Escuela Politécnica Superior de Jaén

Grado en Ingeniería Informática

UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior de Jaén

Trabajo Fin de Grado

Anotación de acordes en piezas musicales con aprendizaje profundo

Alumno: Antonio Manuel Milla Lara

Tutor: Francisco Charte Ojeda

Dpto: Informática

UNIVERSIDAD DE JAÉN

D./Dª Francisco Charte Ojeda, tutor(es) del Trabajo Fin de Grado titulado: **Anotación de acordes en piezas musicales con aprendizaje profundo**, que presenta Antonio Manuel Milla Lara, autoriza(n) su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

Jaén, junio de 2021

El estudiante

Antonio Manuel Milla Lara

El tutor

CHARTE OJEDA FRANCISCO - 25996290B

Firmado digitalmente por CHARTE OJEDA FRANCISCO - 25996290B
Nombre de reconocimiento (DN): c=ES, serialNumber=IDCES-25996290B, givenName=FRANCISCO, sn=CHARTE OJEDA, cn=CHARTE OJEDA FRANCISCO - 25996290B
Fecha: 2021.06.28 08:47:08 +02'00'

Francisco Charte Ojeda

Agradecimientos

Me encantaría agradecer a todos mis amigos y familiares su gran apoyo durante todo este tiempo y por no dudar de mí en ningún momento.

Agradecer a Lou todo el cariño, apoyo y ayuda que he recibido en todo momento y que me ha hecho salir adelante y sacar lo mejor de mí.

A mi hermana Marta, por estar siempre ahí, ser un pilar fundamental de mi vida y aconsejarme cuando no veo las cosas claras.

A Javier Martínez Portellano porque, aparte de ser un gran compañero que me ha ayudado con todo innumerables veces, es un gran amigo que siempre está ahí para cualquier cosa.

Y por último, a Francisco Charte Ojeda, que ha sido mi profesor modelo desde que lo tuve la primera vez. Él ha sabido hacer que me apasione todo lo que enseña sin importar el tema. He aprendido muchísimo de él y siempre llevaré su recuerdo conmigo.

Tabla de contenidos

Lista de figuras

Lista de tablas

Capítulo 1

INTRODUCCIÓN

En este capítulo se realiza un breve resumen del contenido incluido en este trabajo, así como la motivación para realizarlo y, finalmente, un esquema con la distribución de los capítulos siguientes.

1.1. Resumen

Este trabajo de fin de grado trata de buscar un método para extraer acordes de piezas musicales de manera automática. La extracción de dichos acordes ha sido siempre un trabajo manual, que necesita un conocimiento previo sobre armonía musical. En este trabajo, se pretende realizar esta extracción, en primer lugar, con modelos de aprendizaje automático más básicos como Random Forest, kNN o regresión logística y, posteriormente, con técnicas de aprendizaje profundo como el perceptrón multicapa o redes neuronales convolucionales.

En esta memoria se describen la búsqueda y obtención de los conjuntos de datos que van a utilizarse, la realización de un análisis exploratorio de los datos descargados, el preprocesamiento de los datos realizado, un análisis exploratorio a los conjuntos de datos finales, el diseño de los diferentes modelos, su entrenamiento, y un análisis de los resultados obtenidos. Todo este trabajo se basa en una serie de conceptos, de teoría musical y aprendizaje automático, explicados en un capítulo de antecedentes, donde también se realiza un estudio del estado del arte de la extracción de acordes de manera automática.

El trabajo realizado ha llevado al diseño de un modelo de red neuronal convolucional que trabaja con una representación del audio conocida como *mel-filterbank*. Este

modelo es capaz de extraer 42 tipos de acordes distintos de una pieza musical con un 80 % de exactitud.

1.2. Motivación

La motivación principal que me ha llevado personalmente a realizar este trabajo es mi afición musical que tengo desde pequeño. Desde que entré en el conservatorio, me enseñaron a amar la música tanto como a comprenderla. La armonía ha jugado siempre un papel muy importante en la música para mí y he visto una muy buena oportunidad para juntar mis dos pasiones, la música y la informática.

