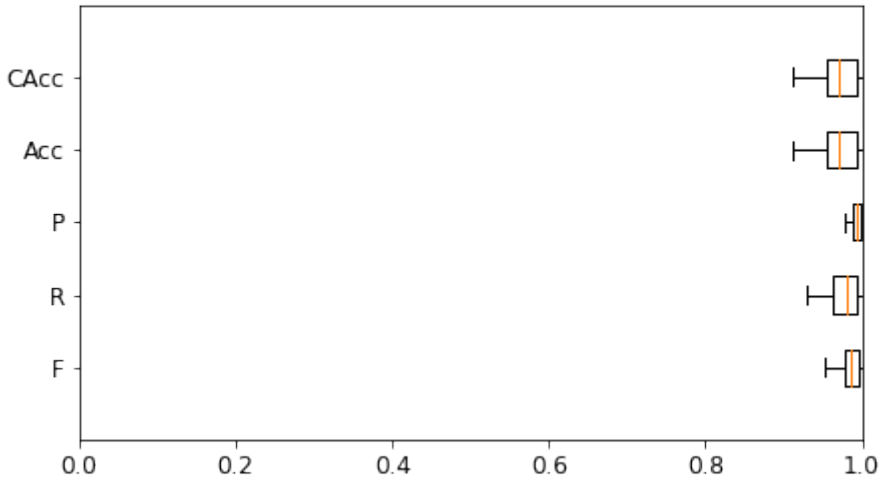


Figura 5.3: test en bajos limpios (no separados).

5.4. Test de Bajos separados con Red Preentrenada con Bajos separados

El segundo test de esta segunda parte del experimento con la red entrenada con bajos separados consiste en realizar un test a esta con los archivos de audio de los bajos separados, para ver si es viable separar una canción y luego poder reproducir el bajo de esta en un archivo MIDI.

Viendo la tabla5.4 y la imagen5.4 que muestra el diagrama de cajas de las métricas, vemos que el F-score sube a casi un porcentaje de 84, por lo que se puede ver que ha mejorado considerablemente pese al ruido que se introduce al separar la canción.

Tabla 5.4: Métricas del test con bajos separados de la red preentrenada con bajos separados

Base de datos	Instrumento	CAcc	Acc	R	P	F
Slakh2100	Bajo	76.23	74.21	82.02	86.54	83.38

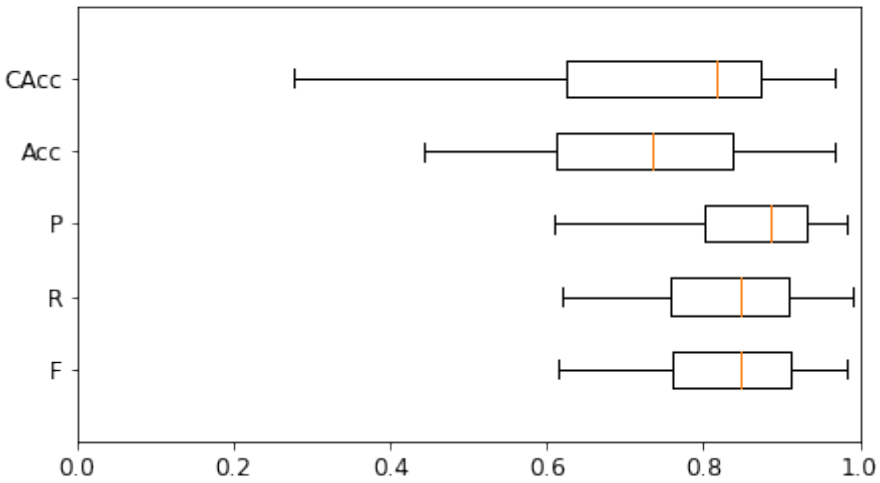


Figura 5.4: test en demucs

5.5. Modelo de Transcripción Multi-Instrumento (MT3)

Como ultimo experimento para poder comparar el resultado de nuestra red con los actuales modelos que están en el mercado hemos vuelto a separar la canción entera, pero esta vez en lugar de hacerlo con la librería Demucs, lo hemos hecho con la red [1] MT3[48] de Google mediante el Google Colab. Como esta red transcribe el MIDI de la canción entera, por lo que hemos tenido que separar este archivo mediante una función que coge solo el audio del bajo.

