



Escuela
Politécnica
Superior

Transcripción musical mediante redes neuronales profundas



Grado en Ingeniería Informática

Trabajo Fin de Grado

Autor:

Jorge Donis del Álamo

Tutor/es:

José Javier Valero Mas

Jorge Calvo Zaragoza

Julio 2021

Transcripción musical mediante redes neuronales profundas

Autor

Jorge Donis del Álamo

Tutor/es

José Javier Valero Mas

Departamento de Lenguajes y Sistemas Informáticos

Jorge Calvo Zaragoza

Departamento de Lenguajes y Sistemas Informáticos



Grado en Ingeniería Informática



Escuela
Politécnica
Superior



Universitat d'Alacant
Universidad de Alicante

ALICANTE, Julio 2021

Preámbulo

Los recientes avances en el campo de la inteligencia artificial están sacando a la luz posibles y muy interesantes aplicaciones de nuestra creciente potencia computacional. El análisis musical es considerado como una tarea de gran esfuerzo cognitivo y terriblemente compleja. En contadas ocasiones relegamos esta tarea a seres humanos, ¿Podrán las máquinas ser de ayuda?

Aún con todo, las redes neuronales profundas, las redes recurrentes o de convolución son herramientas muy potentes que nos pueden ayudar a atacar este problema. Gracias al aprendizaje automático podemos adentrarnos en este mundo del análisis musical desde una perspectiva más técnica, pero igualmente útil. En concreto, me centraré en el campo del **Music Information Retrieval (MIR)**. Este campo de investiga el **procesamiento de señales de audio**. Más en detalle, yo me centraré en la **transcripción automática, Automatic Music Transcription (AMT)**. Ésta consiste en traducir una señal de audio a una representación **simbólica** musical.

Las aplicaciones pueden llegar a ser realmente interesantes, por ejemplo, corregir en tiempo real a los intérpretes que se salen de correcta lectura de una partitura. También puede servir para almacenar información musical de una manera más compacta. A lo largo de este Trabajo Final de Grado (TFG) trataremos todas estas aplicaciones con mayor detalle.

Agradecimientos

En primer lugar, tengo que reconocer la gran labor que han realizado como docentes a los Srs. José Javier Valero Mas y Jorge Calvo Zaragoza. En especial, me gustaría enfatizar la prontitud de sus respuestas y el constante apoyo que me han ofrecido. Desde la revisión de este mismo texto hasta las más concretas minucias de la implementación han estado supervisadas por estas dos personas. Es desde luego que sin estas dos personas no podría haber realizado este trabajo.

En segundo lugar, quiero agradecer a todos los profesores que me han enseñado los conceptos básicos y fundamentales con los que he llevado a cabo esta pequeña labor de investigación. Los Srs. Juan Ramón Rico y Francisco Gallego han sido dos maravillosos educadores que me han instruido en las concresciones de la inteligencia artificial, pero también me han enseñado a ser un programador elegante y profesional.

Finalmente, quiero expresar mi gratitud para con mis padres y mi familia, constante fuente de afecto e impulso.

*Alle Menschen werden Brüder
Wo dein sanfter Flügel weilt.*

//

*Todos los hombres se hacen hermanos
allá donde tu suave ala se posa.*

Friedrich Schiller.

Índice general

1	Introducción	1
1.1	¿Qué es el MIR?	1
1.2	Music Classification	2
1.3	Recommender Systems	2
1.4	Music Source Separation	5
1.5	Music generation	6
1.6	Automatic Music Transcription	8
2	Marco teórico	11
2.1	Señales de audio	11
2.2	Análisis de Fourier	14
2.3	Short-Time Fourier Transform	16
2.4	Automatic Music Transcription	21
3	Objetivos	25
4	Metodología	27
4.1	Red Neuronal	27
4.2	Connectionist Temporal Classification	31
4.3	Mini-Batch Gradient Descent	34
4.4	Código	36
5	Resultados	37
5.1	<i>Dataset</i>	37
5.2	Métrica de error	37
5.3	Modelos	38
5.3.1	Modelo inicial	38
5.3.2	Cambio de espectrograma	42
5.3.3	Ajuste de compases	43
6	Conclusiones	47
	Bibliografía	49
	Lista de Acrónimos y Abreviaturas	53

Índice de figuras

1.1	Imagen asociada a <i>Poker Face</i>	2
1.2	Características para una canción de Jazz (arriba) y Hip-Hop (abajo).	4
1.3	Descomposición de un espectrograma en sus correspondientes tracks (Cano y cols., 2019).	6
1.4	Descomposición de $S = I \times A$ Non-negative Matrix Factorization (NMF) . .	7
1.5	Extracto de la fuga en Do \sharp menor n°1 del <i>Das wohltemperierte Klavier I</i> , de J.S Bach.	7
1.6	Ejemplo de improvisación basada en reglas (Alexandre Donze, s.f.). Se proporciona una melodía y harmonía original (a). Después podemos ver una improvisación con reglas (b) y otra sin reglas (c).	8
1.7	La entrada (a) al problema es la señal de audio. (b) es la representación tiempo frecuencia de la señal (espectrograma). Tras el proceso de AMT se obtiene (c), que en este caso es una representación de <i>piano roll</i> . Finalmente se obtiene (d), la notación simbólica.	9
2.1	Vibración de una cuerda del violín cuando es percutida (<i>pizzicato</i>). Se puede apreciar la repetición periódica sinusoidal.	11
2.2	Representación de una señal periódica compleja . A la izquierda, en el dominio de amplitud; a la derecha, en el dominio de frecuencia. (Kiper, 2016)	12
2.3	Representación de una señal periódica (sinusoidal) simple	12
2.4	Ejemplo de ruido (fonema /s/). Podemos observar una distribución homogénea o densa de las frecuencias. Además, no se pueden detectar harmónicos de manera clara.	13
2.5	Sonido de percusión (palmada). Vemos una clara <i>inharmonicity</i> , de manera similar al ruido.	14
2.6	Descomposición frecuencial mediante la transformada de Fourier. $e(t) = a(t) + b(t) + c(t) + d(t)$	15
2.7	Aplicación de la función de ventana de Hann: a) muestra la onda original y b) después de aplicarle la función.	17
2.8	Comparativa de espectros frecuenciales calculados con Discrete Fourier Transform (DFT). En a) no se aplica ningún tipo de función de ventana a la onda compleja. En b) se ha aplicado una ventana de Hann antes realizar la DFT.	18
2.9	Resumen de la Short-Time Fourier Transform (STFT).	19
2.10	M es el tamaño de la ventana y R el <i>hop-length</i> . En este caso M es igual al número de muestras por <i>frame</i> n . Normalmente, en Short-Time Fourier Transform (STFT) suele ser así.	20
2.11	Espectrograma tiempo-frecuencia. En este caso se usa una escala logarítmica en el eje y. Esto es útil para resaltar las frecuencias fundamentales (que son las más graves).	21

4.1	Arquitectura de una red CRNN (Convolutional Recurrent Neural Network).	28
4.2	Ejemplo de espectrograma sintético. Se puede apreciar cómo apenas existen frecuencias fuera de los parciales. Es decir, hay poca <i>inharmonicity</i> (al tratarse de un instrumento sintético).	29
4.3	Ejemplo de aplicación de un <i>kernel</i> de convolución.	29
4.4	Ejemplo de red recurrente. Podemos ver que para cada instante, se proporciona una salida y_t (que depende de las anteriores por a_{t-1} , pero también otra salida a_t que se conecta al siguiente <i>frame</i> . De esta manera, se introduce el concepto de temporalidad y de memoria.	30
4.5	Notación simbólica <i>semantic</i>	31
4.6	Ejemplo de decodificación en Connectionist Temporal Classification (CTC). Podemos apreciar cómo varios <i>frames</i> se corresponden con la misma salida y el uso del carácter en blanco (-) para cambiar de la salida a a la b	33
4.7	Ejemplo de predicción en CTC.	33
5.1	Evolución del error dentro y fuera de la muestra durante el proceso de aprendizaje.	39
5.2	Comparativa entre los dos tipos de espectrogramas empleados. Ambos espectrogramas provienen de la misma muestra de audio. Ambas imágenes han sido reescaladas. Cabe destacar que el modelo trabaja con imágenes en escala de grises.	43
5.3	Evolución del error dentro y fuera de la muestra durante el proceso de aprendizaje. Se emplea el espectrograma con escala de Mel.	44

Índice de tablas

4.1	Arquitectura de la Convolutional Recurrent Neural Network (CRNN)	32
5.1	Ejemplo de predicción para el modelo inicial. Nótese que la predicción nº 10 contiene un vacío. Realmente la dimensionalidad de la tupla de predicción y el <i>ground-truth</i> es distinta. El vacío ha sido desplazado deliberadamente <i>a posteriori</i> para obtener un alineamiento más lógico.	40
5.2	Ejemplo de predicción. Al igual que en la Tabla 5.1, se mueve la predicción vacía a la posición 10 deliberadamente.	41
5.3	Transcripciones con y sin corrección <i>a posteriori</i> de las barras de compás, respectivamente.	45

Índice de Códigos

4.1	Clase SemanticTranslator	30
4.2	Función de decodificación para CTC.	34
4.3	Función de generación de <i>batches</i> de aprendizaje.	35
4.4	Función de comprobación de tamaño mínimo del espectrograma	35
5.1	Cálculo de la distancia de edición normalizada	38
5.2	Algoritmo de relleno de barras de compás.	44

1 Introducción

A lo largo de este Capítulo explicaremos qué es el MIR y cuáles son algunas de sus ramas y sus posibles aplicaciones. No entraremos en demasiado detalle sobre ninguna de ellas, a excepción del AMT, que es el foco principal de este TFG.

1.1 ¿Qué es el MIR?

Music Information Retrieval (MIR) es la ciencia interdisciplinar que estudia la extracción y el procesamiento de información a partir de la música. Por concretar, en muy contadas ocasiones se trata de procesar información *contextual* (Orio, 2006). Es decir, MIR está fuertemente ligado al **procesamiento de señales**. En general, podemos decir que la música, a un nivel objetivo (no artístico) está formada de:

1. La señal de sonido.
2. La partitura.
3. Información contextual (metadatos).

Tanto la señal como la partitura pueden ser usadas dentro de MIR, pero es muy complicado usar metadatos. Por ejemplo, supongamos la 9^a sinfonía de Beethoven. Algunos metadatos serían

```
1 "autor" : {  
2     "nombre" : "Beethoven, Ludwig Van",  
3     "fechaNacimiento" : "17/12/1770",  
4     "ciudadNacimiento" : "Bonn"  
5 },  
6 "genero" : "sinfonia",  
7 "numMovimientos" : 4
```

Sería muy complicado usar cualquiera de estos datos en ninguno de los campos del MIR. Por ejemplo, si queremos construir un sistema de recomendación musical, de poco nos vale saber que una canción es una *balada lenta de cantante femenina grabada en 1980*. Esto es, simplemente, porque no reduciría mucho el ámbito de la búsqueda.

Con esto, nos tenemos que centrar, fundamentalmente, en el **procesamiento del audio**. De aquí se puede extraer una cantidad enorme de información. A continuación explicaré algunos de los fundamentales ámbitos de estudio del MIR y sus aplicaciones.

1.2 Music Classification

En general, este campo estudia el “etiquetado” o clasificación de una música dentro de una categoría¹. Estas categorías pueden ser muy variadas. Por ejemplo: *jazz*, *clásica*, *R&B*, *reggae*... pero también: *drums*, *violin*, *electronic*... Se puede clasificar por instrumentos, ritmos, géneros, número de cantantes, género de los cantantes, tempo...

Pero este campo realmente es mucho más subjetivo de lo que parece. Existen sistemas como *Drinkify*² que te ofrecen un cóctel personalizado para acompañar la música que estás escuchando. O por ejemplo, la plataforma de reproducción *Pandora* muestra imágenes de fondo acordes a la música de tu lista de reproducción.

Las técnicas de clasificación son diversas. Algunas de ellas son la búsqueda de los k-vecinos más cercanos, Support Vector Machine (SVM) o redes neuronales (Hagblade y cols., 2011). En este artículo también se explica la asociación de imágenes con canciones. Esto se hace usando el género como categoría común intermedia. Se obtienen resultados interesantes: por ejemplo, la imagen de la Figura 1.1 es asociada con la canción *Poker Face*, de Lady Gaga (comparten el género *Pop*). Esta es una asociación bastante correcta.



Figura 1.1: Imagen asociada a *Poker Face*

Las aplicaciones del etiquetado de canciones son evidentes. Desde un punto de vista comercial, analizar las relaciones entre grupos poblaciones y sus géneros musicales favoritos es claramente explotable.

También este campo de la clasificación está relacionado con el de la recomendación. En una plataforma de distribución de música, se puede recomendar al usuario géneros similares o canciones con el mismo género.

1.3 Recommender Systems

Los sistemas de recomendación de canciones han experimentado un claro crecimiento en los últimos años. Esto es debido a la aparición de plataformas de streaming como *Spotify*, *Pandora* o *Apple Music*, que proveen a los usuarios con una cantidad incommensurable de canciones. De entre tantos productos, es difícil ofrecer el correcto al cliente.

En definitiva, se trata de **extraer características** de las señales de audio y luego emplear **métricas de distancia** entre los distintos vectores de características. Si la distancia es baja,

¹El número de categorías no tiene porqué ser establecido a priori.

²<http://www.drinkify.org>

entonces las canciones serán similares. El sistema trata de recomendar al usuario canciones nuevas similares a las canciones que ya ha escuchado.

Pandora, en su proyecto *Music Genome Project*³ utiliza vectores de hasta 450 descriptores, entre ellos: *aggressive female vocalist*, *prominent backup vocals*, *abstract lyrics* o *use of unusual harmonies* (Schedl y cols., 2018).

Para explicar más en detalle el proceso, tomaré como ejemplo el artículo de Seyerlehner y cols. (2010. ISMIR 2010).

Primero se realiza un preprocessado de la señal de audio que consiste un *downsampling* a 22kHz y la posterior transformación a tiempo-frecuencia mediante la STFT⁴. Después se proceden a calcular **distintas características**.

Por una parte se calcula el *Patrón Espectral* (SP). Éste codifica el contenido *tímbrico* de la canción⁵. El SP se calcula ordenando (por bloques) la intensidad de las frecuencias. A partir del SP se calcula el DSP (Delta Spectral Pattern), que simplemente mide la variabilidad de los timbres. De manera análoga, se calcula, a partir del DSP el VDSP (Variance Delta Spectral Pattern).

Para codificar la estructura rítmica, se usa el Patrón Logarítmico de Fluctuación (LFP). Éste intenta medir la periodicidad de las frecuencias. Por ejemplo, en un compás simple de $\frac{4}{4}$, con un tempo *andante* (80 negras por minuto), cada 1,33 segundos se podrá observar un aumento de intensidad en todas las frecuencias (en concreto, las frecuencias de las notas de los instrumentos como el bajo).

Después se calcula el Patrón de Correlación (CP). Éste es una matriz cuadrada en la que se mide la correlación entre cada una de las distintas frecuencias. Por ejemplo, si la batería suele tocar el *hi-hat* junto al bombo, se verá una relación entre esas frecuencias (lo cual es una característica representativa de esa canción).

Finalmente se calcula el Patrón de Contraste Espectral (SCP). Simplemente mide (por bloques) la distancia entre la intensidad de la frecuencia más intensa y la intensidad de la frecuencia menos intensa. En general, esto es una métrica de la *harmonicity*⁶.

En la Figura 1.2 podemos ver una representación de todas estas características:

Posteriormente se concatenan todas las características en un vector de 9448 dimensiones. Este vector es el que *caracteriza a la canción*. Dos canciones similares tendrán vectores similares.

El vector se puede usar directamente para la **clasificación en géneros**. En este artículo, se emplea una red neuronal para ello. La entrada es el propio vector y las salidas son las distintas categorías.

También se pueden comparar vectores para ver su similitud. Para ello se usa la **distanzia de Manhattan**⁷.

$$d_1(p, q) = \sum_{i=1}^n |p_i - q_i|$$

³<https://www.pandora.com/about/mgp>

⁴Estos conceptos serán explicados en detalle en el Capítulo 2.

⁵Tal y como se explicará más adelante, el timbre de un instrumento depende de la intensidad de las distintas frecuencias que componen el sonido.

⁶Este concepto será explicado en el Capítulo 2.