Otro motivo para realizar este trabajo es que la extracción de acordes es una tarea que conlleva tiempo, y lograr su obtención automática sería bastante útil. La actual falta de modelos de obtención de acordes de forma precisa se debe a la complejidad de esta tarea.

Tradicionalmente, la obtención de los acordes se realiza analizando la partitura de una obra, nota por nota, y actualmente casi ninguna pieza musical de carácter popular cuenta con un partitura con la que podamos analizar los acordes; por ello, esta extracción manual de acordes se complica más.

1.3. Estructura de la memoria

La distribución de capítulos se realizará de la siguiente forma:

Capítulo 1. Introducción

Este capítulo hace un resumen del trabajo completo, expone la motivación por la que se ha realizado el trabajo y describe la estructura de la memoria.

Capítulo 2. Antecedentes

Se explican todos los conocimientos previos que hay que tener para comprender el contenido del presente trabajo. Se compone de una parte de teoría musical y de otra

de aprendizaje automático.

Capítulo 3. Objetivos

Se describen el objetivo principal y los objetivos específicos que engloba este trabajo.

Capítulo 4. Materiales y métodos

Se explica de forma exhaustiva todo el proceso de realización del trabajo. Se compone de la búsqueda y obtención del conjunto de datos, el preprocesamiento de los datos, el análisis exploratorio, el desarrollo de los clasificadores básicos y de las redes neuronales, el método seguido para evaluar los modelos y las herramientas utilizadas en todo el desarrollo del trabajo.

Capítulo 5. Resultados

Aquí se presentan los resultados obtenidos de cada modelo, se evalúan y se realiza una comparación entre ellos.

Capítulo 6. Conclusiones

En este capítulo se hace un resumen de todo el proceso realizado y se muestran las conclusiones finales.

Capítulo 2

ANTECEDENTES

Para entender el contexto de este trabajo es necesario comprender conceptos que pertenecen a dos campos distintos. Por una parte, la teoría musical y, por otra, el aprendizaje automático. En este capítulo se presentarán todos los conceptos necesarios para entender correctamente este trabajo. En la primera sección se expondrán los fundamentos musicales, en la segunda se facilita una introducción al aprendizaje automático y el capítulo terminará con una revisión del estado del arte.

2.1. Teoría musical

Antes de empezar, hay que señalar que se van a simplificar algunos conceptos ya que no es necesario conocer todos los detalles para explicar lo necesario para este trabajo.

La teoría musical es un campo de estudio muy amplio. Dentro de él se pueden distinguir diversos elementos, entre los cuales se encuentra el análisis de una pieza musical. Una pieza musical se puede analizar desde cuatro perspectivas diferentes, como se explica más exhaustivamente en Piston and DeVoto [1994]. Existen los análisis rítmico, melódico, armónico y formal. El análisis rítmico se centra en lo que respecta al ritmo de una obra, el melódico a la melodía, el armónico a la armonía y el formal a la forma o estructura que presenta, para describir las partes que componen una obra y sus relaciones entre ellas. En el presente trabajo nos interesa presentar el análisis armónico.

2.1.1. Discretización del sonido

Como introducción, es importante definir lo que es una nota musical. El umbral de audición de un ser humano medio comprende desde los 20 Hz a los 20 kHz. Para poder identificar de forma más sencilla cada uno de los sonidos intermedios, se discretizó este umbral creando las notas musicales. Una nota es la abstracción de una onda sonora senoidal con una frecuencia concreta.

Para conocer la frecuencia a la que corresponde una nota necesitamos conocer también la escala en la que se encuentra. Una misma nota como la tiene 440 Hz en la escala 4, 880 en la escala 5 o 3520 en la escala 7, por ejemplo. Si vemos los Hz a los que corresponde, la escala inmediatamente superior duplica los Hz de cada una de sus notas.