Este archivo .mid lo hemos pasado por un programa que nos saca el pianoroll de esta pista, lo podemos ver en la figura5.6d y a continuación lo vamos a comparar con el obtenido de los demás test para poder saber cómo funcionan nuestros modelos de una manera más objetiva.

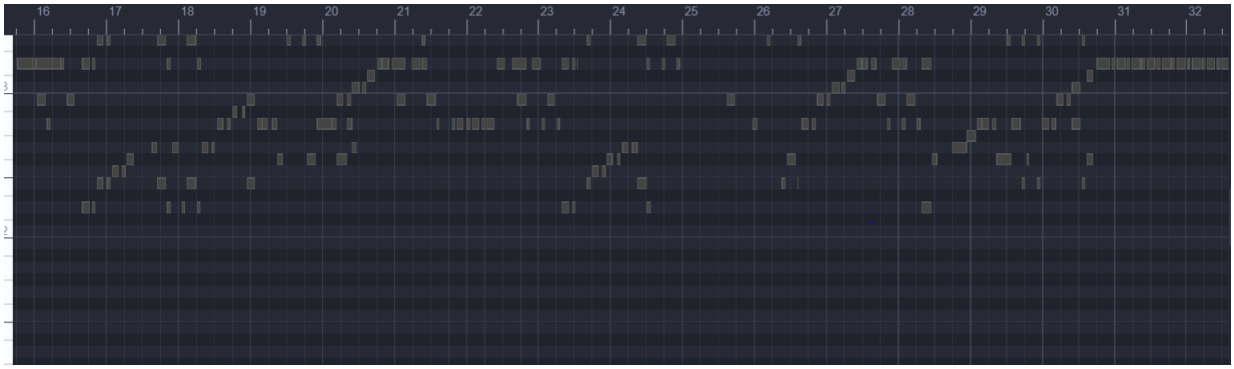


Figura 5.5: Pianoroll obtenido del archivo MIDI sacado de la separación de la canción mediante la red MT3.

[1] `https://colab.research.google.com/github/magenta/mt3/blob/main/mt3/colab/music_transcription_with_transformers.ipynb`, último acceso Mayo 2022

5.6. Comparativa

Para comparar los resultados obtenidos en los tests hemos hecho una tabla5.5 en la que juntamos los resultados de cada test para poder analizarlos con más facilidad.

Se puede ver a simple vista que los mejores resultados de nuestra red, da igual cual haya sido el entrenamiento, se dan para los tests sobre bajos limpios, lo cual era de esperar, ya que al separar las canciones se introduce un ruido adicional que supone un problema a la hora de detectar las notas y su tono.

Pese a que los dos mejores resultados son con diferentes entrenamientos y sobre bajos limpios, esta claro que el mejor de los dos ha sido el de la red entrenada con bajos separados, con un porcentaje de 97 frente al del 90 de la red entrenada con bajos limpios.

Con este resultado podemos decir que a parte de mejorar al transcribir canciones con algo de ruido añadido, debido a su separación previa, que era nuestro principal objetivo, también mejoramos la función principal de la red anterior.

Como podemos ver el porcentaje del F-score de la transcripción de bajos con ruido ha mejorado en un 12 %. Aunque no parezca mucho, este porcentaje esta muy bien, ya que la mayoría de tiempo al ser silencios, los suele hacer bien, por lo que subir un 12 % respecto al total del tiempo de la canción es la diferencia entre que suenen muchas notas innecesarias y con tono diferente al que debería.