⁷También conocida como distanciia L_1

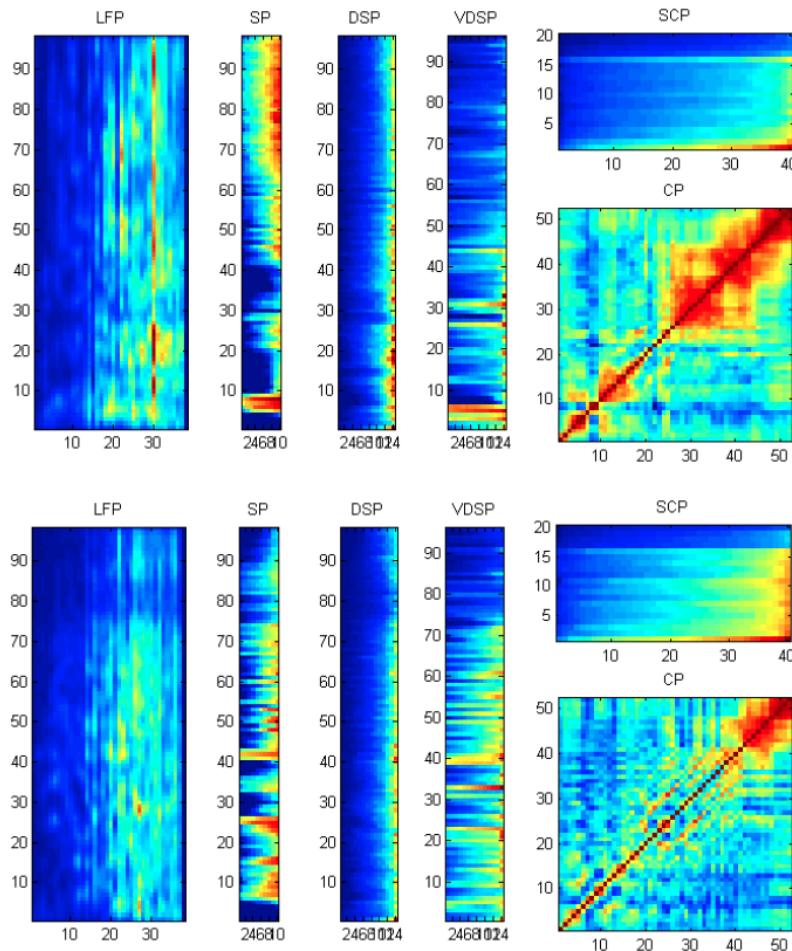


Figura 1.2: Características para una canción de Jazz (arriba) y Hip-Hop (abajo).

donde $p = [p_1, p_2, \dots, p_n]$ y $q = [q_1, q_2, \dots, q_n]$ son dos vectores con la misma dimensión n .

En este caso $n = 9448$. El problema es que cada dimensión contiene valores en rangos muy dispares. Por ello, hay que **normalizar** los datos.

Finalmente, también existe una segunda aproximación a la recomendación. Se pueden establecer un número determinado de categorías o etiquetas (no necesariamente tienen que ser géneros musicales) y clasificar el vector de características de la canción. A la salida de la red neuronal, tendremos un vector con el grado de confianza en la clasificación de cada una de las distintas categorías. Podemos comparar estos grados de confianza entre distintas canciones para medir su similitud.

Por último me gustaría destacar que existe una gran cantidad de factores que influencian el “deseo” de escuchar una canción. Éstos no están limitados al contenido *intrínseco* de la misma (el audio). Algunos factores contextuales son: (Adomavicius y cols., 2011).

- La personalidad y el estado emocional (Ferwerda y cols., 2015), (Rentfrow y Gosling, 2003).

- La actividad que se está realizando mientras se escucha música (Gillhofer y Schedl, 2015), (Wang y cols., 2012).
- El tiempo, la situación social o los lugares de interés del usuario (Adomavicius y cols., 2011), (Kaminskas y cols., 2013).
- Las canciones que se han escuchado inmediatamente antes⁸(Mcfee y Lanckriet, 2012),(Zheleva y cols., 2010).

1.4 Music Source Separation

Music Source Separation (MSS) es el campo de la MIR que consiste en la descomposición de una canción en sus correspondientes *tracks*⁹ o incluso instrumentos. La Figura 1.3 muestra el resultado al cual se trata de llegar con Music Source Separation (MSS).

El primer paso es, como suele darse en MIR, representar de la señal en el plano tiempo-frecuencia (espectrograma). Después, existen varios métodos para hacer la separación.

Uno de ellos es la **Non-negative Matrix Factorization (NMF)**. La entrada al problema es el espectrograma, que en definitiva es una matriz de intensidades de frecuencias $S_{t,n}$ donde t es el número de *timesteps*¹⁰ y n es el número de distintas frecuencias. El problema que se intenta resolver es obtener dos matrices $D_{n,k}$ (diccionario) y $A_{k,t}$ (matriz de activación) tales que $S = DA^{11}$. k vendría a ser el tamaño del diccionario. En general, se busca una $k < n$. Cada columna de I es un vector de k *plantillas espectrales* (cada plantilla, de longitud f). Tras la factorización, se puede apreciar en A los instantes t durante los cuales se activa cada una de las plantillas espectrales. En la Figura 1.4 se observa la factorización claramente. Si escogemos una k igual al número de notas (o notas MIDI), entonces estaremos haciendo AMT (ver Sección 1.6).

Con esto, ya tendríamos los intervalos temporales durante los cuales ciertas frecuencias se activan con una cierta intensidad. Si $k = 5$, tendríamos la separación para los *tracks*: *vocals*, *bass*, *drums*, *guitar* y *others*. Finalmente, queda extraer las frecuencias necesarias del espectrograma original. Una vez tenemos las frecuencias para cada *track* aisladas, queda realizar la **transformación inversa frecuencia-tiempo** para volver a obtener una señal de sonido.

Cabe decir que este proceso puede brindar unos mejores resultados si la canción ya está separada en formato **estéreo**, ya que la percusión y el bajo suelen encontrarse en el canal izquierdo y el resto de instrumentos en el derecho.

Las aplicaciones del MSS son evidentes. Cuando no se tiene acceso a los *tracks* originales¹², se pueden obtener de manera automática. Éstos pueden ser usados simplemente para cantar karaoke, por entretenimiento. También puede servir en ámbitos educativos para enseñar

⁸Es interesante cómo las preferencias del oyente varían según la *sesión* de escucha musical. Es decir, hay canciones que van bien “en conjunto”.

⁹*vocals*, *bass*, *drums*, *guitar* y *others* suelen ser las categorías más empleadas.

¹⁰*timestep* es un término muy empleado en MIR. Viene a representar la menor cantidad de tiempo discretizable posible. En definitiva, son ventanas de tiempo. Por ejemplo, en **librosa**, al calcular la STFT sobre una señal de 22kHz, se obtienen *timesteps* de 93 ms. También se conoce como *time frames*.

¹¹Suponiendo que las tres matrices están formadas por enteros no negativos.

¹²Master record.

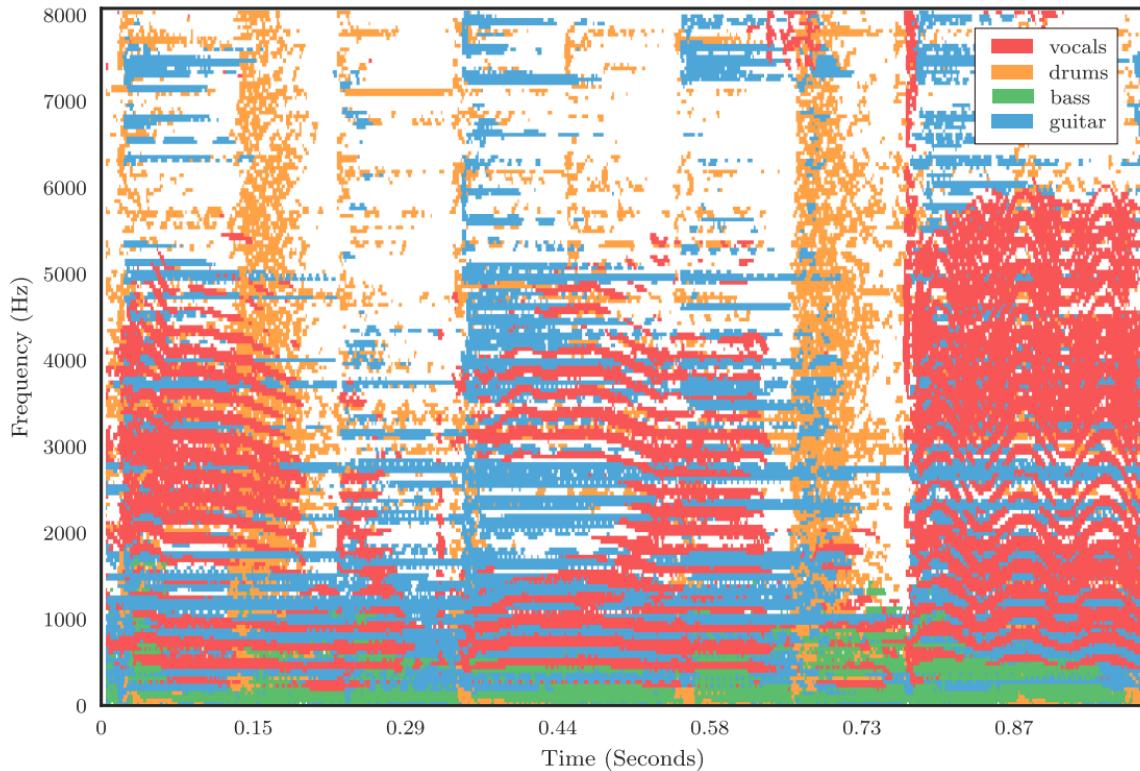


Figura 1.3: Descomposición de un espectrograma en sus correspondientes tracks (Cano y cols., 2019).

canto. Otro ejemplo de aplicación sería aligerar la carga de un sistema de recomendación musical.

1.5 Music generation

Este es uno de los campos más interesantes del MIR. Consiste en componer música de manera automática, con poca o ninguna ayuda por parte de los humanos. Aquí entra en juego una discusión artística sobre **qué es componer** o la originalidad de las composiciones. ¿Sería el autor de la composición la máquina o el programador? ¿O sería el usuario del programa?

Aún con todo, se puede ver a lo largo de la historia que la composición musical ya contiene algo de *algoritmia* de manera inherente. Por ejemplo, durante el período de la práctica común, existían una serie de progresiones harmónicas que se solían seguir de manera muy rígida. Por ejemplo, que una sonata solía tener cuatro movimientos y que el segundo de estos solía ser en la tonalidad relativa. Por no hablar del contrapunto occidental (ver Figura 1.5).

Existen, básicamente, dos maneras de abordar este problema (Alexandre Donze, s.f.): mediante reglas y mediante aprendizaje automático (datos).

Las aproximaciones basadas en reglas intentan definir qué es “bueno”, y a partir de ahí, improvisan (ver Figura 1.6). El problema es que escoger estas reglas resulta muy complicado. Si las reglas son demasiado rígidas, la improvisación no es lo suficientemente creativa. Sin

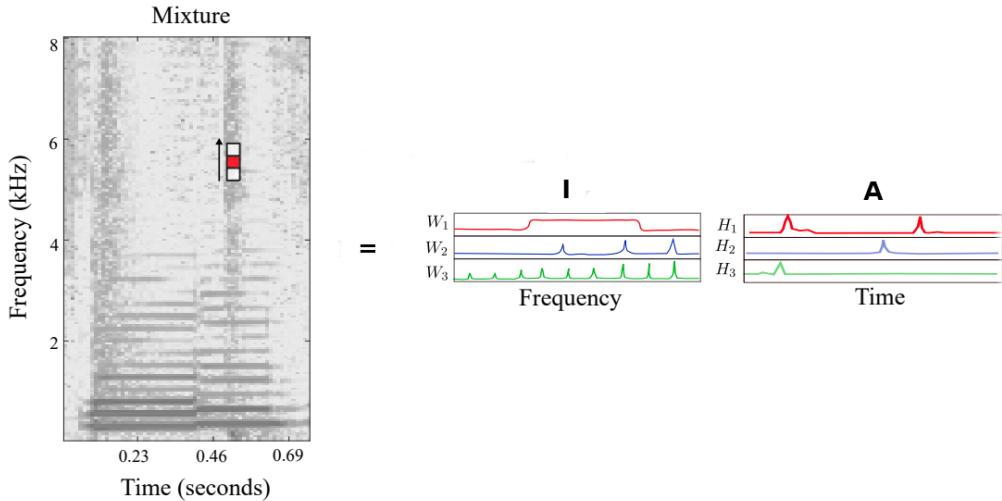


Figura 1.4: Descomposición de $S = I \times A$ NMF

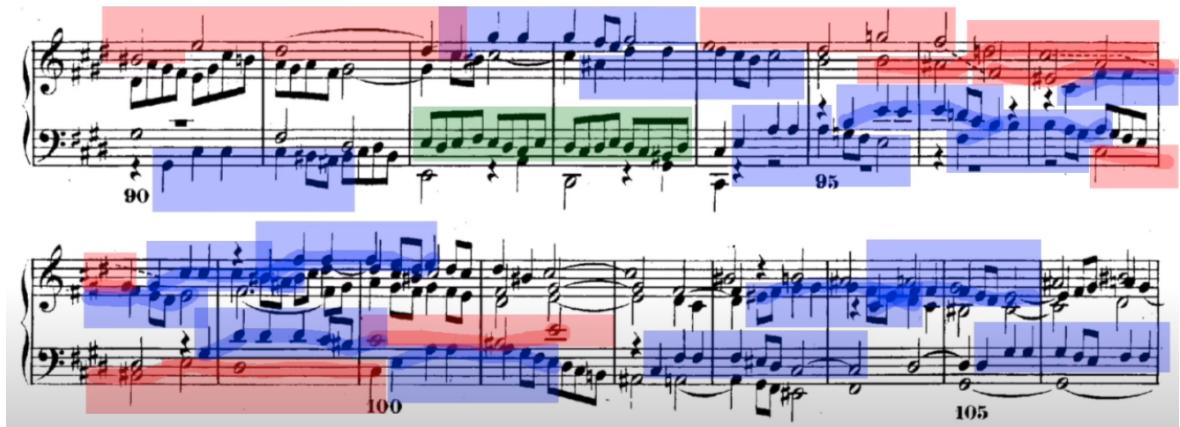


Figura 1.5: En este extracto de la fuga en Do \sharp menor nº1 del *Das wohltemperierte Klavier I*, de J.S Bach podemos observar claramente el **contrapunto**. El primer tema (rojo) se repite en las cuatro voces, al igual que el azul. Existe cierta *algoritmia* o metodismo en el contrapunto. Los temas se han de repetir, transportados a cierta distancia, pero siguiendo el mismo ritmo (o con pequeñas variaciones). Además, la conjunción de estos temas ha de seguir una progresión harmónica. Parece casi un problema matemático.

The figure consists of three musical staves, labeled (a), (b), and (c). Each staff has a treble clef and a key signature of one sharp (F#). The first staff (a) contains a melody and harmonic progression: C (measures 1-2), Am (measure 3), Dm (measure 4), G7 (measure 5), C (measure 6), F (measure 7), and Fm (measure 8). The second staff (b) shows an improvisation that follows the harmonic progression, with some notes highlighted by blue dots. The third staff (c) shows an improvisation that also follows the harmonic progression, but with many notes marked by red asterisks.

Figura 1.6: Ejemplo de improvisación basada en reglas (Alexandre Donze, s.f.). Se proporciona una melodía y harmonía original (a). Después podemos ver una improvisación con reglas (b) y otra sin reglas (c).

embargo, si las reglas son muy laxas, la composición final no es “bonita” (contiene demasiado ruido o notas “erróneas”).

La aproximación basada en aprendizaje automático utiliza la notación simbólica de canciones como entrenamiento. Existen modelos basados en redes neuronales, aunque también son muy utilizados los **Modelos Ocultos de Markov** (HMM).

1.6 Automatic Music Transcription

La transcripción musical automática o Automatic Music Transcription (AMT), consiste en la traducción de señales de audio a una **representación musical simbólica**. Esta es la principal rama del MIR que trataré en mi TFG. A lo largo del Capítulo 2 explicaré claramente el proceso. La Figura 1.7 sirve como síntesis del problema a resolver.

Las aplicaciones del AMT son múltiples:

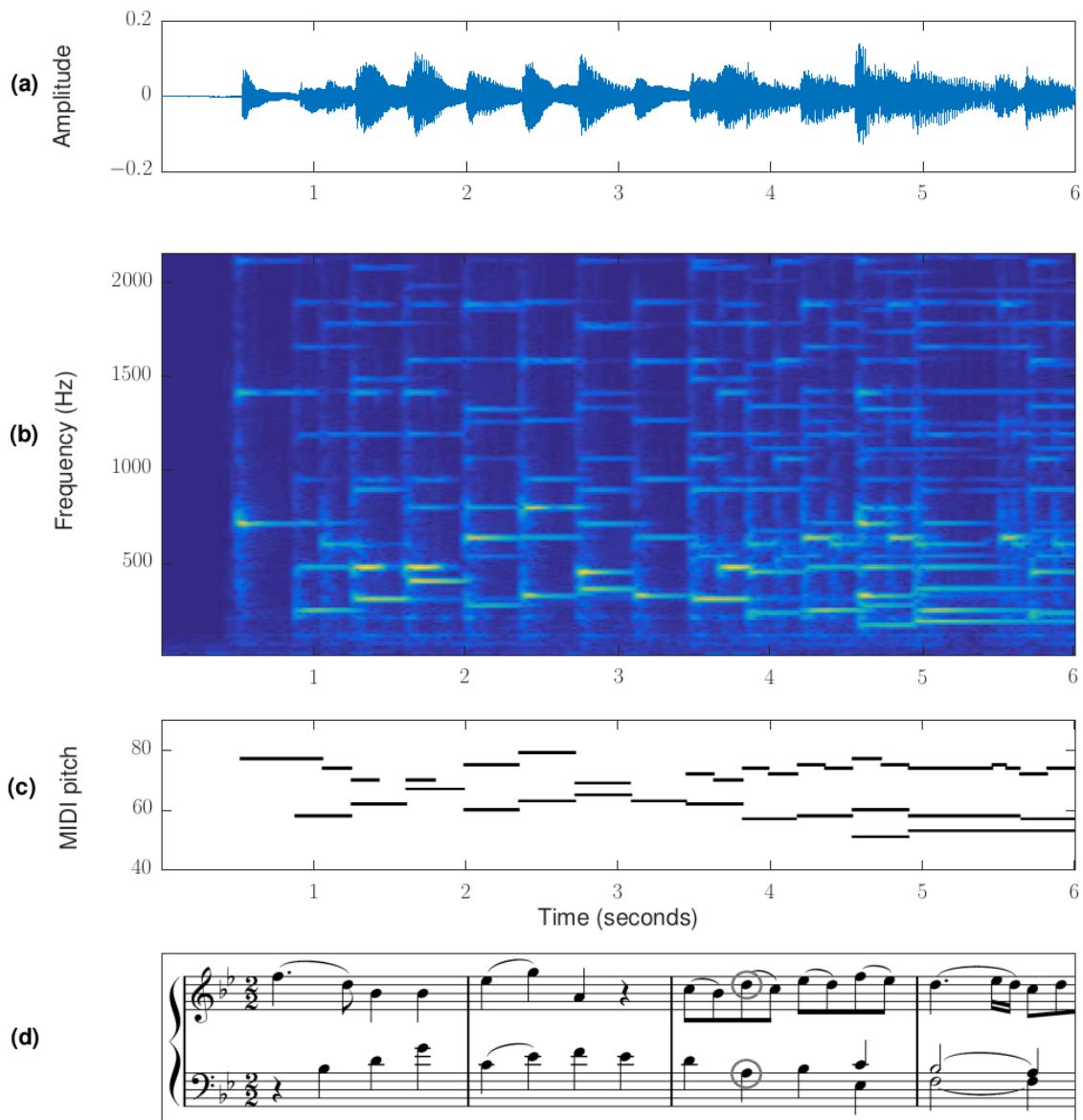


Figura 1.7: La entrada (a) al problema es la señal de audio. (b) es la representación tiempo frecuencia de la señal (espectrograma). Tras el proceso de AMT se obtiene (c), que en este caso es una representación de *piano roll*. Finalmente se obtiene (d), la notación simbólica.