En la música occidental existen doce notas diferentes, siete de las cuales, por razones históricas, han obtenido un nombre; las cinco restantes se llaman respecto a la nota más cercana de las siete anteriores. Las notas con nombre propio son do, re, mi, fa, sol, la y si, pero, por ejemplo, do # (sostenido) se refiere a la nota por encima del do, también llamada re b (bemol), nota por debajo del re. El orden completo sería el representado en la figura 2.1.

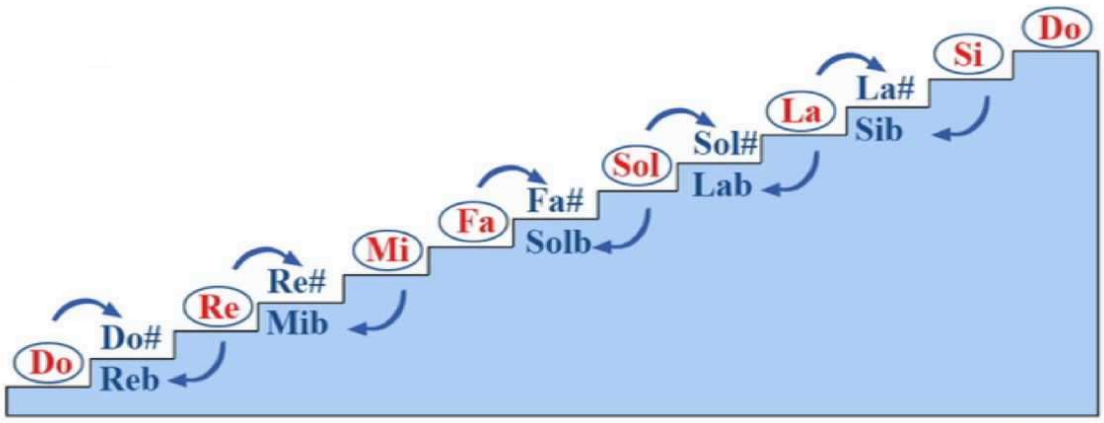

Figura 2.1: Representación de una escala cromática [YouTube, 2013]

Cabe destacar que la distancia entre todas ellas es la misma, de un semitono. Un tono equivale a dos semitonos.

Una vez conocidas las notas musicales podemos hablar de armonía.

2.1.2. Armonía

La armonía es el estudio de notas musicales combinadas simultáneamente, también llamadas acordes. La música occidental generalmente se ha formado sobre la tonalidad o sistema tonal, que es una organización jerárquica de las notas a partir de otra llamada centro tonal o tónica, junto a una serie de acordes y escalas. Esta jerarquía puede representar las diferentes funciones tonales de dicha tonalidad con acordes. En Schönberg and Barce [2004] se recoge mucha más información al respecto.

Acordes

Uno de las palabras clave sobre las que se forma este TFG es el término de acorde. Un acorde es un conjunto de tres o más notas que constituyen una unidad armónica. Los acordes toman distintos nombres dependiendo de las notas con las que se formen. Pueden ser mayores, menores, aumentados, disminuidos, con 7ª menor, con 7ª mayor, semidisminuidos, con 9ª mayor, 9ª menor, con 6ª y otros muchos.

Un acorde se puede representar en diversos cifrados; entre ellos, el cifrado funcional y el cifrado por grados. Este último expresa en números romanos el acorde respecto a la tónica. En este trabajo se van a tratar los acordes con el cifrado americano. Este representa las notas musicales con letras:

Figura 2.2: Representación de las notas en el cifrado americano

Al escuchar un solo acorde no se puede saber su tonalidad. Esto se debe a que un acorde puede pertenecer a varias tonalidades y ejercer funciones tonales diferentes en cada una. Para poder conocer la tonalidad de una pieza musical, se necesita una sucesión de acordes que la identifique.