Tabla 5.5: Tabla comparativa de métricas de cada uno de los tests realizados en este trabajo: Los dos primeros test están hechos con la red entrenada por bajos limpios, los dos últimos son sobre la red entrenada con bajos separados.

Base de datos	Bajo de entrenamiento	Bajo de entrada	CAcc	Acc	R	P	F
Slakh2100	Bajos limpios	Bajos limpios	87.83	87.69	88.97	91.54	90.16
		Bajos separados	65.25	59.7	70.97	72.22	71.55
	Bajos separados	Bajos limpios	95.75	95.69	96.58	99.02	97.13
		Bajos separados	76.23	74.21	82.02	86.54	83.38

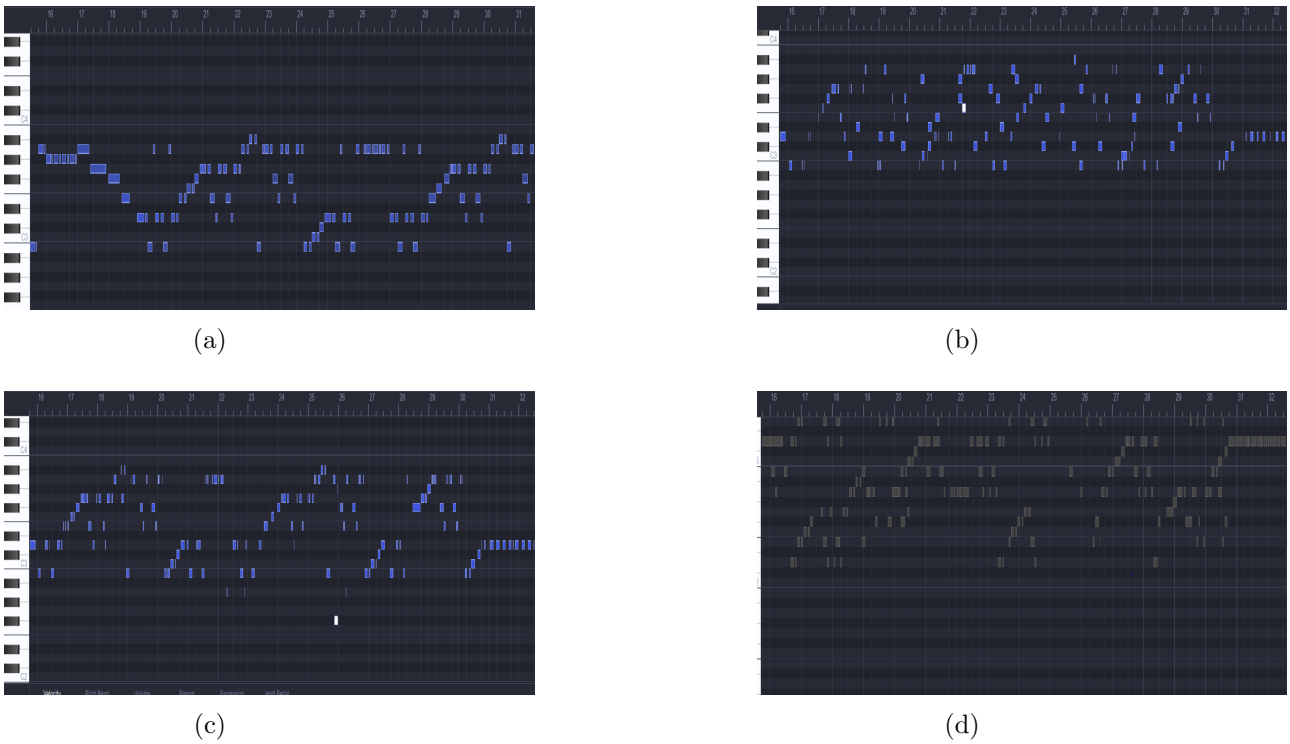
A parte de comparar las métricas entre los diferentes entrenamientos de la red, también sacamos cuatro pianorolls de una pista de bajo de una canción. Los cuatro pianorolls sacados son con cuatro archivos .mid de una misma pista: el .mid de la pista real que ya nos viene en la base de datos, el .mid que hemos sacado de filtrar el bajo del MIDI resultante de separar la canción con MT3; y los otros dos archivos que hemos comparado son aquellos resultantes de separar la canción con la librería Demucs, sacamos el .mid del bajo con la red basada en Deep Salience entrenada con bajos limpios y y la entrenada con bajos separados.