- **Almacenamiento:** desde un punto de vista totalmente pragmático, puede resultar ventajoso almacenar la representación simbólica de las músicas. En definitiva, la partitura no deja de ser una “codificación” **comprimida** del sonido que se desea representar. Los libros se guardan escritos con las letras del abecedario y no con grabaciones de sus lecturas. Lo mismo se puede dar para la música.
- **Sistemas de recomendación:** las representaciones simbólicas nos brindan otro punto de vista dentro del análisis comparativo de músicas. Esto puede ser provechoso a la hora de analizar la similitud entre dos músicas. Si éstas comparten figuras rítmicas (por ejemplo, ritmos de negra punteada), tonalidad, progresiones harmónicas... entonces serán ellas mismas similares. Está claro que todas estas características se pueden obtener sin pasar por una representación simbólica intermedia; pero también es cierto que es mucho más sencillo extraer dichas características a partir de la representación simbólica en vez de directamente de la señal de audio.
- **Búsqueda:** la notación simbólica nos permite **agilizar** los tiempos de búsqueda. Por ejemplo, si observamos que cierta melodía nos resulta interesante, podemos buscar dentro de un corpus con miles de canciones para encontrar músicas que contienen esa misma melodía, o esa melodía transpuesta, o con un ritmo similar...
- **Musicología de las improvisaciones:** la principal utilidad del AMT es la propia transcripción. El proceso artístico de creación musical no tiene que pasar necesariamente por la composición. El *jazz* es un claro ejemplo de música de carácter improvisado¹³. Otro ejemplo es el **cante**. El proyecto **COFLA**¹⁴ de la Universidad de Sevilla estudia precisamente la transcripción automática de cantaores.

¹³Evidentemente, existen composiciones jazzísticas.

¹⁴<http://www.cofla-project.com>

2 Marco teórico

A lo largo de este Capítulo explicaremos en detalle todos los fundamentos teóricos necesarios para la comprensión del AMT. Atacaremos el problema desde la entrada (señales de audio) hasta la salida (representación musical simbólica). Una vez explicados estos conceptos, en el Capítulo 4 se abarcará mi aproximación a la resolución a este problema de una manera más concreta.

2.1 Señales de audio

Uno de los objetivos fundamentales del AMT es la detección de la **altura**¹ de los sonidos. Para comprender el concepto de altura es necesario estudiar la naturaleza del sonido.

El audio es la entrada al problema de la transcripción musical. Como ejemplo, tomaremos la Figura 1.7. De hecho, esta figura es una buena referencia para entender el problema como conjunto y faremos referencia a la misma con bastante frecuencia a lo largo del Capítulo.

Una **onda sonora** es una perturbación en la presión del aire que resulta de una vibración. Cuando la cuerda de un violín es frotada (ver Figura 2.1), hace que vibre con una cierta frecuencia. Esta vibración desplaza el aire, provocando variaciones en la presión del mismo. Estas variaciones de presión llegan al oído humano, donde son convertidas en ondas mecánicas que el cerebro interpreta como sonidos.

Esta perturbación del sonido a lo largo del tiempo se puede representar tal y como indica la Figura 2.2 (gráfica izquierda). Casi todas las representaciones de la señal indican variabilidad de alguna magnitud respecto del **tiempo**. En el caso de la **amplitud**, ésta puede hacer referencia a otras varias magnitudes. Por ejemplo, durante la propagación física de la onda a través del sonido, la magnitud será la presión (μPa en el caso de la Figura 2.2). Dentro del procesamiento de señales digitales, la amplitud representa la **diferencia de potencial**. Esta diferencia de potencial realmente no se suele representar *per se*, sino que se hace uso

¹Los sonidos se pueden clasificar, según su altura, en agudos o graves.



Figura 2.1: Vibración de una cuerda del violín cuando es percutida (*pizzicato*). Se puede apreciar la repetición periódica sinusoidal.

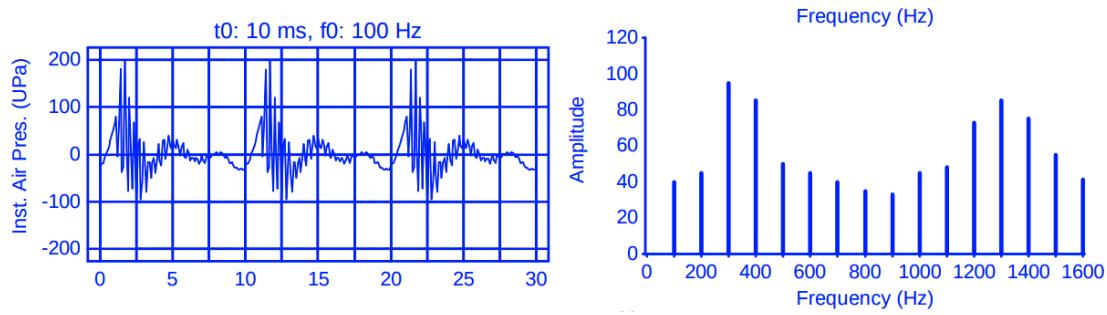


Figura 2.2: Representación de una señal periódica **compleja**. A la izquierda, en el dominio de amplitud; a la derecha, en el dominio de frecuencia. (Kiper, 2016)

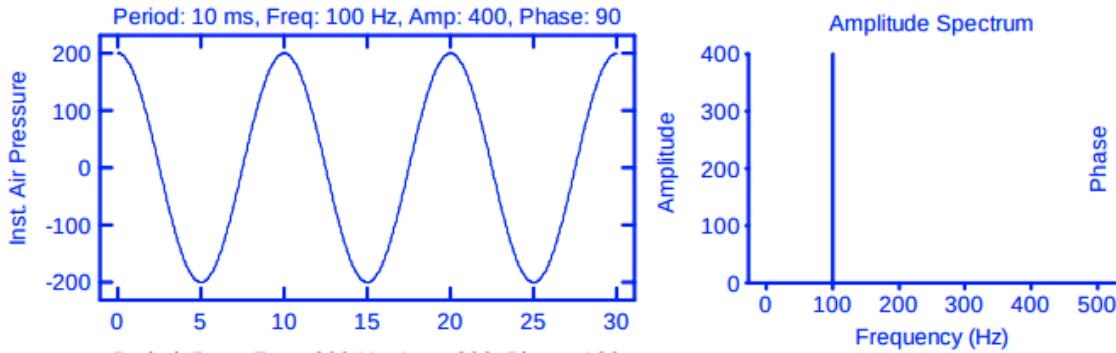


Figura 2.3: Representación de una señal periódica (sinusoidal) **simple** con $f = 100\text{Hz}$.³

de un valor de referencia a_0 ² y se emplean razones decibélicas. Independientemente del significado físico de la amplitud, lo que es realmente relevante es que **la amplitud representa intensidad**.

Supongamos una señal como la de la Figura 2.3. Ésta es una **señal periódica simple**. Podemos observar una característica y es que los instantes en los que la intensidad de la amplitud es máxima son *periódicos*. Suceden cada 10ms. Normalmente se utiliza la magnitud de **frecuencia** para medir esta periodicidad. La frecuencia es la inversa del periodo.

$$f = \frac{1}{T}$$

Esta frecuencia es claramente percibida por el oído humano. **Las frecuencias más altas son percibidas como más agudas**. En la gráfica de la derecha de la Figura 2.3 podemos ver el **espectro**⁴ de la señal. El espectro nos indica la intensidad de cada una de las frecuencias. En este caso, la frecuencia son 100Hz. Sin embargo, las notas de los instrumentos (o el habla humana) no está compuestas por una única frecuencia, sino por varias. Por ello, a este tipo de

²En el campo del audio digital, normalmente $a = 10^{-5}$. Se escoge este valor para que la amplitud máxima que puede ser emitida por el hardware sea 1.

³Esta frecuencia se percibiría como una nota entre Sol₂ y Sol₂[#].

⁴Normalmente, siempre nos referiremos al espectro frecuencial.

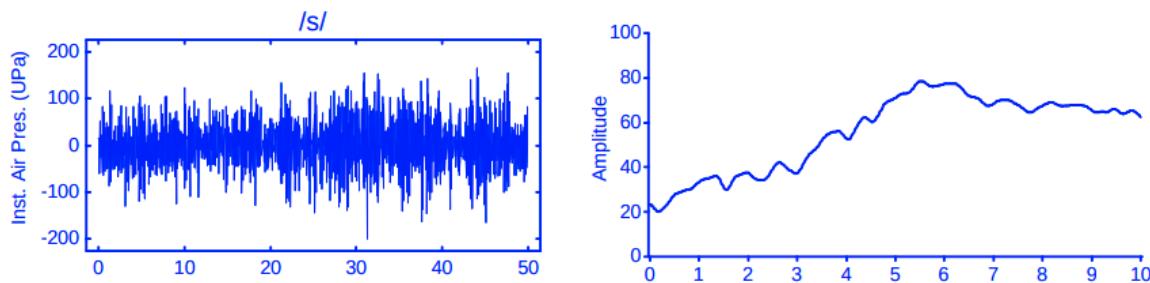


Figura 2.4: Ejemplo de ruido (fonema /s/). Podemos observar una distribución homogénea o **densa** de las frecuencias. Además, no se pueden detectar harmónicos de manera clara.

señales se las conoce como **periódicas complejas**. Un ejemplo de señal periódica compleja (como la que podría producir una cuerda frotada de un violín) es el de la Figura 2.2. A la derecha de esta figura se encuentra su espectro. Podemos observar una clara diferencia éste y el anterior espectro: hay más de una frecuencia. Sin embargo, no todas las frecuencias tienen la misma intensidad, sino que existe cierta **distribución**. Si analizamos esta distribución, llegaremos a una conclusión muy interesante y es que existe una **frecuencia fundamental** f_0 ($f_0 = 200\text{Hz}$, en este caso) y que podemos representar el resto de frecuencias predominantes como múltiplos de esta f_0 . Formalmente:

$$\begin{aligned} F_o &= \{f_i = k f_0 \mid f_i \in F_p, k \in \mathbb{Z}, k > 1\} \\ F_i &= F_p - F_o \end{aligned}$$

Donde F_p es el conjunto de todas las frecuencias de las señales simples⁵. Cada una de estas ondas es conocida **parcial**. F_o son todos aquellos parciales cuya frecuencia es un múltiplo entero positivo de f_0 . Este conjunto es conocido como los *overtones*. Cuando F_o también contiene la propia f_0 , hablamos de **harmónicos**, en general. Para el caso concreto de la Figura 2.2, $f_1 = 400\text{Hz}$, $f_2 = 600\text{Hz}$... Finalmente, F_i es el resto de frecuencias que no pertenecen a los *overtones*. Se conocen como **parciales inharmonicos**.

Cuando un violín toca un La4, decimos que tiene una frecuencia de 400Hz, sin embargo, esto es una simplificación. El violín ha producido un sonido compuesto por muchas frecuencias. Algunas de estas frecuencias son harmónicos, **pero otras no**. Todas estas frecuencias son las que dotan al sonido del violín de cierta **inharmonicity**. Realmente, $f_0 = 440\text{Hz}$, es decir, la única frecuencia a la que nos solemos referir es la frecuencia fundamental. Ésta es la más grave y la que se percibe con mayor intensidad. Sin embargo, el resto de *overtones* son los que caracterizan al sonido particular del violín⁶. Es decir, la distribución de intensidad entre los distintos *overtones* es lo que dicta el **timbre** o “color” de un sonido. En concreto, cada *overtone* afecta a un “color” distinto. Por ejemplo, f_3 , f_6 y f_{12} dotan al sonido de un carácter “nasal”.

Un ejemplo de *inharmonicity* extrema es el *ruido* (ver Figura 2.4). Sin embargo, la percusión también presenta un alto grado de *inharmonicity* (ver Figura 2.5).

⁵Cabe recordar que una onda simple tiene una única frecuencia (es sinusoidal).

⁶Como curiosidad, el violín tiene *overtones* que no pueden ser percibidos por el oído humano.

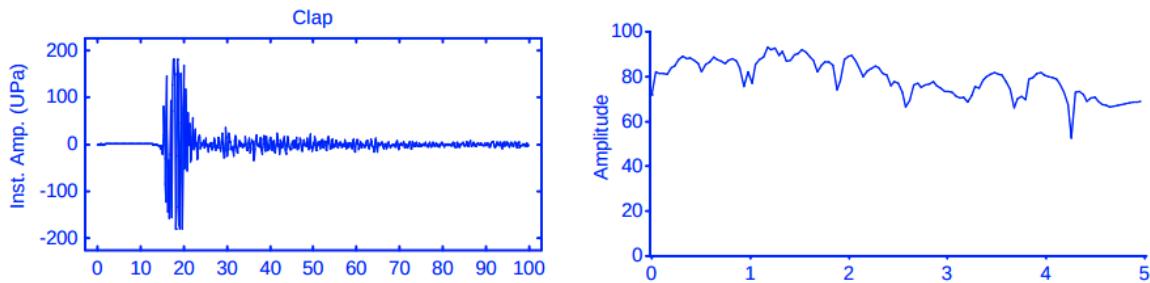


Figura 2.5: Sonido de percusión (palmada). Vemos una clara *inharmonicity*, de manera similar al ruido.

En resumen, **los sonidos se pueden entender como composiciones de varias frecuencias**. En la siguiente sección veremos cómo podemos extraer dichas frecuencias.

2.2 Análisis de Fourier

A lo largo de la anterior sección hemos observado parejas de gráficas que representaban el sonido. A la izquierda veíamos la amplitud de la señal y a la derecha el espectro. Pero, ¿Cómo se obtiene el espectro de una onda?

En el caso de ondas simples, es sencillo calcular la frecuencia. Sólo hay una y es la inversa del tiempo que transcurre entre los picos de mayor intensidad. Pero, ¿qué sucede en el caso de las ondas periódicas complejas?

El **Teorema de Fourier** demuestra que cualquier onda compleja puede ser representada como una combinación de funciones sinusoides simples. La Figura 2.6 nos indica este proceso.

Supongamos que la función $g(t)$ se corresponde con la gráfica (e) de la Figura 2.6. Esta función nos proporciona la intensidad total⁷ de la onda en el instante t . Podemos definir $\hat{g}(f)$ tal que:

$$\hat{g}(f) = \int_{-\infty}^{+\infty} g(t)e^{-2\pi ift} dt$$

donde f es la frecuencia de la cual nos interesa analizar su intensidad e i es la unidad imaginaria⁸. Cabe comprender que $\hat{g}(f)$ es una función que a su vez es construida por composición a partir de $g(t)$. Al integrar respecto al tiempo, nos sigue quedando la variable f , con lo que conseguimos pasar del espectro de amplitud de $g(t)$ al **espectro frecuencial** de $\hat{g}(f)$. Es decir, hemos cambiado de dominio temporal al dominio frecuencial. $\hat{g}(f)$ se conoce como la **Transformada de Fourier** (de g) o Fourier Transform (FT).

Cabe destacar que las ondas con las que se trabaja en el procesamiento de señales de audio digitales no son funciones continuas periódicas, sino **discretas**. Además, no podríamos integrar entre $(-\infty, +\infty)$, porque la señal no es infinitamente larga (y los tiempos negativos no tienen un significado físicamente válido). Por ello, en la práctica se emplea la **Discrete Fourier Transform (DFT)**. Sea un conjunto de valores complejos discreto $x = x_1, x_2 \dots x_{N-1}$ donde N es el número total de elementos. Podemos definir la DFT como:

⁷Recordemos que las ondas complejas están formadas por muchas sinusoides simples.

⁸Cabe destacar que la transformada de Fourier tiene un rango complejo.