Una tonalidad también se representa por el nombre de una nota y, a su vez, puede

ACORDE	CIFRADO				
do mayor	C				
do menor	Cm			C-	
do aumentado	Caug			C+	
do disminuido	Cdim				
do con cuarta suspendida	Csus			Csus4	
do con segunda suspendida	Csus2				
do con segunda añadida	Cadd2				
do sexta	C6				
do menor sexta	Cm6			C-6	
do sexta/novena	C6/9				
do menor sexta novena	Cm6/9			C-6/9	
do séptima mayor	Cmaj7		C$\triangle$	CM7	C$\triangle$7
do séptima	C7				
do menor séptima	Cm7			C-7	
do semidisminuido o do menor séptima con quinta disminuida	C^o	C^{o}7	Cm7(b5)	C-7(b5)	Cm7(-5)
do disminuido de séptima	Cdim7				
do séptima con cuarta suspendida	C7sus			C7sus4	
do séptima mayor con quinta aumentada	CM7(#5)	CM7(+5)		Cmaj7(+5)	C$\triangle$7(#5)
do menor con séptima mayor	Cm(M7)		Cm($\triangle$7)	C-(M7)	Cm(maj7)

Tabla. 2.1: Cifrado de varios tipos de acordes

ser mayor o menor. También puede haber tanto acordes como tonalidad sobre una nota con sostenido o con bemol. Un ejemplo sería si bemol mayor o BbM en cifrado americano.

Análisis armónico

Al analizar una pieza musical armónicamente debemos obtener los acordes de los que se compone la pieza.

Desde hace muchos años, la obtención de acordes se realiza para saber la función tonal que ejerce cada uno de ellos. Para ello, se observa su partitura, se halla la tonalidad y se estudia nota por nota la composición para formar los diferentes acordes que se expresan en grados con números romanos. Esto significa que en la tonalidad de do mayor el acorde del mismo nombre tiene una función tonal de primer grado o tónica (I) y el acorde de sol mayor realiza la función de quinto grado o dominante

(V). Al extraer la función tonal, se pueden observar los patrones armónicos que se encuentran en las obras, por ejemplo I - IV - V. Estos patrones son los mismos si cambias la tonalidad de la obra. En cambio, los nombres de los acordes como do mayor sí serían diferentes al cambiar la tonalidad. En este caso, no interesa conocer como tal las funciones tonales de las obras, sino los acordes en sí. Para conocer más acerca del tema se recomienda consultar Zamacois [1997].

En la figura 2.3 podemos visualizar una tabla donde podemos ver las tonalidades mayores en rojo al inicio de cada fila. Al inicio de cada columna encontramos algunas de las funciones tonales que pueden ejercer los acordes. Y dentro de la tabla encontramos los acordes que ejercen dichas funciones tonales en cada tonalidad. Por ejemplo el acorde C ejerce el primer grado en la tonalidad de do mayor y en la tonalidad de fa mayor ejerce un quinto grado.

	I	IIm	IIIm	IV	V	VIm	VII^o
C	C	Dm	Em	F	G	Am	B^o
Db	Db	Ebm	Fm	Gb	Ab	Bbm	C^o
D	D	Em	F#m	G	A	Bm	C#^{o}
Eb	Eb	Fm	Gm	Ab	Bb	Cm	D^o
E	E	F#m	G#m	A	B	C#m	D#^{o}
F	F	Gm	Am	Bb	C	Dm	E^o
F#	F#	G#m	A#m	B	C#	D#m	E#^{o}
Gb	Gb	Abm	Bbm	Cb	Db	Ebm	F^o
G	G	Am	Bm	C	D	Em	F#^{o}
Ab	Ab	Bbm	Cm	Db	Eb	Fm	G^o
A	A	Bm	C#m	D	E	F#m	G#^{o}
Bb	Bb	Cm	Dm	Eb	F	Gm	A^o
B	B	C#m	D#m	E	F#	G#m	A#^{o}

Figura 2.3: Funciones tonales de los acordes en diferentes tonalidades [Clases de guitarra Lima, 2021]

Un problema que dificulta la obtención de acordes de forma automática es el hecho de que una melodía por sí sola no implica una armonía determinada. Esta melodía puede ser armonizada de formas distintas. Una misma pieza musical puede sonar completamente diferente tan solo cogiendo su melodía y ajustándola a otra tonalidad

y serie de acordes.