Se busca que los pianorolls resultantes de las canciones separadas sean lo más parecido al del real. Y como se puede ver en la figura5.6, el más parecido al primero de los cuatro que es el real (5.6a), es el obtenido de la red MT3 (5.6d), el cual podemos ver que mantiene la

estructura de las notas y acierta los onsets y duración de estas bastante correctamente.

Se puede ver también que de la red basada en Deep Salience, de los dos diferentes entrenamientos, el pianoroll que es mucho más parecido al real es el que conseguimos del .mid obtenido del entrenamiento con bajos separados (5.6c), y se ve que se consigue de una forma aceptable la misma estructura de las notas y los onsets de estas se aciertan igual, pero esta vez la duración de los tiempos es bastante incorrecta y cuando hay una nota larga, esta la representa con un espacio en medio como si fuesen dos distintas (o más).

Por último, el pianoroll obtenido de la red entrenada con bajos limpios (5.6b) es muy malo, y se puede observar a simple vista que no acierta ni las notas, ni el tiempo de estas ni la duración, por lo que podemos decir que hemos conseguido mejorar de forma muy notable la transcripción de fuentes musicales separadas.



(a) (b)

(c) (d)

Figura 5.6: Pianoroll: a) Real, b) Red basada en Deep Salience entrenada con bajos limpios , c) Red basada en Deep Salience entrenada con bajos separados d) MT3

5.7. Discusión

Vistos los resultados obtenidos en la tabla 5.5, como es de esperar, la red trabaja mejor con bajos limpios dado que al separar los bajos se introduce una pérdida adicional en este caso, dada por la efectividad del modelo utilizado para la separación, demucs.

Se podría decir que el trabajo ha salido como se esperaba, mejorando en buena medida la transcripción de archivos de audio separados previamente por la librería Demucs. Además

hemos comprobado que la transcripción sobre bajos limpios se mejora si entrenamos con los bajos una vez han sido separados, en lugar de hacerlo con los bajos limpios, lo cual era algo no se esperaba. Gracias a esto esta red con estos archivos de entrada para el entrenamiento sería superior a la red preentrenada en la que nos basamos como baseline para este trabajo.

La red MT3 de Google se puede ver que es más versátil que la nuestra, ya que no necesita separar los instrumentos para transcribirlos, pero era de esperar, ya que esta elaborada por un equipo de especialistas y entrenada con muchísimos más archivos y horas de música en equipos mucho más potentes que los que teníamos a nuestra disposición.

Capítulo 6

Conclusiones y Líneas Futuras

6.1. Conclusiones

En este trabajo nos hemos centrado en la separación de fuentes y la transcripción musical de melodías polifónicas, basando nuestro trabajo en un estudio de transcripción musical de melodías monofónicas.

El objetivo era conseguir una transcripción musical de buena calidad de bajos separados previamente de la canción completa y posteriormente compararlo con las métricas resultantes de transcribir esta misma pista de audio, del bajo separado, con los pesos de la red que se tenían previamente.

Para ello se llevó a cabo una separación de fuentes a la base de datos de Slakh2100 y posteriormente a un entrenamiento de la red con esos bajos separados para conseguir nuevos pesos en la red y aplicar los test.