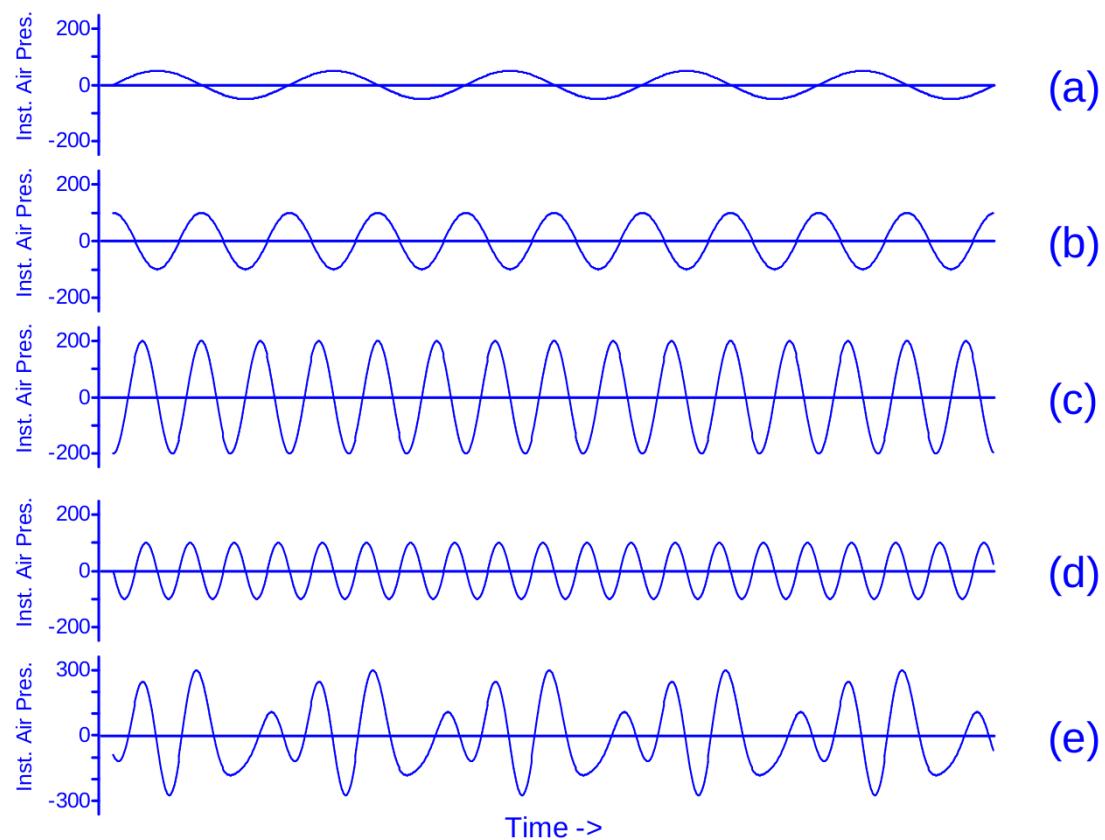


Figura 2.6: Descomposición frecuencial mediante la transformada de Fourier. $e(t) = a(t) + b(t) + c(t) + d(t)$.

$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} kn}$$

donde X_k es la intensidad de la frecuencia k . El resultado es también un número complejo. $|X_k|$ indica la intensidad y $\text{atan}(\frac{\text{Im}(X_k)}{\text{Re}(X_k)})$ indica la **fase**. El aparato auditivo humano no es tan sensible a las variaciones en fase como a las variaciones en intensidad, por ello se suele descartar. Respecto a la frecuencia (k), es comúnmente aceptado que el oído humano es capaz de reconocer frecuencias de entre 20Hz y 20kHz.

En definitiva, las ondas complejas están compuestas por sumas ondas simples. Podemos extraer las frecuencias de estas ondas simples mediante la DFT.

2.3 Short-Time Fourier Transform

En el apartado anterior explicábamos cómo extraer las frecuencias de una onda compleja. Sin embargo, a lo largo de un sonido musical, se suceden muchas distintas ondas a lo largo del **tiempo**. Es decir, que el espectro frecuencial que obteníamos con la DFT tenía dos dimensiones: frecuencia e intensidad. Sin embargo, ahora queremos añadir una tercera dimensión que es el tiempo.

La manera más evidente de hacer esto es establecer bloques de un tamaño fijo y calcular la DFT sobre cada una de ellos. Para no perder información a lo largo del dominio temporal, solaparemos los bloques entre sí cierto número de samples (muestras). A estos bloques se les aplicará⁹ una **función de ventana** para reducir ciertos efectos nocivos de este solape.

La función de ventana es la función por la que pasará la onda antes de ser procesada por la DFT. Suelen ser simétricas y valen 0 más allá de cierto intervalo preestablecido. Una de las ventanas más usadas es la de **Hann**:

$$w[n] = \sin^2\left(\frac{\pi n}{N}\right)$$

Podemos verla aplicada en la Figura 2.7. En general, las funciones de ventana sirven para eliminar el **leakage** (filtración o derrame) que ocurre de manera natural en la DFT. El *leakage* sucede cuando se aplica el análisis de Fourier a funciones discretas **que no se presentan como periódicas**. Por ejemplo, la nota La (440Hz) pasa por 0 cada 2,2ms (tiene un periodo de 2,2ms). Si tenemos una grabación de 224ms, la onda no se podrá entender como periódica, porque hay 4ms que están fuera del periodo. Esto sucede porque 224 no es un múltiplo entero positivo de 2,2. En general, rara vez el número de muestras será un múltiplo entero positivo del periodo. Al aplicar la DFT veremos que hay frecuencias con amplitudes similares a la de 440Hz que realmente no están presentes en la onda original sinusoidal pura. La función de ventana, al suavizar los extremos, nos ayuda a **reducir este leakage**. En la Figura 2.8 podemos apreciar cómo el uso de una venta de Hann hace que la frecuencia real tenga una magnitud más claramente predominante (se reduce el *leakage*).

La elección de la función de ventana no es una tarea trivial (“Understanding FFT’s and Windowing”, s.f.). Cada función tiene sus características. Ciertas funciones nos permitirán

⁹Como se verá más adelante, se trata de una multiplicación.

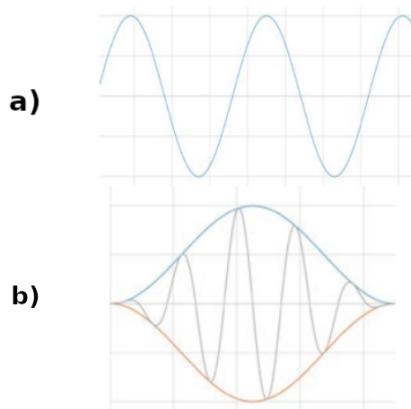


Figura 2.7: Aplicación de la función de ventana de Hann: a) muestra la onda original y b) después de aplicarle la función.

resolver distintas problemáticas. En cualquier caso, hay que analizar el contenido frecuencial de la onda.

- Si la señal contiene interferencias de frecuencias **muy distantes** a la frecuencia de interés, se usará una ventana en la que los extremos decrezcan **rápidamente**. La ventana de Hann es un ejemplo de este tipo de ventana.
- Si la señal contiene interferencias de frecuencias **próximas**, se usará una ventana en la que los extremos decrezcan **lentamente**. Por ejemplo, la ventana de **Hamming**.
- Si existen dos señales con frecuencias muy próximas, es interesante usar un función de ventana cuyo pico central se muy estrecho. Por ejemplo, una ventana **Gaussiana** $w[n] = \exp(-\frac{1}{2}(\frac{n-N/2}{\sigma N/2})^2)$.
- Si la amplitud de la frecuencia es muy pronunciada pero no es muy precisa (ocupa varias frecuencias), entonces se ha de escoger una función de ventana con un poco central amplio. Por ejemplo, la ventana Plank-taper.
- Si el espectro frecuencial es denso (o plano), es mejor usar la ventana uniforme. La ventana uniforme es equivalente a no usar ninguna ventana.

La ventana de Hann es una buena elección en el 95% de los casos (“Understanding FFT’s and Windowing”, s.f.). Tanto la ventana de Hann como la Hamming son **sinusoides** y esto las dota de un pico central ancho que captura muy bien la frecuencia original.

Con esto, el proceso de **Short-Time Fourier Transform (STFT)** quedaría resumido por la Figura 2.9. La entrada al problema es la señal compleja. La salida es el espectro tiempo-frecuencia. Los parámetros que podemos establecer son:

- **Función de ventana w[].** Explicada anteriormente, sirve para suavizar los extremos. Multiplicaremos la onda compleja en el intervalo deseado por este valor.

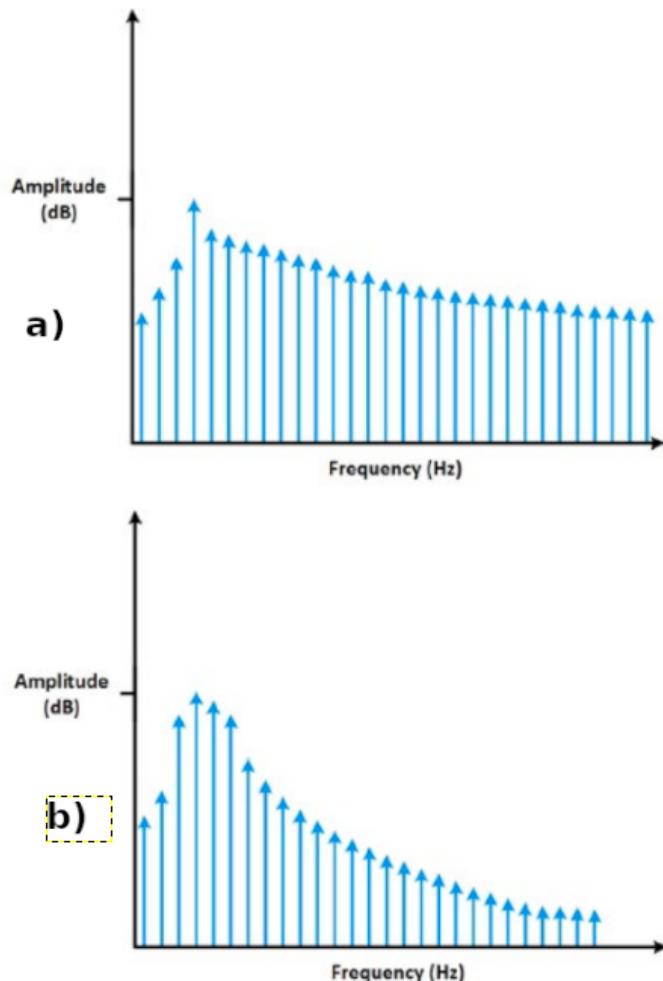


Figura 2.8: Comparativa de espectros frecuenciales calculados con DFT. En a) no se aplica ningún tipo de función de ventana a la onda compleja. En b) se ha aplicado una ventana de **Hann** antes realizar la DFT.

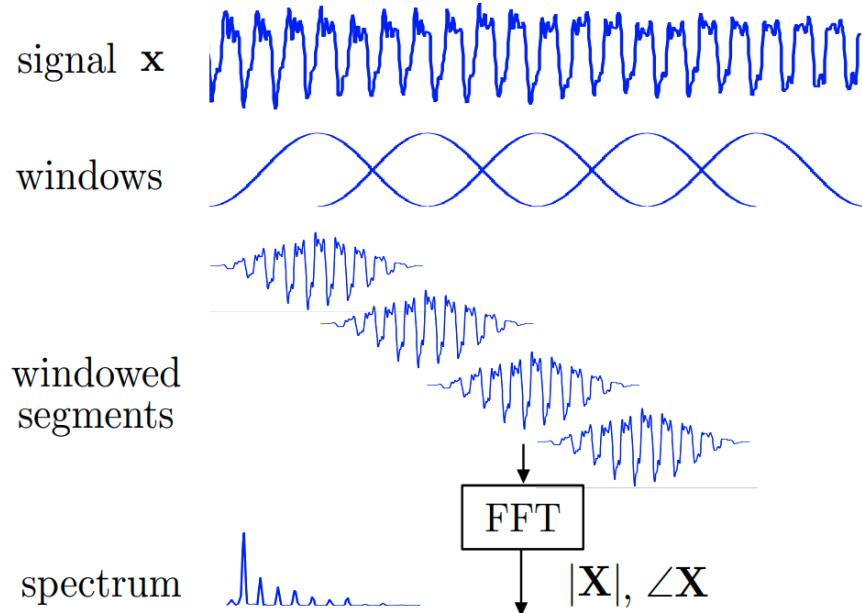


Figura 2.9: Resumen de la Short-Time Fourier Transform (STFT).

- **Número de muestras por bloque n.** Como explicábamos antes, aplicaremos DFT sobre unas pequeñas porciones de la señal original. Estos bloques o *frames* están compuestos por n muestras cada uno. A partir de ahora, en general, al tratar de valores discretos, ya no usaremos el tiempo como magnitud, sino el *número de muestras*¹⁰. En caso de que el número total de muestras de la señal no sea divisible por este valor n , se suele emplear **zero-padding**¹¹ para que cada *frame* llegue hasta este valor.
- **Tamaño de ventana M.** Es el número de muestras que serán multiplicadas por la función de ventana. El análisis de este algoritmo se vuelve muy complejo si $M \neq n$. Sin embargo, M puede ser menor que n . Cuando multiplicamos menos muestras por la función de ventana que el tamaño del *frame*, el resto valdrán 0.
- **Hop-length R.** Este parámetro nos permite describir cómo queremos que se solapen los *frames*. R es el número de muestras entre el comienzo de cada *frame*. R siempre es menor que n si queremos que haya solape. Cuando $R = n$, no hay ningún solape.

La Figura 2.10 muestra gráficamente el valor de todas estas variables. Aquí, el valor más relevante es el **tamaño del frame**. Si aumentamos el tamaño del *frame*, obtendremos un mejor **resolución frecuencial** a costa de una menor **resolución temporal**. Esto es evidente,

¹⁰Evidentemente, el número de muestras es una representación directa de la magnitud temporal, ya que cada muestra representa un sonido de la misma duración. Esta duración está relacionada con la **frecuencia de muestreo**, es decir, la frecuencia con la cual se discretiza la señal analógica. num. muestras = $\nu_{muestreo} * T_{total}$

¹¹Rellenar con 0's un vector n-dimensional hasta ocupar un tamaño máximo para cada dimensión.

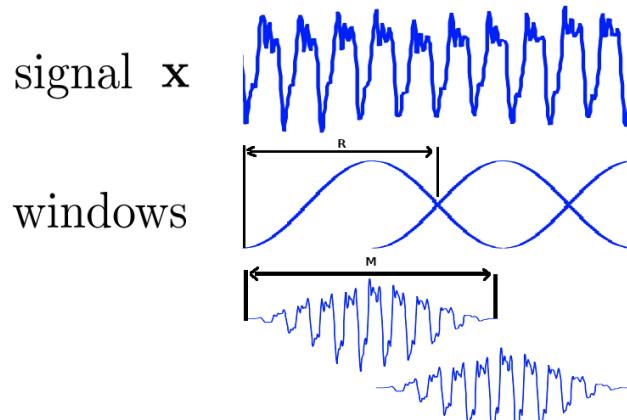


Figura 2.10: M es el tamaño de la ventana y R el *hop-length*. En este caso M es igual al número de muestras por *frame* n . Normalmente, en STFT suele ser así.

al tener menos bloques en total, es más difícil establecer diferencias frecuenciales entre los mismos. Sin embargo, al ser los bloques más largos, tenemos más ocasiones de observar la periodicidad de la onda compleja.

Recapitulando, formalmente, teníamos que la DFT se definía de la siguiente manera:

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-\frac{2\pi i}{N} kn}$$

Aquí, k es un alias para la **frecuencia**. $X(k)$ de x (la señal original) es la intensidad de la frecuencia k .

Formalmente, STFT se define de la siguiente manera:

$$S(m, k) = \sum_{n=0}^{N-1} x(mR + n) w(n) e^{-\frac{2\pi i}{N} kn}$$

donde m es el *frame* actual. Cabe destacar que el significado de N en las dos anteriores ecuaciones es distinto. En la primera N es el número de samples de toda la señal. En la segunda ecuación, N es el número de samples del *frame* (anteriormente referido como n).

También es interesante apreciar cómo el sumatorio se va a repetir de manera idéntica k veces. Por esto, cada vez que hemos empleado la DFT, podemos emplear un algoritmo mucho más eficiente conocido como **Fast Fourier Transform (FFT)**. Este algoritmo no repite k veces el mismo sumatorio, porque hace uso de una matriz en la cual va guardando resultados previamente calculados (programación dinámica iterativa).

Lo más relevante de la fórmula anterior es que **toma dos parámetros**: la frecuencia k y el *frame* m . De la misma manera que la DFT tiene dos dimensiones: frecuencia e intensidad; la STFT tiene **tres dimensiones**: frecuencia, tiempo¹² e intensidad. Estas tres dimensiones

¹²De igual manera que la muestra era un símil para la magnitud temporal, el número de *frame* m también lo es.

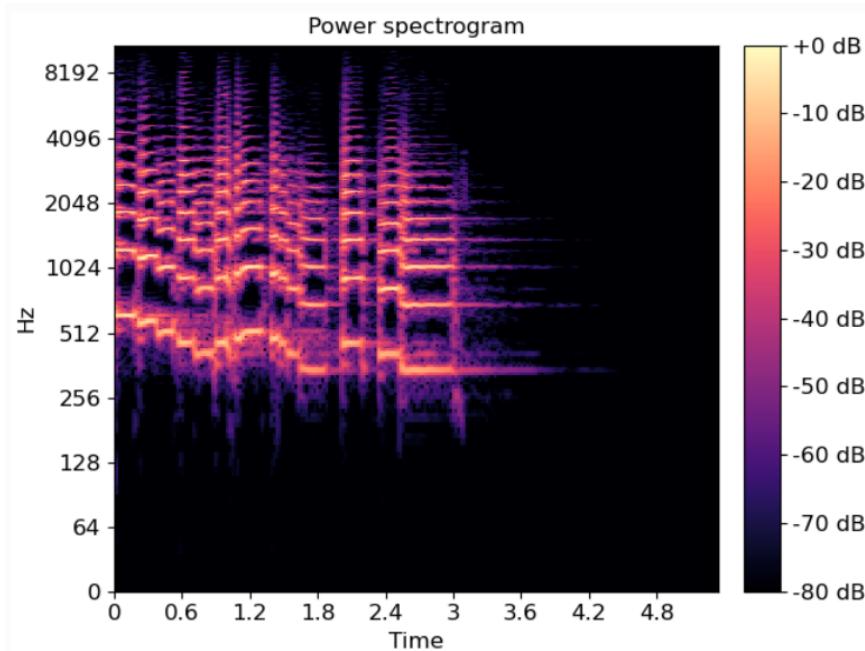


Figura 2.11: Espectrograma tiempo-frecuencia. En este caso se usa una escala logarítmica en el eje y. Esto es útil para resaltar las frecuencias fundamentales (que son las más graves).

son comúnmente representadas mediante dos ejes y un color, dando lugar al famoso **espectrograma** (ver Figura 2.11).