2.2. Aprendizaje automático

En esta sección se explicarán todas las nociones básicas sobre el aprendizaje automático para comprender este trabajo. En primer lugar, se describirá dónde se sitúan todas las técnicas que se van a utilizar dentro del campo de conocimiento de la inteligencia artificial. Posteriormente, se explicará el funcionamiento de los clasificadores básicos y de una red neuronal. Por último, se describirá el estado del arte de la obtención de acordes con aprendizaje automático.

2.2.1. Campo de conocimiento

La inteligencia artificial es un campo de conocimiento amplio que abarca distintas disciplinas, entre los que se encuentran los sistemas basados en reglas, el procesamiento del lenguaje natural y el aprendizaje automático. Es complejo dar una única definición a la inteligencia artificial, ya que, a lo largo de la historia, su definición ha variado. Andreas Kaplan y Michael Haenlein en Kaplan and Haenlein [2019] la definen como «la capacidad de un sistema para interpretar correctamente datos externos, para aprender de dichos datos y emplear esos conocimientos para lograr tareas y metas concretas a través de la adaptación flexible».

En este trabajo se abarcan técnicas de inteligencia artificial más específicas del campo del aprendizaje automático. La definición de aprendizaje automático según Russell [1995] es el subcampo de la IA cuyo objetivo es desarrollar técnicas que permitan que las computadoras aprendan o mejoren su desempeño con la experiencia. Dentro del subcampo del aprendizaje automático, se pueden distinguir las técnicas utilizadas según su finalidad: descriptiva o predictiva, y a través de modelos que pueden ser supervisados o no supervisados.

Tal y como se explica en Vico et al. [2018], los modelos descriptivos tienen como objetivo encontrar y definir relaciones interesantes en los datos utilizando para ello aprendizaje no supervisado. En cambio los predictivos, tienen como objetivo la predicción del valor de una variable de interés en nuevas instancias no vistas anteriormente, utilizando para ello aprendizaje supervisado. Como aprendizaje supervisado entendemos una técnica para deducir una función a partir de unos datos de entrenamiento. Por otra parte, el aprendizaje no supervisado no dispone de conocimiento a priori.

Uno de los campos donde más se trabaja es el del aprendizaje supervisado de modelos predictivos, donde caben las técnicas de regresión y las de clasificación. Dependiendo del tipo de salida obtenida existe la clasificación binaria, multiclase, multietiqueta y otras variantes. En el presente trabajo se utilizan modelos de clasificación multiclase, más concretamente clasificadores básicos, como Random Forest, k vecinos más cercanos y regresión logística. También se utiliza modelos de descenso de gradiente como el perceptrón multicapa y la red neuronal de convolución.

A continuación, se describirán todas las técnicas mencionadas y su funcionamiento.

2.2.2. Clasificadores básicos

Un modelo de clasificación es un algoritmo entrenado con ciertos datos que es capaz de distinguir entre varias clases definidas previamente. Este modelo sigue un algoritmo en particular para aprender a clasificar y puede tener un rendimiento bueno o malo. Cada tipo de modelo de clasificación puede obtener diferente rendimiento al solucionar un problema. Según los datos que se tratan o el número de clases que se separan pueden existir varios modelos con rendimiento similar. En este apartado describiremos tres modelos de clasificación sencillos. En primer lugar, se explicará el funcionamiento de Random Forest; posteriormente, el de k vecinos más cercanos y, por último, el de regresión logística.

Random Forest

Un modelo Random Forest [Breiman, 2001] está compuesto por múltiples árboles de decisión. Los árboles de decisión [Quinlan, 1996] son algoritmos que buscan reglas lógicas para separar distintas clases. El resultado del clasificador será el resultado obtenido por la mayoría de árboles de decisión. Este método es interesante porque los datos con los que entrena a los diferentes árboles no son los mismos. A cada árbol no le proporciona el total de los datos. Esto hace que el resultado sea mejor que el de un solo árbol de decisión.