Una vez aplicados los test se ha podido comprobar que el resultado de trabajo ha sido un éxito, ya que no solo mejora considerablemente la transcripción musical sobre bajos separados, haciendo que responda mucho mejor al ruido introducido al separar las diferentes pistas, sino que además se ha mejorado igualmente la transcripción musical sobre pistas limpias ya grabadas por separado sin ruido.

6.2. Líneas Futuras

Por otro lado, aunque el resultado del trabajo sea satisfactorio y sirva como base para generar un modelo nuevo de transcripción musical para fuentes separadas, habría varios puntos a mejorar y otros múltiples objetivos sobre los que trabajar.

Pese a que se han conseguido buenos resultados, siempre se puede mejorar y este debería ser un objetivo a futuro, por lo que se podrían buscar nuevas maneras de mejorar este entrenamiento, o incluso la red. Algunas formas pueden ser: conseguir una base de datos mayor, intentar buscar hiper-parámetros de la red que nos lleven a mejores entrenamientos

o incluso intentar modificar la red.

También se podría avanzar por la mejora de la separación de fuentes, intentando buscar un modelo que no introduzca tanto ruido al separar. Para ello se podría estudiar como los timbres afectan a la separación y transcripción e intentar mejorarlo, ya que la separación musical depende de la frecuencia pero no tanto del timbre, es decir, dos instrumentos tocando a la misma frecuencia, lo separa como uno solo.

Otra posibilidad de avance sería entrenar otros instrumentos, y en un futuro buscar una red que entrenándola con muchos instrumentos, consiga niveles aceptables de acierto. O incluso desarrollar nuevos modelos que transcriban sin necesidad de separar todos los instrumentos a la vez,como hemos visto que hacía el MT3.

Como se puede ver en el futuro de la transcripción musical aun se tiene mucho por delante en lo que respecta a la inteligencia artificial, y cada vez se van descubriendo hallazgos nuevos que permiten crear herramientas para mejorar esta.

6.3. Valoración personal

En lo personal, gracias a este trabajo he podido profundizar mucho más en campos que durante el grado se mencionan y se enseñan vagamente en alguna optativa o en parte de ellas.

Si que es cierto que en este grado se dan algunas asignaturas de programación, pero están la mayoría relacionadas con la robótica, y realizando este trabajo he podido adentrarme y aprender más en los temas relacionados con redes neuronales y con la programación relacionada con estas, ya que el trabajo del que se partía era un artículo basado en un código muy elaborado y que llevaba tiempo entenderlo, así como aprender a usar los comandos de CMD para poder aplicar la librería de separación de fuentes e instalar todos los paquetes de Anaconda, al igual que mejorar mucho los conocimientos sobre Python y conocer GitHub.

En definitiva que gracias a este trabajo he podido aprender mucho más sobre estos programas y sobre programación que necesitaré en un futuro próximo, así como muchos conceptos relacionados con la música y el procesamiento de audio. Estas dos cosas juntas han hecho que se abran nuevas posibilidades en mi futuro, ya sea investigando, trabajando o simplemente dedicándole una parte de mi tiempo libre.

Capítulo 7

Bibliografía

[1] Habshah Midi, Saroje Kumar Sarkar, and Sohel Rana. Collinearity diagnostics of binary logistic regression model. *Journal of interdisciplinary mathematics*, 13(3):253–267, 2010.

[2] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[3] Alexandre Défossez, Nicolas Usunier, Léon Bottou, and Francis Bach. Demucs: Deep extractor for music sources with extra unlabeled data remixed. *arXiv preprint arXiv:1909.01174*, 2019.

[4] Ivet Challenger-Pérez, Yanet Díaz-Ricardo, and Roberto Antonio Becerra-García. El lenguaje de programación python. *Ciencias Holguín*, 20(2):1–13, 2014.