2.4 Automatic Music Transcription

Una posible definición para la transcripción musical automática es el proceso de traducción de una señal acústica en algún tipo de notación musical (Benetos y cols., 2013). Sin embargo, existen múltiples interpretaciones. También se puede entender como la conversión de una grabación a una notación de *piano-roll* (Cemgil, 2004) (ver Figura 1.7, subfigura c). En cualquier caso, es una tarea terriblemente compleja, ya que resulta difícil incluso para los propios músicos. Frecuentemente existen discusiones (subjetivas) sobre la correcta transcripción de una pieza musical (List, 1974).

AMT engloba muchas subtareas. Cabe detectar las múltiples alturas que suena simultáneamente, las notas, la aparición y desaparición de las mismas, la segregación en voces, la detección del compás, el análisis de los instrumentos de percusión, el reconocimiento tímbrico de los instrumentos, la cuantización... Todo esto sin contar con la elección de una codificación musical adecuada que nos permita representar lo mismo que se puede representar con la notación musical occidental.

Aún con todo, este campo es muy amplio. Existen distintos tipos de transcripción, según el nivel de abstracción al que queramos llegar:

- **Transcripción a nivel de frame.** Es el nivel más bajo. Consiste en la **detección de**

alturas. Normalmente, este problema se trata con música **polifónica**¹³. Esto es porque la estimación de la altura de un único instrumento y una única voz es prácticamente trivial (Benetos y cols., 2013). Esto se puede observar claramente en la Figura 2.11. Las frecuencias más bajas se corresponden con la altura. Aún así, existen algoritmos de estimación de frecuencias fundamentales para señales monofónicas como el de **YIN**. El problema de detección de alturas para música polifónica se conoce como **Multi-Pitch Estimation (MPE)** o *Multiple F0 Estimation*. Este problema consiste en la estimación del **número de notas** y la **altura de las mismas** para cada *frame* temporal.

- **Transcripción a nivel de nota.** También conocido como *note tracking*. Este nivel va un paso más alla que el anterior. Consiste en *conectar* las anteriores estimaciones entre *frames*. Evidentemente, una nota puede ocupar más de un único *frame*¹⁴. Para detectar qué alturas de qué *frames* se corresponden con qué notas, normalmente se post-procesa la salida del Multi-Pitch Estimation (MPE) (Benetos y cols., 2019). Existen múltiples maneras de hacer este post-procesado: filtrado de medianas (Su y Yang, 2015a), Modelos Ocultos de Markov (HMMs) (Nam y cols., s.f.) y redes neuronales (Valero-Mas y cols., 2018) y (Boulanger-Lewandowski y cols., 2012) . Cuando hablamos de **nota**, nos referimos a:
 - Una altura.
 - Un *onset*, es decir, un instante temporal en el cual comienza la nota.
 - Un *offset*, es decir, un instante temporal en el cual deja de existir la nota.
- **Transcripción a nivel de flujo.** También conocida como Multi-Pitch Streaming (MPS). Se trata de la agrupación de las anteriores notas en flujos. Por flujo nos podemos referir a instrumento (estariamos hablando de una combinación de Music Source Separation (MSS) y Automatic Music Transcription (AMT)) o voz. Esta claro que para separar el conjunto de notas en sus distintas líneas melódicas (que no deja de ser una tarea subjetiva y en la que muchos musicólogos entrarían en debate) no se pueden usar las mismas técnicas que en el apartado anterior. Esto es porque se necesita información *contextual* y no se puede analizar únicamente un *frame*. Cuando cada voz está representada por un tipo de instrumento distinto¹⁵, a este tipo de análisis se le conoce como *timbre-tracking* o *instrument tracking*. Existen pocos estudios que aborden este problema, debido a su complejidad (Benetos y Dixon, 2013).
- **Transcripción a nivel de partitura.** La transcripción a nivel de flujo nos da una salida discreta *paramétrica*. Aún es necesario dar el paso a la notación musical occidental, la partitura. Esto es un problema terriblemente complejo que abarca desde la comprensión de la harmonía y el ritmo hasta la separación en voces. Uno de los muchos problemas a los que se enfrenta este nivel de análisis es la elección de la escala. Por ejemplo, si una música está en Fa mayor pero frecuentemente emplea la nota Si natural, se podría interpretar que la escala es propiamente Do mayor. En definitiva, existe un gran nivel de *subjetividad*.

¹³La polifonía es un tipo de textura musical en la cual existen múltiples **voces**.

¹⁴De hecho, normalmente lo hará, ya que los *frames* suelen durar 10ms.

¹⁵Esto no es algo habitual. Por ejemplo, el piano puede tener muchas voces distintas, o en una orquesta existen violines primero y segundo.

Si bien es cierto que existen sistemas que traducen de notación MIDI a notación musical, los resultados dejan mucho que desear.

A continuación destacaremos algunos de los problemas más prevalentes dentro del AMT:

- Se trata de un problema **muy heterogéneo**. La polifonía es una textura muy compleja. Cada instrumento tiene su propio timbre, sus propios harmónicos, su propia voz, su propia intensidad...
- Existe un **gran solape** entre las distintas voces. Como se explicaba en la sección 2.1, cada nota realmente está compuesta por muchos parciales distintos. Por ejemplo, en el acorde de Do mayor (Do, Mi, Sol), van a existir parciales harmónicos en Do que también serán compartidos por los del Mi o los del Sol. En concreto, el Do comparte un 46,7% de sus harmónicos con las otras dos notas, el Mi un 33,3% y el Sol 60%. Entonces, ¿cómo sabemos que se están tocando esas tres notas?
- El *tempo* no está regido por el compás, sino por el director y el resto de intérpretes. Es decir, cuando un músico está interpretando su voz, no se tapa los oídos y lee la partitura, sino que presta una gran atención al resto de voces. De esta manera, se viola el principio de independencia de las voces, haciendo de la música un gran conjunto en el que todas las voces se influencian mutuamente.
- Existen pocos datos etiquetados para la transcripción a nivel de partitura (*audio to score*). Si bien es cierto que existen muchas partituras, éstas no son buenos valores de verdad (*ground-truth*) para el AMT. Principalmente porque las partituras **no están alineadas** con respecto al audio. El proceso de etiquetado tiene que ser llevado a cabo por intérpretes y músicos profesionales. Es un proceso largo, tedioso, caro, subjetivo y altamente susceptible a errores humanos. Además, es difícil establecer una **representación simbólica** que sea buena para el entrenamiento y que permita utilizar varias voces (polifonía). Sin embargo, sí que existen muchos *datasets* aptos para el aprendizaje supervisado de modelos de traducción a nivel de *frame* y de nota.

Actualmente existen, fundamentalmente, dos métodos para realizar la amt: Non-negative Matrix Factorization (NMF) y redes neuronales (NNs). En general, los métodos de NMF están siendo sustituidos por métodos de redes neuronales. Sin embargo, existen ventajas e inconvenientes para cada método. Las redes neuronales suelen hacer uso de **redes recurrentes** o **Recurrent Neural Networks (RNNs)**. Estas redes son capaces de aprender información contextual sobre los *frames* anteriores y posteriores. De esta manera, se obtiene un resultado global mejor respecto a la NMF. Sin embargo, las redes neuronales tienen un gran inconveniente: requieren de una **enorme cantidad de datos etiquetados** que, como comentábamos anteriormente, son muy difíciles de obtener. Además, la NMF tiene una gran ventaja y es la capacidad que tiene para ser **reentrenada** (Benetos y cols., 2019). Es decir, NMF puede ser entrenada con unas condiciones acústicas determinadas, pero en caso de que cambiaren (por ejemplo, podría cambiar ligeramente el timbre del piano), con tan sólo unos segundos de audio con las nuevas condiciones, se obtendrían resultados similares. Sin embargo, en el caso de redes neuronales, con tan sólo unos segundos de audio, los errores serían superiores al NMF por órdenes de magnitud (Ewert y Sandler, 2016).

¹⁵Por ejemplo, el dataset MAESTRO cuenta con más de 200 horas de interpretación virtuosísticas para piano.

3 Objetivos

El principal objetivo de este TFG es **implementar un sistema de transcripción automática** a nivel de partitura (*audio to score*).

Tal y como se ha explicado anteriormente, la casuística es extremadamente compleja y es, de hecho, un campo activo de investigación. Por este motivo, se relajarán ciertas restricciones para poder llegar a resultados válidos. En concreto, se tomará como entrada **señales de audio monofónicas** (con una única voz). Además, la representación simbólica final no tendrá porqué ser gráfica (equivalente a una partitura escrita real), sino que se emplearán los propios símbolos de la notación.

El proceso de estimación de F0¹ para música monofónica es relativamente sencillo, ya que ésta es la frecuencia predominante. Sin embargo, con el fin de exemplificar algunas técnicas de AMT, se abordará el problema con una **red neuronal**.

En concreto, se trata de:

1. Generar los datos necesarios. Es decir, analizar las señales de audio y extraer las características correspondientes. Este paso incluye todo tipo de preprocesamiento.
2. Establecer las etiquetas (*ground-truth*) acordes a cada dato de entrada. Cabrá escoger un **sistema de notación simbólica** que brinde buenos resultados.
3. Escoger la arquitectura de la red. Aquí se incluirán redes ampliamente estudiadas y utilizadas como las convolucionales.
4. Establecer una métrica de error de entrenamiento y otra de validación.
5. Escoger los parámetros de entrenamiento necesarios y entrenar la red.
6. Intentar encontrar mejoras en la arquitectura de la red o cualquier otra parte del sistema. Establecer comparaciones entre las distintas iteraciones del sistema.

¹Frecuencia fundamental.

4 Metodología

A lo largo de este Capítulo explicaremos la aproximación al problema de la **Automatic Music Transcription (AMT)**. En este caso, entenderemos el problema como uno de **clasiificación**¹ y emplearemos técnicas de aprendizaje automático basadas en redes neuronales. En concreto, usaremos una red neuronal convolucional, una red recurrente y el método de entrenamiento **Connectionist Temporal Classification (CTC)**. A este tipo de red se le puede llamar con el acrónimo CRNN (Convolutional Recurrent Neural Network).

En general, podemos entender este problema como si se tratara de **Optical Character Recognition (OCR)**² (ver Figura 4.1). Intentaremos explicar este problema de arriba a abajo, empezando por los datos de entrada y acabando por la representación simbólica.

4.1 Red Neuronal

La entrada es la imagen spectrograma tiempo-frecuencia de la muestra de audio (ver Figura 2.11). En nuestro caso, el corpus³ cuenta únicamente con ficheros MIDI. El primer paso es usar algún instrumento virtual para generar las señales de audio. En este caso, con la herramienta `timidity` generamos los fichero con codificación WAV usando un piano virtual con una tasa de muestreo de 44.1KHz. Después procedemos a calcular la Short-Time Fourier Transform (STFT) con `hop_length` de 512 muestras y `window_length` de 2048 muestras. La función de ventana es la de Hann. A la hora de graficar el spectrograma usamos una escala lineal en el eje Y. El resultado final se puede ver en la Figura 4.2. El único post-procesado que se hace de esta imagen es un escalado (la altura tiene que ser fija) y una reducción a un único canal de color (blanco y negro).

Sobre esta imagen aplicaremos **filtros de convolución** (ver Figura 4.3). Los filtros (o *kernels*) de convolución son matrices que son aplicadas sobre la imagen. Cada píxel se convierte en una combinación lineal de sus píxeles adyacentes, en la que los coeficientes son los valores de la matriz de convolución. Estos coeficientes son aprendidos durante el *backpropagation*. Cabe establecer el número de filtros y el tamaño del *kernel*.

Tras cada filtro de convolución se aplica un **pooling**. El *pooling* es sencillamente un *agrupamiento* de los píxeles. Normalmente se escoge el valor máximo de cada kernel, pero también se puede aplicar la media o incluso el valor mínimo. Cabe destacar que cuando se aplica una capa de *pooling* se reduce la dimensionalidad.

Tras aplicar varias capas de convolución y pooling, en definitiva, la red está aprendiendo características sobre la “forma” de la imagen. Y con las capas de pooling se está comprimiendo

¹Realmente el problema es conocido como **Sequence Labeling**. Dada una entrada, se intenta encontrar una secuencia de símbolos que la describa. Sin embargo, en definitiva, la elección de cada símbolo es el resultado de un proceso de clasificación.

²De hecho, para entender bien el uso de la CRNN, empecé implementando una solución para OCR.

³Cuerpo de datos sobre el que entrenaremos y validaremos el modelo.

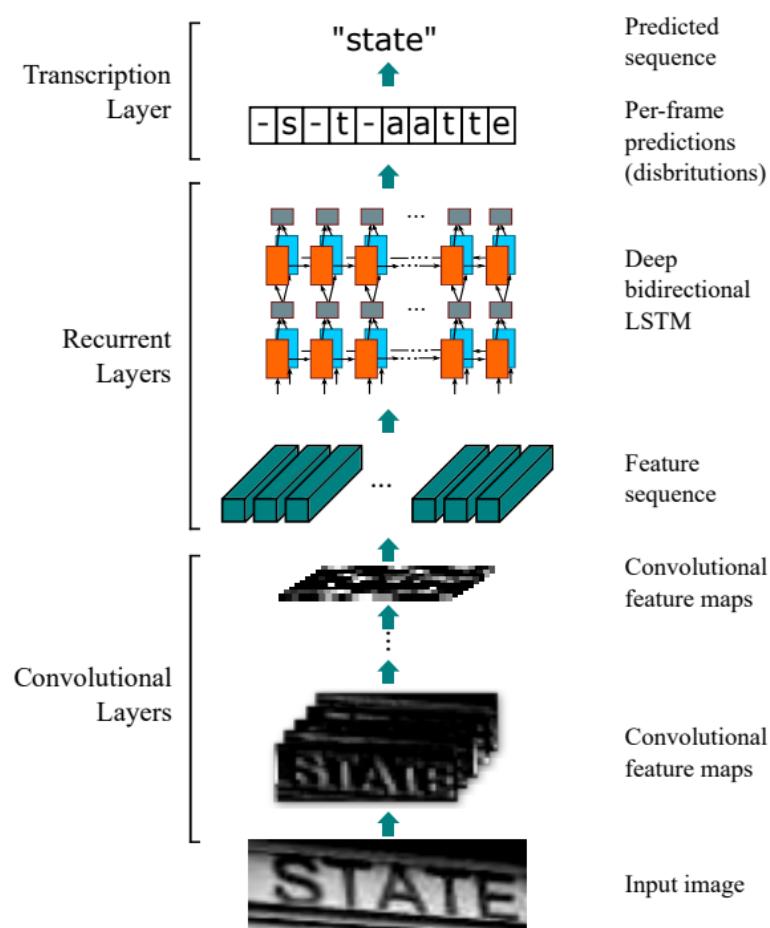


Figura 4.1: Arquitectura de una red CRNN (Convolutional Recurrent Neural Network).

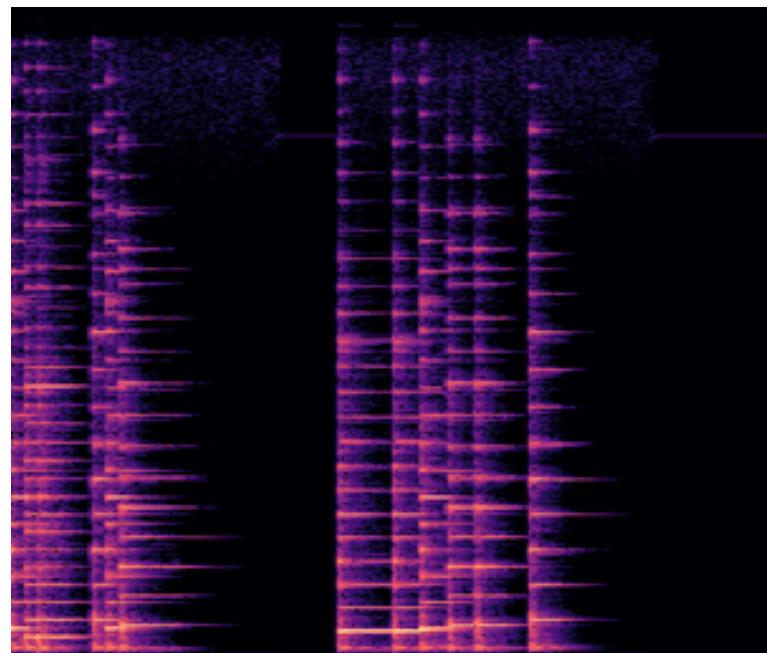


Figura 4.2: Ejemplo de espectrograma sintético. Se puede apreciar cómo apenas existen frecuencias fuera de los parciales. Es decir, hay poca *inharmonicity* (al tratarse de un instrumento sintético).