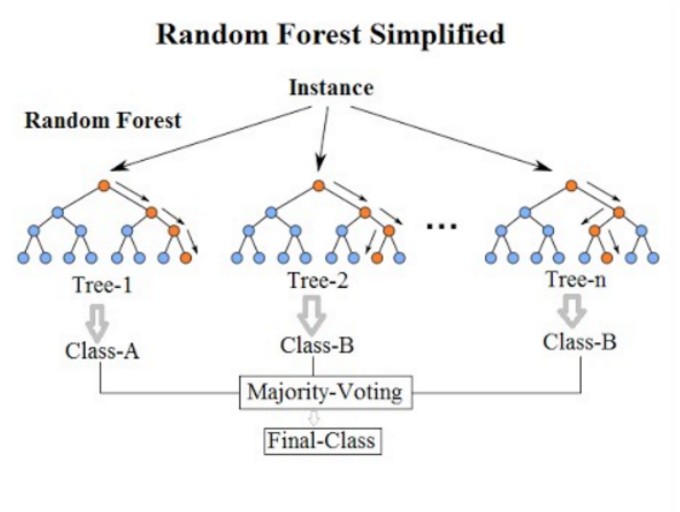

Figura 2.4: Diagrama del Random Forest [contributors, 2021]

kNN

Un modelo kNN calcula la probabilidad de que un elemento pertenezca a una clase determinada. Este modelo se dice que sigue un tipo de aprendizaje perezoso (*lazy learning*) debido a que no genera un modelo fruto del entrenamiento previo, sino que aprende conforme prueba los datos de test. Este algoritmo genera un entorno local de un número fijo de vecinos (k) también llamado vecindario y, en torno a este vecindario, clasifica el elemento nuevo calculando las distancias con sus vecinos. k es un hiperparámetro configurable, de forma que se tomará un número mayor o menor de vecinos a partir de los cuales se hará la predicción. Según cambia k se tiene un modelo que tiene mayor o menor variabilidad, al considerar desde solo el vecino más cercano hasta una vecindad más o menos grande.

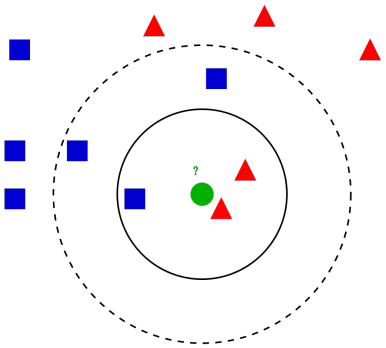

Figura 2.5: Ejemplo del algoritmo de kNN [Wikipedia, 2020]

Regresión logística

La regresión logística es un modelo estadístico de clasificación que funciona de manera similar a la regresión lineal, ya que explica datos con una función lineal. La diferencia entre la regresión lineal y logística es que la variable de salida de la lineal es cuantitativa y la de la logística cualitativa. Este modelo de clasificación no es capaz de clasificar correctamente un problema que tenga cierta complejidad.