[5] Gautam Kamath and Christos Tzamos. Anaconda: A non-adaptive conditional sampling algorithm for distribution testing. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 679–693. SIAM, 2019.

[6] Pierre Raybaut. Spyder-documentation. *Available online at: pythonhosted. org*, 2009.

[7] Bernadette M Randles, Irene V Pasquetto, Milena S Golshan, and Christine L Borgman. Using the jupyter notebook as a tool for open science: An empirical study. In *2017 ACM/IEEE Joint Conference on Digital Libraries (JCDL)*, pages 1–2. IEEE, 2017.

[8] Bo Pang, Erik Nijkamp, and Ying Nian Wu. Deep learning with tensorflow: A review. *Journal of Educational and Behavioral Statistics*, 45(2):227–248, 2020.

[9] Alan Mathison Turing. Mind. *Mind*, 59(236):433–460, 1950.

[10] Patrick Henry Winston. *Artificial intelligence*. Addison-Wesley Longman Publishing Co., Inc., 1984.

[11] Agostino De Santis, Bruno Siciliano, Alessandro De Luca, and Antonio Bicchi. An atlas of physical human–robot interaction. *Mechanism and Machine Theory*, 43(3):253–270, 2008.

[12] https://gestion.pe/blog/analizandotusinversiones/2018/02/el-crecimiento-de-la-inteligencia-artificial-que-genera-nvidia.html/. Último acceso: septiembre 2022.

[13] Mark B Taylor. Stilts-a package for command-line processing of tabular data. In *Astronomical Data Analysis Software and Systems XV*, volume 351, page 666, 2006.

[14] Ethan Manilow, Gordon Wichern, Prem Seetharaman, and Jonathan Le Roux. Cutting music source separation some slakh: A dataset to study the impact of training data quality and quantity. In *2019 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, pages 45–49. IEEE, 2019.

[15] P Russel Norvig and S Artificial Intelligence. A modern approach. *Prentice Hall Upper Saddle River, NJ, USA: Rani, M., Nayak, R., & Vyas, OP (2015). An ontology-based adaptive personalized e-learning system, assisted by software agents on cloud storage. Knowledge-Based Systems*, 90:33–48, 2002.

[16] Onsets and frames: Dual-objective piano transcription. https://www.salesforce.com/mx/blog/2018/7/Machine-Learning-y-Deep-Learning-aprende-las-diferencias.html.

[17] Entrenamiento de redes neuronales: mejorando el gradiente descendente. http://www.cs.us.es/~fsancho/?e=165. Último acceso: septiembre 2022.

[18] Sholom M Weiss and Casimir A Kulikowski. *Computer systems that learn: classification and prediction methods from statistics, neural nets, machine learning, and expert systems*. Morgan Kaufmann Publishers Inc., 1991.

[19] Su-Hyun Han, Ko Woon Kim, SangYun Kim, and Young Chul Youn. Artificial neural network: understanding the basic concepts without mathematics. *Dementia and Neurocognitive Disorders*, 17(3):83–89, 2018.

[20] https://www.xeridia.com/blog/redes-neuronales-artificiales-que-son-y-como-se-entrenan-parte-i. http://www.cs.us.es/~fsancho/?e=165. Último acceso: septiembre 2022.

[21] Redes neuronales. https://bootcampai.medium.com/redes-neuronales-13349dd1a5bb#:~:text=Funciones%20de%20activaci%C3%B3n,que%20permitir%C3%A1%20reconstruir%20o%20predecir. Último acceso: septiembre 2022.

[22] Timothy P Lillicrap, Adam Santoro, Luke Marris, Colin J Akerman, and Geoffrey Hinton. Backpropagation and the brain. *Nature Reviews Neuroscience*, 21(6):335–346, 2020.

[23] Arquitectura de redes neuronales. `https://interactivechaos.com/es/manual/tutorial-de-machine-learning/arquitectura-de-redes-neuronales`.