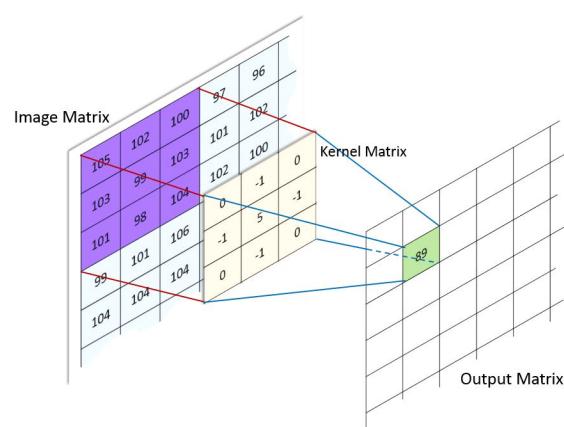


Figura 4.3: Ejemplo de aplicación de un *kernel* de convolución.

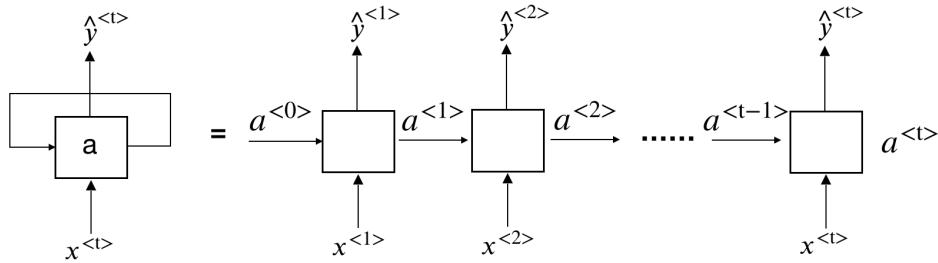


Figura 4.4: Ejemplo de red recurrente. Podemos ver que para cada instante, se proporciona una salida y_t (que depende de las anteriores por a_{t-1} , pero también otra salida a_t que se conecta al siguiente *frame*. De esta manera, se introduce el concepto de temporalidad y de memoria.

esta información. Al final, se puede entender que la imagen original ha sido segmentada y sobre cada segmento se ha obtenido un vector de *características*.

Sobre cada vector de características se aplica una capa **recurrente**. En este caso dos redes **Long Short-Term Memory (LSTM)**. Las redes recurrentes son redes cuya entrada depende de su propia salida (ver Figura 4.4). Son empleadas con datos **secuenciales** de longitud indeterminada, como es el caso de las señales de audio, o las imágenes⁴. En este caso, sirve para introducir una dimensión temporal al problema de aprendizaje. De hecho, emplean el algoritmo de *backpropagation through time* para minimizar la función de error. En definitiva, las redes recurrentes tienen un **estado interno** que es un alias para la memoria. Una red Long Short-Term Memory (LSTM), cuando analiza el segmento (*frame*) t_4 tiene en cuenta las salida de la red t_3 , que a su vez depende de la $t_2\dots$. Además, se puede conectar la salida de la red del *frame* siguiente a la propia, con lo que se obtiene una red LSTM **bidireccional**. Éstas tienen memoria tanto de los *frames* anteriores como de los siguientes.

Finalmente, se usa una capa densa normal para proporcionar la salida *softmax* para cada categoría. Las categorías son el *ground-truth*, es decir, las etiquetas válidas. Son la transcripción a notación **simbólica**. En este caso hemos acabado escogiendo una notación apodada **semantic** (ver Figura 4.5). Esta notación tiene un total de **1781 tokens distintos**. Hay que tener en cuenta que se tienen que codificar todas las notas con todas las alteraciones y duraciones posibles. Además de los silencios, los compases y las claves. Para agilizar el proceso de traducción entre un **string** como **clef-G2** y el índice del token correspondiente (387, en este caso), se emplea un clase auxiliar **SemanticTranslator** (ver código 4.1). Esta clase tiene dos estructuras de datos, un array y un diccionario. El array tiene como claves los tokens (strings) y como valor el índice. De esta manera, usando una tabla hash se puede traducir en tiempo constante $O(1)$. En el caso del array, el índice es el número de token y el valor es el propio token. Estas dos estructuras de datos están serializadas mediante **pickle** y guardadas en memoria secundaria.

Código 4.1: Clase SemanticTranslator

⁴El ancho de spectrograma (la dimensión temporal) es variable para cada señal de audio (no todas las muestras duran lo mismo).

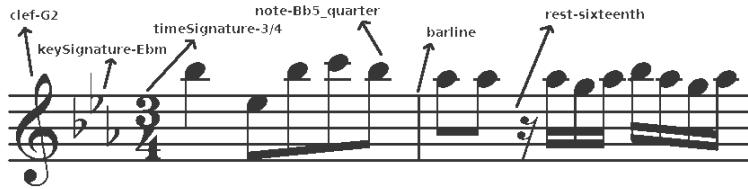


Figura 4.5: Notación simbólica *semantic*.

```

1 class SemanticTranslator:
2
3     def __init__(self, semantic_dict, semantic_array):
4         self.semantic_dict = semantic_dict
5         self.semantic_array = semantic_array
6         self.blank_class = len(semantic_array)
7
8     def encode_semantic_token(self, semantic_token : str) -> int:
9         return self.semantic_dict[semantic_token]
10
11    def decode_semantic_class_index(self, index : int) -> str:
12        return self.semantic_array[index]

```

Con esto, la salida de la CRNN tendrá 1781 dimensiones **para cada frame t_i** . Cada *frame* se corresponderá con las características de una región rectangular de los píxeles del espectrograma. El tamaño de esta región depende del factor de reducción. Éste depende a su vez del **número de capas de pooling** y del tamaño de las mismas⁵. En la arquitectura que se presenta se usan espectrogramas de es 256 píxeles de alto y de ancho variable. En la última capa de convolución se emplean 128 kernels. Con esto cada *frame* t_i tiene asociado un vector de $256/2^3 * 128 * 2^3 = 4096$ características que se corresponden a una región de de 8 píxeles de ancho por 256 de alto. Esta es la entrada a cada nodo de la LSTM. La tabla 4.1 muestra la arquitectura en detalle. Ésta incluye capas de *dropout* para mejor generalización y capas de *batch normalization* para acelerar el aprendizaje.

A la salida de la Convolutional Recurrent Neural Network (CRNN) tenemos un vector de predicciones *softmax* para cada token. A continuación explicaremos la métrica de error empleada para que el modelo aprenda.

4.2 Connectionist Temporal Classification

Para calcular el error tenemos que establecer una métrica entre el vector de *softmax* (una matriz, en definitiva) a la salida de nuestra red con las etiquetas reales *ground-truth*. La gran problemática de este modelo, al igual que sucede con Optical Character Recognition (OCR), es que **el tamaño de la entrada no se corresponde con el de la salida**. Para resolver este problema usaremos **Connectionist Temporal Classification (CTC)** (Graves y cols., 2006).

Supongamos una única predicción de nuestra red. Al pasar por la última capa de la CRNN, existe una matriz de $l \times m$ dimensiones, donde l es el número de *frames*⁶ (o *timesteps*) y m

⁵El ancho de cada *frame* (en píxeles) será igual a p^s , donde p es el número de capas de *pooling* y s es el ancho del filtro (que se entiende como cuadrado).

⁶Cabe destacar que l dependerá del ancho del espectrograma, que a su vez depende de la duración original

Tipo de capa	Forma de salida	Número de parámetros
padded_images	[(None, 256, None, 1)]	0
conv2d	(None, 256, None, 64)	640
max_pooling2d	(None, 128, None, 64)	0
conv2d_1	(None, 128, None, 64)	36928
max_pooling2d_1	(None, 64, None, 64)	0
conv2d_2	(None, 64, None, 64)	36928
batch_normalization	(None, 64, None, 64)	256
max_pooling2d_2	(None, 32, None, 64)	0
conv2d_3	(None, 32, None, 128)	73856
dropout	(None, 32, None, 128)	0
permute	(None, None, 32, 128)	0
reshape	(None, None, 4096)	0
bidirectional	(None, None, 256)	4326400
batch_normalization_1	(None, None, 256)	1024
bidirectional_1	(None, None, 256)	394240
dense (Dense)	(None, None, 1782)	457974

Tabla 4.1: Arquitectura de la Convolutional Recurrent Neural Network (CRNN)

es el número de características (1781).

Por otro lado, las etiquetas reales *ground-truth* para esa muestra de audio son un vector de $u \leq l$ dimensiones. En definitiva $u \neq l$ para la mayoría de los casos, con lo que tiene que existir algún tipo de **alineamiento** el cual maximice la probabilidad de escoger las etiquetas correctas. Cabe puntualizar que un token de la etiqueta puede ocupar varios *frames*, es decir, que varias predicciones iguales seguidas se han de “concatenar” como una única predicción. Para cambiar de token predicho a lo largo del tiempo es necesario introducir un nuevo carácter: **el token blanco** (o token vacío). La Figura 4.6 explica el uso de este carácter.

Sin embargo, aún queda por ver cómo calcular la métrica de error, que ha de tener en cuenta todos los posibles alineamientos. Tomemos la Figura 4.7 como ejemplo. Supongamos que la salida real es **a** ($l = 2$ y $u = 1$). Existen tres posibles predicciones válidas:

1. **a, a**

2. **-, a**

3. **a, -**

Hay que “premiar” a la red por usar cualquiera de estas posibles combinaciones. Es decir, hay que tener en cuenta todas ellas para calcular el error. Para ello, simplemente se calculan todos los posibles caminos (3, en este caso) y se realiza el productorio de las probabilidades. Finalmente, se suman las probabilidades de todos los posibles caminos. En este ejemplo, la probabilidad conjunta de realizar una predicción correcta es $0,4 \cdot 0,4 + 0,4 \cdot 0,6 + 0,6 \cdot 0,4 =$

de la muestra.

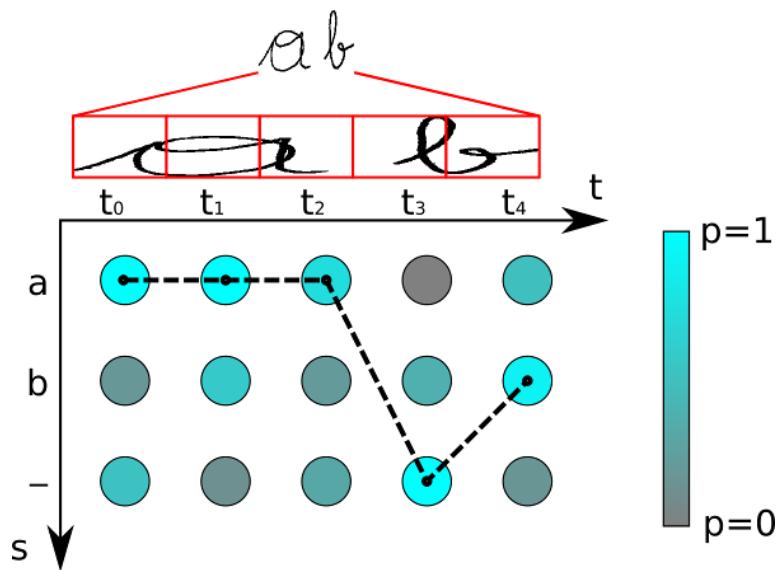


Figura 4.6: Ejemplo de decodificación en CTC. Podemos apreciar cómo varios *frames* se corresponden con la misma salida y el uso del carácter en blanco (-) para cambiar de la salida a a la b.

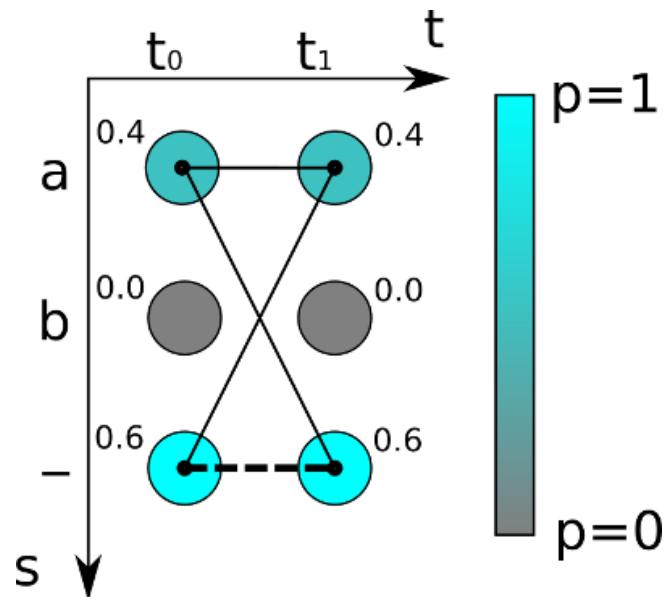


Figura 4.7: Ejemplo de predicción en CTC.

0,64. Como el Stochastic Gradient Descent (SGD) se entiende como un problema de minimización, más que la probabilidad, nos interesa calcular el **error**. El error simplemente será el logaritmo negativo de la probabilidad (0,19 en este caso).

De lo anterior se puede deducir que el número de posibles caminos válidos (que nos lleven a la etiqueta correcta) puede llegar a ser igual a lu^u en el peor de los casos. Es decir, se trata de un algoritmo de complejidad **exponencial**. Sin embargo, se puede entender que muchos de estos caminos están formados por composición de otros prefijos más cortos previamente calculados. Mediante programación dinámica se pueden obtener complejidades polinómicas en vez de exponenciales, que es precisamente lo que hace CTC.

Finalmente, quedaría realizar una predicción sobre la matriz de softmax. Para esto simplemente se puede emplear un algoritmo voraz. La Figura 4.6 explica claramente el proceso. Simplemente se escoge la probabilidad más elevada para cada *frame*. Finalmente se eliminan los caracteres repetidos y caracteres en blanco. La implementación se puede ver en el fragmento de código 4.2.

Código 4.2: Función de decodificación para CTC.

```

1 def _decode_softmax_indices(prediction) -> List[int]:
2     best_indices = [ np.argmax(x) for x in prediction ]
3     indices_no_repeated = _remove_repeated_tokens(best_indices)
4     class_pred_list = [ x for x in indices_no_repeated if x != semantic_translator.blank_class]
5     return class_pred_list

```

4.3 Mini-Batch Gradient Descent

El gran problema de las redes neuronales profundas es la enorme cantidad de datos necesarios para que los modelos aprendan y generalicen. Además, cabe considerar que la entrada a nuestra red son imágenes de ancho variable, pero con un mínimo de 65536 características. Con todo esto, se hace físicamente imposible cargar todas las imágenes⁷ en memoria. Para resolver esta problemática existen dos alternativas al *Batch Gradient Descent*:

1. **Stochastic Gradient Descent.** Se toma un único ejemplo para calcular el gradiente de la función de error. Posteriormente se actualizan los pesos usando una coeficiente de aprendizaje.
2. **Mini Batch Gradient Descent.** Se toma un número de ejemplos $n < N$ para calcular el gradiente.

Dado el tamaño de las imágenes y el alto número de parámetros de la arquitectura, con una GPU con 6 GiB de memoria, se podía establecer un **BATCH_SIZE** de 5. Con esto se acelera mucho el proceso de convergencia.

Sin embargo, el aprendizaje por Mini-Batch Gradient Descent no es trivial para una métrica de error como CTC. Esto es porque **cada sample del batch tiene una dimensión distinta**. En concreto varía el ancho del espectrograma, que es directamente proporcional a la duración de la señal de audio.

⁷El corpus de entrenamiento cuenta con más de 85 mil imágenes.

La red tiene que aplicar exactamente el mismo número de operaciones a cada *sample* del *batch*. Por ello, es necesario **igualar las dimensiones** de todas las entradas. En este caso opto por hacer un ***padding horizontal*** en todas las imágenes. Se iguala el ancho de todas las imágenes del *batch* a la anchura máxima de es *batch*. Al hacer *padding* con 0's, el resultado es añadir columnas en negro por la derecha de cada imagen. Sin embargo, esto acarrea un problema y es el aumento en tiempo de ejecución. La red tendrá que ejecutarse sobre todas las columnas en negro. Si establecemos un **BATCH_SIZE** muy elevado, esto aumentará la posibilidad de que haya una mayor diferencia entre tamaños para cada muestra. Esta diferencia de tamaños se traducirá en más columnas en negro y más tiempo de ejecución perdido.

El fragmento de código 4.3 se corresponde con la generación de los *batches* de entrenamiento. Al igual que se hace padding de las imágenes, también se ha de hacer padding de las etiquetas *ground-truth*. En el caso de las etiquetas, se hace padding con el carácter en blanco, que está codificado con la última posición en la lista de posibles tokens (1781). A la hora de calcular el error CTC es necesario también indicar al algoritmo qué porción de la matriz de características se corresponde con la imagen original y qué porción se corresponde con las características de las columnas negras. Lo mismo sucede con las etiquetas.

Código 4.3: Función de generación de *batches* de aprendizaje.

```

1 def __gen_train_batch(images, encodings, blank_token):
2     max_img_len = max([x.shape[1] for x in images])
3     max_encoding_len = max([len(x) for x in encodings])
4     original_image_lengths_after_pooling = np.array([x.shape[1] // POOLING_RATIO for x in images])
5     original_encoding_lengths = np.array([len(x) for x in encodings])
6     images = [ImageProcessing.pad_img_horizontal(x, max_img_len) for x in images]
7     encodings = [__pad_encoding(x, max_encoding_len, blank_token) for x in encodings]
8     return {
9         'padded_images': np.array(images),
10        'padded_encodings': np.array(encodings),
11        'original_image_lengths_after_pooling': original_image_lengths_after_pooling,
12        'original_encoding_lengths': original_encoding_lengths
13    }

```

Además, cabe hacer una comprobación fundamental. El ancho de la imagen es variable, **pero ha de tener un valor mínimo**. En concreto, el ancho de la imagen dividido entre el **POOLING_RATIO** ha de ser mayor que el número de del *ground-truth*. El fragmento de código 4.4 realiza esta comprobación. Por ejemplo, supongamos una imagen que representa la palabra **aeropuerto**. El ancho mínimo de la imagen, en píxeles, ha de ser $8 * 10 = 80$, ya que **aeropuerto** tiene 10 letras y se hace una predicción por cada 8 píxeles. Evidentemente, este es el ancho mínimo. Las imágenes pueden ser, en principio, infinitamente anchas (aunque las señales de audio tienen una duración de entre 3 y 15 segundos).

Código 4.4: Función de comprobación de tamaño mínimo del spectrograma

```

1 def __images_wide_enough(imgs, symbolic_seqs):
2     for img, symbolic_seq in zip(imgs, symbolic_seqs):
3         if img.shape[1] // POOLING_RATIO < len(symbolic_seq) + num_repeated_tokens(symbolic_seq):
4             return False
5     return True

```

4.4 Código

Toda la implementación se encuentra en el repositorio público:

<https://github.com/jorgeDonis/AudioTranscription>

Es necesario tener instalado Tensorflow v2.0 y es recomendable el uso de una GPU para el aprendizaje.

La rama `OCR` contiene una CRNN para resolver el problema de OCR y la rama `SyntheticAudio` trata el problema de AMT.

5 Resultados

En este capítulo analizaremos la bondad de nuestro modelo, es decir, la capacidad que tiene para transcribir sonidos a notación simbólica de manera automática. Primeramente hablaremos del conjunto de datos (*dataset*) empleado. Posteriormente hemos de establecer una **métrica de error**. Finalmente analizaremos los resultados obtenidos por los distintos modelos de predicción.

5.1 Dataset

El *dataset* empleado es **PrIMuS** (Printed Images of Music Staves), tomado prestado de Calvo-Zaragoza y Rizo (2018). Este *dataset* fue creado con el propósito de entrenar un modelo que realizara **Optical Music Recognition (OMR)**¹. PrIMuS contiene **87678 fragmentos de música real**. Estos fragmentos suelen ser el comienzo (que suele tener un carácter melódico) de la música. Estas músicas forman parte del *dataset* **RISM** (Répertoire International des Sources Musicales). Cada sample contiene 5 archivos:

1. El audio en formato **WAV**.
2. Una representación gráfica (en **PNG**) de la partitura.
3. La representación simbólica en el formato de la Music Encoding Initiative (MEI).
4. La representación simbólica **semántica** (ver Figura 4.5).
5. La representación simbólica **agnóstica**. Este tipo de representación es más gráfica y marca cada símbolo de manera independiente. Por ejemplo, en la notación simbólica la tonalidad de D mayor se representa como **KeySignature-DM**, mientras que la agnóstica es **accidental.sharp-L5**. Esto simplemente indica que hay un sostenido en la nota Fa. Este sostenido, que se encuentra en la armadura, es el mismo que el que podría haber una alteración de un compás posterior.

En nuestro caso, sólo nos interesa el audio (a partir del cual generaremos los espectrogramas) y la representación simbólica **semántica** (*ground-truth*).

5.2 Métrica de error

Esta métrica ha de ser capaz de medir la distancia entre dos vectores $p = [p_1, p_2 \dots p_m]$ y $q = [q_1, q_2 \dots q_n]$ donde m puede ser distinto de n . En nuestro problema, p tiene una dimensión

¹El Optical Music Recognition (OMR) es similar al Optical Character Recognition (OCR), sólo que en este caso se predicen tokens simbólicos musicales en vez de caracteres.

l , que es el número de *timesteps* en la predicción² y q tiene una dimensión u , que es el número de tokens en la etiqueta *ground-truth*. Tanto p como q tienen un dominio $\mathbb{D} = [0, 1781)$ (ya que existen 1781 tipos de tokens).

Para resolver este problema emplearemos la **distancia de edición de Levenshtein**. Ésta se define como el número de operaciones necesarias para transformar p en q . Supongamos $q = [2, 3]$. Las tres posibles operaciones son:

1. **Inserción..** $q \rightarrow [2, 3, x]$.
2. **Borrado.** $q \rightarrow [2]$
3. **Sustitución.** $q \rightarrow [2, x]$

De manera más general, se pueden establecer distintos pesos para cada una de estas operaciones. Sin embargo, en la implementación utilizada, **todas las operaciones tienen coste unitario**. Con esto, se puede deducir que el número de operaciones de edición no puede ser superior a $\max(m, n)$.

Aún con esto, es fundamental realizar un cálculo adicional, **normalizar** las distancias. Antes explicábamos que p y q no tenían porqué compartir la misma dimensionalidad. Sin embargo, a su vez pertenecen a un conjunto de vectores cada uno de ellos con dimensionalidad distinta. Esto es evidente, porque no todas las muestras tienen la misma duración. Con lo cual, es necesario realizar algún tipo de normalización. En nuestra implementación empleamos la **normalización uniforme**. Con esto, la distancia de edición $d(p, q) = [0, 1]$. Para normalizar, simplemente cabe dividir entre el número de tokens. El fragmento de código 5.1 indica el cálculo de la distancia de edición normalizada. `_edit_distance()` es la distancia de edición de Levenshtein (implementada en una librería de mismo nombre).

Código 5.1: Cálculo de la distancia de edición normalizada

```

1 def _get_loss(model, val_generator):
2     predictions, true_encodings = _predict_val_samples(model, val_generator)
3     total_loss = 0
4     for i in range(0, len(predictions)):
5         l_d = _edit_distance(true_encodings[i], predictions[i])
6         total_loss += l_d / len(true_encodings[i])
7     return total_loss / len(predictions)

```

5.3 Modelos

5.3.1 Modelo inicial

Para el entrenamiento se emplearon 75.000 muestras de audio. Éstas tienen una duración de entre 3 y 15 segundos. El conjunto de test contaba con 5.000 muestras. Los espectrogramas tenían un alto de 256 píxeles. Se eligió un tamaño de *batch* de 3 durante el entrenamiento. La Figura 5.1 indica la evolución del error a lo largo del aprendizaje. Cada época tardó en entrenarse aproximadamente 57 minutos.

²La predicción se obtiene decodificando la matriz softmax mediante el algoritmo de la Figura 4.6, tal y como se explicaba en la Sección 4.2.

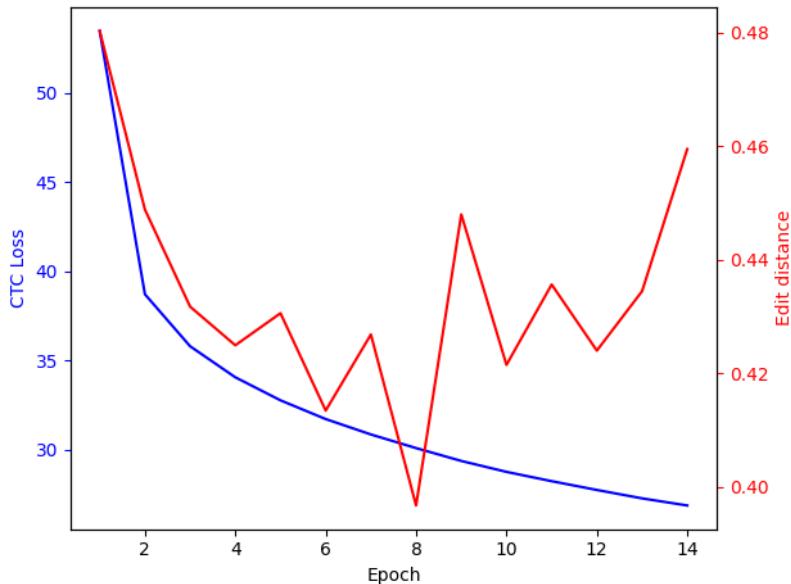


Figura 5.1: Evolución del error dentro y fuera de la muestra durante el proceso de aprendizaje.

Podemos apreciar cómo se consigue un error de validación mínimo de **0,396** en la época 8. Después, empieza a haber *overfitting*. Siempre se guarda el modelo con el menor error de validación.

Un ejemplo de predicción se pueden observar en la Tabla 5.1. Por un lado, cabe destacar que tanto la duración como la altura de todas las notas es correcta; sin embargo, esto no suele suceder en casi ninguna predicción. Aún con esto, existen dos errores. El primero es **el compás**. La predicción del compás es más compleja de lo que puede parecer a simple vista. Por ejemplo los compases $\frac{2}{4}$ y $\frac{4}{4}$ pueden ser ambos empleados para marcar el ritmo en una misma música. Evidentemente, habrá el doble de compases en el primer caso. La diferencia entre la elección del primer o segundo compás es totalmente subjetiva y se debe a una serie de matices muy delicados. En definitiva, **es el acento el que marca el compás**. En un compás $\frac{4}{4}$ se sigue el patrón: *fuerte, débil, semifuerte, débil*. Mientras que en dos compases $\frac{2}{4}$ se seguiría un patrón *fuerte, débil, fuerte, débil*³. Pero estos términos de *fuerte* y *semifuerte* son muy subjetivos. Desde luego, es difícil apreciar esta relación de intensidades en una interpretación realizada con un instrumento virtual. En el caso de la predicción de la figura 5.1, se predice un $\frac{2}{4}$ cuando la notación original marcaba $\frac{4}{4}$. Aún con todo, existiría un error adicional y es que la **la longitud de la predicción es errónea**. Esto suele darse en la práctica totalidad de las predicciones, por una multitud de motivos. En concreto, en esta ocasión se debe a la **ausencia de una barra de compás**. Los errores con las barras de compás son muy comunes en las predicciones del modelo. En la Sección 5.3.3 se discutirá cómo solucionar este problema.

³Existe otro matiz muy relevante: en ocasiones existe una ligera pausa al final de algunos compases conocida como *cadencia*. Esta pausa no se da entre los tiempos de un compás. La cadencia suele emplearse para resaltar el final de una frase musical. Una vez más, el concepto de “frase musical” es subjetivo.

⁴En notación musical, un compás de $\frac{4}{4}$ es equivalente al tiempo común, **C** o $\text{C}\frac{4}{4}$.

n	Original	Predicción
1	clef-G2	clef-G2
2	timeSignature-2/4	timeSignature-C
3	note-C5_quarter	note-C5_quarter
4	note-C5_eighth	note-C5_eighth
5	note-E5_eighth	note-E5_eighth
6	barline	barline
7	note-D5_quarter	note-D5_quarter
8	note-D5_eighth	note-D5_eighth
9	note-F5_eighth	note-F5_eighth
10	barline	
11	note-E5_eighth	note-E5_eighth
12	note-C5_eighth	note-C5_eighth
13	note-B4_eighth	note-B4_eighth
14	note-C5_eighth	note-C5_eighth
15	barline	barline
16	note-E5_eighth	note-E5_eighth
17	note-B4_eighth	note-B4_eighth
18	note-G4_quarter	note-G4_quarter
19	barline	barline
20	note-C5_quarter	note-C5_quarter
21	note-C5_eighth	note-C5_eighth
22	note-E5_eighth	note-E5_eighth

Tabla 5.1: Ejemplo de predicción para el modelo inicial. Nótese que la predicción nº 10 contiene un vacío. Realmente la dimensionalidad de la tupla de predicción y el *ground-truth* es distinta. El vacío ha sido desplazado deliberadamente *a posteriori* para obtener un alineamiento más lógico.

n	Original	Predicción
1	clef-G2	clef-G2
2	keySignature-BbM	timeSignature-3/4
3	timeSignature-C	note-E5_quarter.
4	note-Eb5_quarter.	rest-sixteenth
5	rest-sixteenth	note-E5_sixteenth
6	note-Eb5_sixteenth	note-E5_quarter
7	note-E5_quarter	note-F5_eighth.
8	barline	note-F4_sixteenth
9	note-F5_eighth.	barline
10	note-F4_sixteenth	
11	note-F4_eighth.	note-F4_eighth.
12	note-F4_sixteenth	note-F4_sixteenth
13	note-F4_quarter	note-F4_quarter
14	rest-eighth	rest-eighth
15	note-F5_sixteenth	note-F5_sixteenth
16	note-C5_sixteenth	note-C5_sixteenth
17	barline	barline
18	note-Ab5_half	note-G#5_half
19	note-F5_quarter	note-F5_quarter
20	note-D5_quarter	barline
21	barline	note-D5_quarter
22	note-B4_quarter	note-B4_quarter

Tabla 5.2: Ejemplo de predicción. Al igual que en la Tabla 5.1, se mueve la predicción vacía a la posición 10 deliberadamente.

El ejemplo de predicción de la Tabla 5.2 nos muestra más errores comunes. En general, podemos ver cómo tanto la predicción de la altura de las notas como su duración es errónea. Se añaden notas erróneas (los Mi5 en los tokens 3 y 5) y se omiten otras (Eb5 en el token 6). A diferencia de la predicción de la Tabla 5.1, en este caso se predice un compás $\frac{3}{4}$ en vez de uno $\frac{4}{4}$. Este error es más grave, ya que estaríamos pasando de un ritmo binario a uno ternario⁵. No sólo el ritmo es incorrecto sino que el primer compás necesita una semicorchea con puntillo adicional para llegar a las tres unidades de tiempo. Aún con esto, el segundo y tercer compás⁶ sí que tienen 3 negras cada uno, lo cual es interesante. Es decir, la red ha aprendido a generar compases coherentes, aún con algunos errores.

Otro punto interesante a analizar de la Tabla 5.2 es el token nº 18. En este caso se predice un Sol#5 cuando el *ground truth* es un Ab5. Ambas notas representan la misma frecuencia (830.61 Hz). A este fenómeno se lo conoce por el nombre de **enarmonía**. Surge de la afinación *bien temperada*, que es un tipo de afinación ampliamente empleada desde el siglo XVII. En este tipo de afinación existen notas con exactamente la misma altura. Esto no sucede en otros sistemas de afinación como el *justo*. En definitiva, es muy complicado que el modelo

⁵Aún con todo, este cambio de ritmo se puede considerar como válido, de manera subjetiva.

⁶Excluimos el análisis del último compás, ya que se encuentra incompleto.

sepa si se trata de un Sol#5 o de un A#5. Habría que recurrir a la **escala** empleada. Sin embargo, establecer la escala también resultado complicado. En este fragmento se emplea la escala de Sib mayor; sin embargo, no aparece la tónica (Sib) en ningún momento. Para mayor confusión, se emplea un Si natural y una alteración en el A#5. De esta manera, incluso un agente humano tendría dificultades para determinar la escala únicamente contando con esta breve señal acústica. En mi opinión subjetiva, me atrevería a decir que la escala se ajusta más bien a un Do menor harmónico. En este caso, la red no llega a hacer una predicción sobre la escala⁷.

5.3.2 Cambio de spectrograma

El primer cambio que pensamos que podría mejorar la transcripción fue el spectrograma. La Figura 5.2 muestra los cambios realizados. En concreto:

1. Se cambia la altura de la imagen de 256 a 192 píxeles (perdiendo resolución).
2. Se establece un **umbral** mínimo de intensidad de -70 (valor de intensidad de píxel propio de `matplotlib`). Si no se supera este umbral, el píxel pasa a ser negro. También se establece un umbral máximo de 8.
3. **Se reduce el rango de frecuencias.** Para abarcar las 88 notas del piano virtual, se usa un rango de entre 27.5Hz (A0) y 3520Hz (A8).
4. Se cambia a un `hop_length` de 128 samples.
5. Se establece un tamaño de ventana y un número de muestras por bloque (n) de 1024 (ambos).
6. Se emplea una escala de **Mel** en la dimensión frecuencial.

La escala de Mel (de la palabra *melodía*) es una escala perceptiva de alturas basada en experimentos con oyentes. En estos experimentos se intenta establecer una escala tal que todas las frecuencias *suenen* equidistantes a oídos humanos. Existen varias escalas de Mel (según los experimentos). La fórmula empleada por `librosa` para pasar de f hercios (Hz) a m mels es la siguiente:

$$m = 2595 \log_{10} \left(1 + \frac{f}{700} \right)$$

Podemos apreciar como la escala es logarítmica. A primera vista, parece que la Subfigura 5.2b presenta varias ventajas frente a la Subfigura 5.2a. Por un lado, se usan menos frecuencias. Esto ayuda a eliminar información que podemos considerar como redundante. Cabe recordar que estamos tratando con transcripción de músicas *monofónicas*. Es decir, sólo va a existir una frecuencia fundamental en cada instante. El spectrograma de la derecha claramente muestra como se eliminan muchos harmónicos que son irrelevantes. Por otro lado, esta escala de Mel nos ayuda a **resaltar las frecuencias más graves**. Esto es idóneo, ya que las frecuencias más graves son las fundamentales, que son las que determinan la altura

⁷Esto sucede porque existen ejemplos en Do mayor, sin escala.