[24] Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[25] Red neuronal convolucional cnn. `https://www.diegocalvo.es/red-neuronal-convolucional/`.

[26] Judith Brown. Calculation of a constant q spectral transform. *Journal of the Acoustical Society of America*, 89(1):425–434, January 1991.

[27] David Erroz Arroyo. Visualizando neuronas en redes neuronales convolucionales. 2019.

[28] Explain pooling layers: Max pooling, average pooling, global average pooling, and global max pooling. `https://androidkt.com/explain-pooling-layers-max-pooling-average-pooling-global-average-pooling-and-global`

[29] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[30] The illustrated transformer. `https://jalammar.github.io/illustrated-transformer/`. Último acceso: septiembre 2022.

[31] Ethan Manilow, Prem Seetharman, and Justin Salamon. *Open Source Tools & Data for Music Source Separation*. https://source-separation.github.io/tutorial, 2020.

[32] Craig Macartney and Tillman Weyde. Improved speech enhancement with the wave-u-net. *arXiv preprint arXiv:1811.11307*, 2018.

[33] Yun-Ning Hung and Alexander Lerch. Multitask learning for instrument activation aware music source separation. *arXiv preprint arXiv:2008.00616*, 2020.

[34] Romain Hennequin, Anis Khlif, Felix Voituret, and Manuel Moussallam. Spleeter: a fast and efficient music source separation tool with pre-trained models. *Journal of Open Source Software*, 5(50):2154, 2020.

[35] Andreas Jansson, Eric Humphrey, Nicola Montecchio, Rachel Bittner, Aparna Kumar, and Tillman Weyde. Singing voice separation with deep u-net convolutional networks. 2017.

[36] Ethan Manilow, Prem Seetharaman, and Bryan Pardo. Simultaneous separation and transcription of mixtures with multiple polyphonic and percussive instruments. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 771–775. IEEE, 2020.

[37] Johanne Hizanidis, Nikos E Kouvaris, Gorka Zamora-López, Albert Díaz-Guilera, and Chris G Antonopoulos. Chimera-like states in modular neural networks. *Scientific reports*, 6(1):1–11, 2016.

[38] Judith Brown. Calculation of a constant q spectral transform. *Journal of the Acoustical Society of America*, 89(1):425–434, January 1991.

[39] Christopher A Brown and Sid P Bacon. Fundamental frequency and speech intelligibility in background noise. *Hearing research*, 266(1-2):52–59, 2010.

[40] Daniel Griffin and Jae Lim. Signal estimation from modified short-time fourier transform. *IEEE Transactions on acoustics, speech, and signal processing*, 32(2):236–243, 1984.

[41] Brian McFee, Colin Raffel, Dawen Liang, Daniel P Ellis, Matt McVicar, Eric Battenberg, and Oriol Nieto. librosa: Audio and music signal analysis in python. In *Proceedings of the 14th python in science conference*, volume 8, pages 18–25. Citeseer, 2015.

[42] Shigeki Sagayama, Keigo Takahashi, Hirokazu Kameoka, and Takuya Nishimoto. Specmurt anasylis: A piano-roll-visualization of polyphonic music signal by deconvolution of log-frequency spectrum. In *ISCA Tutorial and Research Workshop (ITRW) on Statistical and Perceptual Audio Processing*, 2004.

[43] Rachel M Bittner, Brian McFee, Justin Salamon, Peter Li, and Juan Pablo Bello. Deep salience representations for f0 estimation in polyphonic music. In *ISMIR*, pages 63–70, 2017.

[44] Rachel M Bittner, Justin Salamon, Mike Tierney, Matthias Mauch, Chris Cannam, and Juan Pablo Bello. Medleydb: A multitrack dataset for annotation-intensive mir research. In *ISMIR*, volume 14, pages 155–160, 2014.