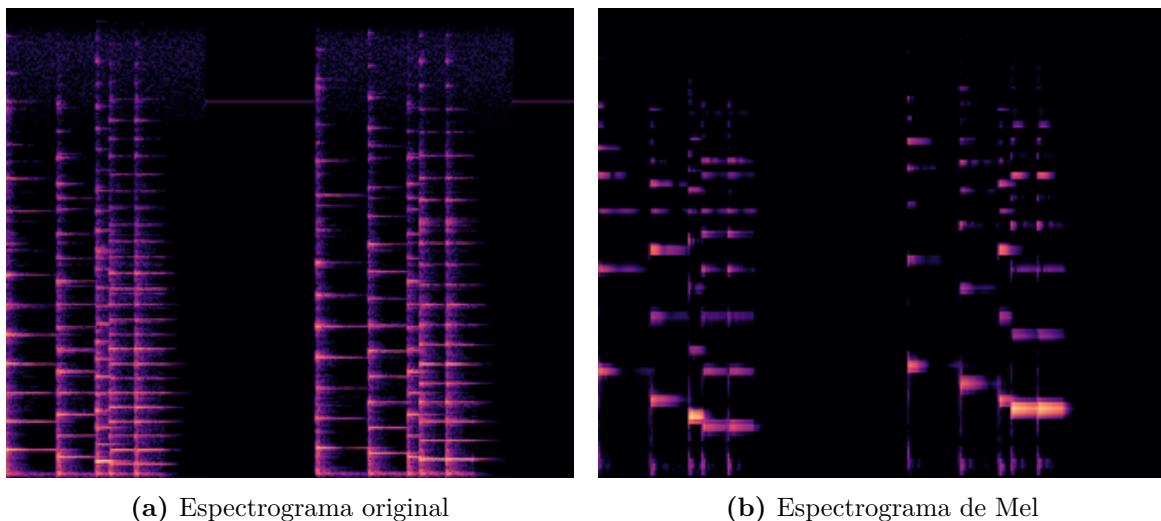


Figura 5.2: Comparativa entre los dos tipos de espectrogramas empleados. Ambos espectrogramas provienen de la misma muestra de audio. Ambas imágenes han sido reescaladas. Cabe destacar que el modelo trabaja con imágenes en escala de grises.

de las notas. Finalmente, la imagen tiene menos píxeles, con lo que se reduce el tiempo de entrenamiento y se permite usar tamaños de *batch* más grandes.

Con todo esto, se volvió a entrenar sobre el mismo conjunto de entrenamiento y validación, únicamente cambiando los espectrogramas. Si bien es cierto que se redujo considerablemente el tiempo de entrenamiento (el tiempo de entrenamiento por época bajó de 57 minutos a 48, aún aumentando el tamaño de *batch* de 3 a 16), no se consiguió reducir el error de validación. Se probó a cambiar la arquitectura, pero tampoco se obtuvieron mejores resultados.

La Figura 5.3 muestra la evolución del error a lo largo del aprendizaje. El mejor modelo arrojaba un error de validación de **0,42**, que sigue siendo peor que el del modelo inicial (0,39). Esto puede ser explicado por la reducción en el tamaño de la imagen. Es posible que al reducir la información se empeore el rendimiento del modelo.

5.3.3 Ajuste de compases

La última mejora que se propuso para el modelo fue el ajuste de compases. Tal y comentábamos en el análisis de la Tabla 5.2 en la Sección 5.3.1, **la predicción de los compases suele ser errónea**. En ocasiones se generan compases demasiado largos, cortos o incoherentes entre sí. Dada esta problemática, parece una buena idea **generar las barras de compás a posteriori**.

El fragmento de código 5.2 indica el algoritmo empleado. En esencia:

1. Se eliminan las barras de compás
2. Se itera sobre las sucesivas notas
3. Se añaden barras de compás cada vez que el valor relativo acumulado de las notas y

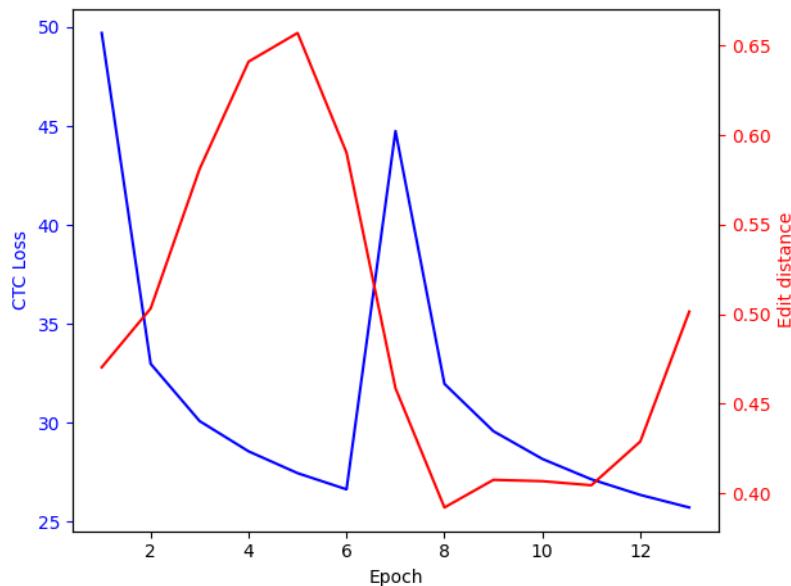


Figura 5.3: Evolución del error dentro y fuera de la muestra durante el proceso de aprendizaje. Se emplea el espectrograma con escala de Mel.

silencios leídos es igual o superior⁸ al compás empleado.

Código 5.2: Algoritmo de relleno de barras de compás.

```

1 def fix_rhythm(tokens : List[str]) -> List[str]:
2     fixed_tokens_no_barlines = [ x for x in tokens if x != 'barline' ]
3     fixed_tokens = []
4     i = 0
5     time_signature = None
6     while i < len(fixed_tokens_no_barlines):
7         fixed_tokens.append(fixed_tokens_no_barlines[i])
8         if 'timeSignature' in fixed_tokens_no_barlines[i]:
9             time_signature = _get_time_signature_frac(fixed_tokens_no_barlines[i])
10            i += 1
11            break
12        i += 1
13    if time_signature != None:
14        bar_relative_val = 0
15        while i < len(fixed_tokens_no_barlines):
16            relative_val = _get_relative_val(fixed_tokens_no_barlines[i])
17            fixed_tokens.append(fixed_tokens_no_barlines[i])
18            bar_relative_val += relative_val
19            if (bar_relative_val >= time_signature):
20                fixed_tokens.append('barline')
21                bar_relative_val = 0
22            i += 1
23    return fixed_tokens

```

En teoría este algoritmo debería de reducir el error de validación, sin embargo, llega a empeorarse sustancialmente (**0,47**). No obstante, aunque las transcripciones tengan una dis-

⁸Véase la línea de código nº19

n	Original	Corregido
1	clef-G2	clef-G2
2	timeSignature-C	timeSignature-C
3	note-C5_half	note-C5_half
4	note-E5_quarter.	note-E5_quarter.
5	note-F5_eighth	note-F5_eighth
6	note-G5_quarter	barline
7	note-F5_quarter	note-G5_quarter
8	barline	note-F5_quarter
9	note-E5_quarter	note-E5_quarter
10	note-D5_eighth	note-D5_eighth
11	note-E5_eighth	note-E5_eighth
12	note-C5_half	barline
13	note-G5_half	note-C5_half
14	note-A5_quarter	note-G5_half
15	note-A5_eighth	barline
16	note-Bb5_eighth	note-A5_quarter
17	barline	note-A5_eighth
18		note-Bb5_eighth

Tabla 5.3: Transcripciones con y sin corrección *a posteriori* de las barras de compás, respectivamente.

tancia de edición mayor respecto a los valores de verdad, esto no quita que tengan un **significado musical más correcto**. La Tabla 5.3 muestra un ejemplo de corrección de barras de compás.

6 Conclusiones

El modelo de transcripción presentado muestra claramente que **es posible traducir una señal de audio en una notación simbólica musical**. Por simplicidad, en este caso hemos trabajado con música monofónica, pero se puede extraer que este modelo, con ciertos ajustes, podría brindar buenos resultados para música polifónica.

Además, se demuestra que **una única red neuronal** puede, en un solo paso, transcribir audio en notación simbólica (modelo *end-to-end*). Esto es gracias al método de entrenamiento **CTC (Connectionist Temporal Classification)**. Podemos concluir que este método es, en general, una buena herramienta para resolver problemas de **etiquetado de secuencias**, como Optical Character Recognition (OCR)¹, Automatic Speech Recognition (ASR) o Handwritten Text Recognition (HTR).

Más en detalle, podemos comprobar que **el espectro frecuencial es una buena codificación de la información musical para Automatic Music Transcription (AMT)**. En concreto, la Short-Time Fourier Transform (STFT) y el uso de imágenes de espectrogramas parecen ser una buena representación gráfica de las muestras de audio.

Aún con todo, se desprenden algunas conclusiones aparentemente ilógicas de los experimentos. Parece sorprendente que la corrección *a posteriori* de las barras de compás no mejore el error de validación. Aún más sospechoso es que el uso de una escala de Mel no brinde mejores resultados. Sin embargo, el funcionamiento de las redes neuronales no deja de ser un tanto oscuro y en ocasiones es difícil extraer conclusiones lógicas cuando se trabaja con ellas².

Finalmente me gustaría recalcar un par de aspectos sobre el Music Information Retrieval (MIR). Este es un campo de investigación plenamente activo (aunque algo reducido). La investigación sigue activa precisamente porque aún estamos muy lejos de llegar a resultados prácticos. Sin embargo, quisiera destacar la relevancia y utilidad que puede llegar a tener la MIR en la educación artística y en la música en general. Sin duda, un futuro en el que las máquinas compusieran música o ayudaran a componerla sería apasionante.

¹Tal y como se comentaba anteriormente, primero, con fines didácticos, se implementó un modelo de reconocimiento de caracteres que también brindaba buenos resultados.

²Sin embargo, existen áreas de investigación en ML (Machine Learning) que intentan generar descripciones sobre precisamente esto. Por ejemplo, existen modelos que resaltan con algún color los píxeles de la imagen que más influencian en el error durante el proceso de entrenamiento cuando se usan redes convolucionales.

Bibliografía

- Adomavicius, G., Mobasher, B., Ricci, F., y Tuzhilin, A. (2011). Context-aware recommender systems. *AI Mag.*, 32, 67–80.
- Alexandre Donze, S. L. S. A. S. D. W., Rafael Valle. Ilge Akkaya. (s.f.). Machine improvisation with formal specifications.
- Benetos, E., y Dixon, S. (2013). Multiple-instrument polyphonic music transcription using a temporally constrained shift-invariant model. *The Journal of the Acoustical Society of America*, 133(3), 1727-1741. Descargado de <https://doi.org/10.1121/1.4790351> doi: 10.1121/1.4790351
- Benetos, E., Dixon, S., Duan, Z., y Ewert, S. (2019). Automatic music transcription: An overview. *IEEE Signal Processing Magazine*, 36(1), 20-30. doi: 10.1109/MSP.2018.2869928
- Benetos, E., Dixon, S., Giannoulis, D., Kirchhoff, H., y Klapuri, A. (2013, 01 de Dec). Automatic music transcription: challenges and future directions. *Journal of Intelligent Information Systems*, 41(3), 407-434. Descargado de <https://doi.org/10.1007/s10844-013-0258-3> doi: 10.1007/s10844-013-0258-3
- Boulanger-Lewandowski, N., Bengio, Y., y Vincent, P. (2012, junio). Modeling Temporal Dependencies in High-Dimensional Sequences: Application to Polyphonic Music Generation and Transcription. *arXiv e-prints*, arXiv:1206.6392.
- Bryan, N., y Sun, D. (2013, abril). Source separation tutorial mini-series ii:introduction to non-negative matrixfactorization. En *Dsp seminar*.
- Calvo-Zaragoza, J., y Rizo, D. (2018). End-to-end neural optical music recognition of monophonic scores. *Applied Sciences*, 8(4). Descargado de <https://www.mdpi.com/2076-3417/8/4/606> doi: 10.3390/app8040606
- Cano, E., Fitzgerald, D., Liutkus, A., Plumbley, M., y Stöter, F.-R. (2019). *Musical source separation: An introduction* (Vol. 36) (no 1). doi: ff10.1109/MSP.2018.2874719ff.ffhal-01945345
- Cemgil, A. (2004). *Bayesian music transcription* (Tesis Doctoral no publicada). Netherlands.
- Défossez, A., Usunier, N., Bottou, L., y Bach, F. (2021). Music source separation in the waveform domain.
- Ewert, S., y Sandler, M. (2016). Piano transcription in the studio using an extensible alternating directions framework. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 24(11), 1983-1997. doi: 10.1109/TASLP.2016.2593801

- Ferwerda, B., Schedl, M., y Tkalcic, M. (2015). Personality emotional states: understanding users music listening needs. En *Extended proceedings of the 23rd international conference on user modeling, adaptation and personalization (umap)*. Dublin, Ireland.
- Gillhofer, M., y Schedl, M. (2015). Iron maiden while jogging, debussy for dinner? an analysis of music listening behavior in context. En *Proceedings of the 21st international conference on multimedia modeling (mmm)*. Sydney, Australia.
- Graves, A., Fernández, S., Gomez, F., y Schmidhuber, J. (2006). Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks. En *Proceedings of the 23rd international conference on machine learning* (p. 369–376). New York, NY, USA: Association for Computing Machinery. Descargado de <https://doi.org/10.1145/1143844.1143891> doi: 10.1145/1143844.1143891
- Hagblade, M., Hong, Y., y Kao, K. (2011). Music genre classification..
- Kaminskas, M., Ricci, F., y Schedl, M. (2013). Location-aware music recommendation using auto-tagging and hybrid matching. En *Proceedings of the 7th acm conference on recommender systems (recsys)*. Hong Kong, China.
- Kasák, P., Jarina, R., y Chmulík, M. (2020). Music information retrieval for educational purposes - an overview. En *2020 18th international conference on emerging elearning technologies and applications (iceta)* (p. 296-304). doi: 10.1109/ICETA51985.2020.9379216
- Kiper, D. (2016). Physics of sound..
- List, G. (1974, septiembre). The reliability of transcription. *Ethnomusicology*, 18(9), 353-377.
- Liu, L., y Benetos, E. (2021). Handbook of artificial intelligence for music. En M. E.R. (Ed.), (p. 693-714). Springer. doi: 10.1007/978-3-030-72116-9_24
- Mcfee, B., y Lanckriet, G. (2012). Hypergraph models of playlist dialects. En *Proceedings of the 13th international society for music information retrieval conference (ismir)*. Porto, Portuga.
- Nam, J., Ngiam, J., Lee, H., y Slaney, M. (s.f.). *A classification-based polyphonic piano transcription approach using learned feature representations*.
- Orio, N. (2006, noviembre). Music retrieval: A tutorial and review. *now, 1(1)*. doi: 10.1561/1500000002
- Rentfrow, P., y Gosling, S. (2003). The do re mi's of everyday life: the structure and personality correlates of music preferences. *J Personal Soc Psychol*, 84(6), 1236–1256.
- Schedl, M., Zamani, H., y Chen, C. (2018). Current challenges and visions in music recommender systems research. *Int J Multimed Info Retr*, 7, 95–116. doi: 10.1007/s13735-018-0154-2
- Seyerlehner, K., Schedl, M., Pohle, T., y Knees, P. (2010. ISMIR 2010). Using block-level features for genre classification, tag classification and music similarity estimation. En *Extended abstract to the music information retrieval evaluation exchange (mirex 2010)/11th international society for music information retrieval conference*. Utrecht, the Netherlands.

- Su, L., y Yang, Y.-H. (2015a). Combining spectral and temporal representations for multipitch estimation of polyphonic music. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 23(10), 1600-1612. doi: 10.1109/TASLP.2015.2442411
- Su, L., y Yang, Y.-H. (2015b). Combining spectral and temporal representations for multipitch estimation of polyphonic music. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 23(10), 1600-1612. doi: 10.1109/TASLP.2015.2442411
- Understanding fft's and windowing [Manual de software informático]. (s.f.).
- Valero-Mas, J. J., Benetos, E., y Iñesta, J. M. (2018). A supervised classification approach for note tracking in polyphonic piano transcription. *Journal of New Music Research*, 47(3), 249-263. Descargado de <https://doi.org/10.1080/09298215.2018.1451546> doi: 10.1080/09298215.2018.1451546
- Wang, X., Rosenblum, D., y Wang, Y. (2012). Context-aware mobile music recommendation for daily activities. En *Proceedings of the 20th acm international conference on multimedia* (pp. 99–108). Nara, Japan: ACM.
- Zheleva, E., Guiver, J., Rodrigues, M., Milić-Frayling, E., y null, N. (2010). Statistical models of music-listening sessions in social media. En *Proceedings of the 19th international conference on world wide web (www)* (pp. 1019–1028). Raleigh, NC, USA.

Lista de Acrónimos y Abreviaturas

AMT	Automatic Music Transcription.
ASR	Automatic Speech Recognition.
CP	Correlation Pattern.
CRNN	Convolutional Recurrent Neural Network.
CTC	Connectionist Temporal Classification.
DFT	Discrete Fourier Transform.
DSP	Delta Spectral Pattern.
DTFT	Discrete-Time Fourier Transform.
FFT	Fast Fourier Transform.
FT	Fourier Transform.
HMM	Hidden Markov Model.
HTR	Handwritten Text Recognition.
IEEE	Institute of Electrical and Electronics Engineers.
LFP	Logarithmic Fluctuation Pattern.
LSTM	Long Short-Term Memory.
MIR	Music Information Retrieval.
MPE	Multi-Pitch Estimation.
MPS	Multi-Pitch Streaming.
MSS	Music Source Separation.
NMF	Non-negative Matrix Factorization.
NN	Neural Network.
OCR	Optical Character Recognition.
OMR	Optical Music Recognition.
RNN	Recurrent Neural Network.
SCP	Spectral Contrast Pattern.
SGD	Stochastic Gradient Descent.
SP	Spectral Pattern.
STFT	Short-Time Fourier Transform.
SVM	Support Vector Machine.
TFG	Trabajo Final de Grado.
VDSP	Variance Delta Spectral Pattern.