[45] Curtis Hawthorne, Erich Elsen, Jialin Song, Adam Roberts, Ian Simon, Colin Raffel, Jesse Engel, Sageev Oore, and Douglas Eck. Onsets and frames: Dual-objective piano transcription. In *Proceedings of the 19th International Society for Music Information Retrieval Conference, ISMIR 2018, Paris, France, 2018*, 2018.

[46] Rainer Kelz, Matthias Dorfer, Filip Korzeniowski, Sebastian Böck, Andreas Arzt, and Gerhard Widmer. On the potential of simple framewise approaches to piano transcription. *CoRR*, abs/1612.05153, 2016.

[47] Santiago Fernández, Alex Graves, and Jürgen Schmidhuber. Phoneme recognition in timit with blstm-ctc. *arXiv preprint arXiv:0804.3269*, 2008.

[48] Josh Gardner, Ian Simon, Ethan Manilow, Curtis Hawthorne, and Jesse H. Engel. MT3: multi-task multitrack music transcription. *CoRR*, abs/2111.03017, 2021.

[49] Curtis Hawthorne, Andriy Stasyuk, Adam Roberts, Ian Simon, Cheng-Zhi Anna Huang, Sander Dieleman, Erich Elsen, Jesse Engel, and Douglas Eck. Enabling factorized piano music modeling and generation with the maestro dataset. *arXiv preprint arXiv:1810.12247*, 2018.

[50] Michael FM Engels, Theo Thielemans, Danny Verbinnen, Jan P Tollenaere, and Rudi Verbeeck. Cerberus: a system supporting the sequential screening process. *Journal of chemical information and computer sciences*, 40(2):241–245, 2000.

[51] Qingyang Xi, Rachel M Bittner, Johan Pauwels, Xuzhou Ye, and Juan Pablo Bello. Guitarset: A dataset for guitar transcription. In *ISMIR*, pages 453–460, 2018.

[52] John Thickstun, Zaid Harchaoui, and Sham Kakade. Learning features of music from scratch. *arXiv preprint arXiv:1611.09827*, 2016.

[53] Bochen Li, Xinzhao Liu, Karthik Dinesh, Zhiyao Duan, and Gaurav Sharma. Creating a multitrack classical music performance dataset for multimodal music analysis: Challenges, insights, and applications. *IEEE Transactions on Multimedia*, 21(2):522–535, 2018.

[54] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, Peter J Liu, et al. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21(140):1–67, 2020.

[55] Josh Gardner, Ian Simon, Ethan Manilow, Curtis Hawthorne, and Jesse Engel. Mt3: Multi-task multitrack music transcription. *arXiv preprint arXiv:2111.03017*, 2021.

[56] Carlos Hernandez-Olivan, Ignacio Zay Pinilla, Carlos Hernandez-Lopez, and Jose R. Beltran. A comparison of deep learning methods for timbre analysis in polyphonic automatic music transcription. *Electronics*, 10(7), 2021.

[57] Slakh2100 (synthesized lakh dataset). `https://paperswithcode.com/dataset/slakh2100`. Último acceso: septiembre 2022.

[58] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[59] Padding, strides, max-pooling y stacking en las redes convolucionales. https://www.codificandobits.com/blog/padding-strides-maxpooling-stacking-redes-convolucionales/.

[60] Basic of source separation. https://source-separation.github.io/tutorial/basics/evaluation.html.

[61] Joachim Ganseman, Paul Scheunders, Gautham Mysore, and Jonathan Abel. Evaluation of a score-informed source separation system. pages 219–224, 01 2010.

[62] Tom Mitchell, Bruce Buchanan, Gerald DeJong, Thomas Dietterich, Paul Rosenbloom, and Alex Waibel. Machine learning. *Annual review of computer science*, 4(1):417–433, 1990.

[63] Onsets and frames: Dual-objective piano transcription. https://magenta.tensorflow.org/onsets-frames.

Lista de Figuras

Lista de Tablas