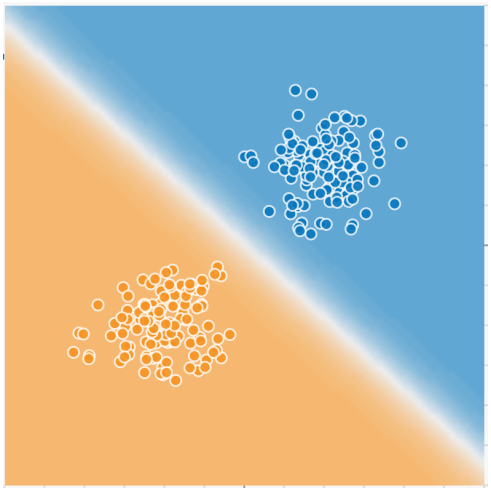

Figura 2.6: Clasificación satisfactoria con regresión logística

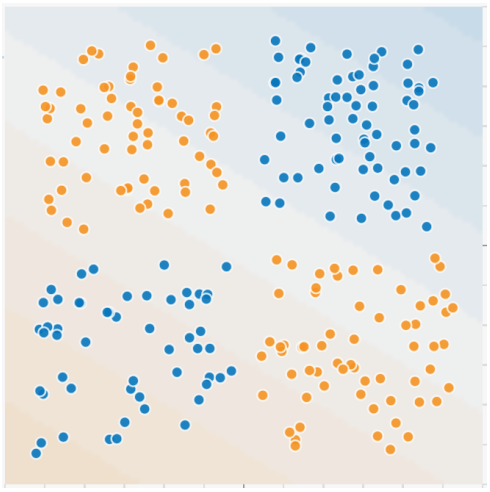

Figura 2.7: Clasificación no satisfactoria con regresión logística

2.2.3. Redes neuronales artificiales

Las redes neuronales [Schmidhuber, 2015] se han convertido en la familia de algoritmos de aprendizaje automático más utilizada en los últimos años debido a la mejora de la técnica y la tecnología. Estas imitan el funcionamiento de una red neuronal biológica. Para comprender el funcionamiento de una red neuronal artificial, es importante conocer de antemano el término de neurona.

Concepto de neurona artificial

Una neurona de una red neuronal artificial es la unidad básica de procesamiento dentro de una red neuronal. Una neurona dispone de unos valores de entrada a partir de los cuales realiza un cálculo interno y obtiene un valor de salida. Funciona como una función matemática que realiza una suma ponderada de dichos valores de entra-

da. La ponderación asignada a cada valor viene dada por el peso de cada valor de entrada. A la función matemática también se suma un término independiente llamado sesgo (*bias* en inglés). Los pesos y el sesgo funcionan como parámetros del modelo que se va a entrenar. Una única neurona permite solucionar problemas de clasificación con regresión lineal, que funciona como la regresión logística, pero, si es un problema más complejo, se necesita usar más neuronas conectadas.

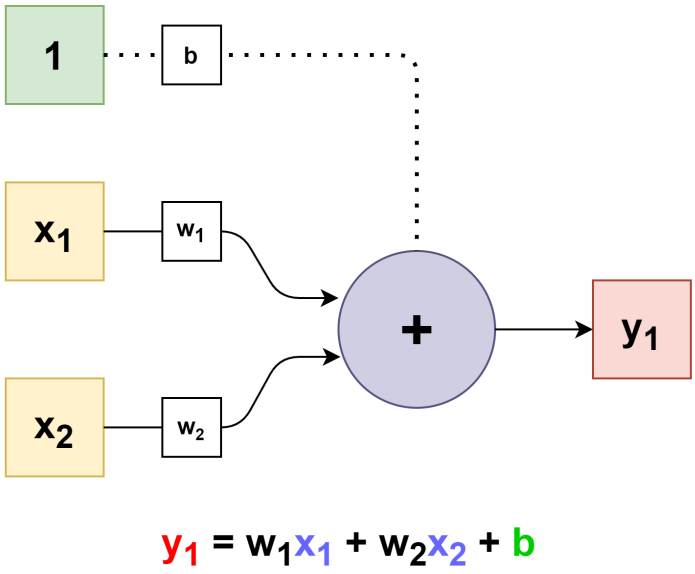

Figura 2.8: Diagrama representativo de una neurona artificial

Red de una red neuronal

Las neuronas se pueden organizar vertical u horizontalmente. Todas las neuronas organizadas verticalmente forman una capa. Una red neuronal se dice que es más profunda cuando dispone de un mayor número de capas. A la primera capa de una red neuronal se le llama capa de entrada y, a la última, capa de salida. En la figura 2.9 podemos ver su arquitectura. El resto se denominan capas ocultas. Cuando conectamos varias capas de neuronas, las salidas de la primera capa pasan a la segunda y esto permite extraer conocimiento de los datos; puede extraer características. Al añadir múltiples capas, se distingue este modelo de los clasificadores básicos en esa abstracción de los datos y, por tanto, su campo dentro del aprendizaje automático se denomina aprendizaje profundo (*deep learning* en inglés).

Cada neurona tiene un componente adicional llamado función de activación, que introduce en la red neuronal una componente no lineal. Si no se utilizase una función de activación todas las neuronas actuarían siguiendo un comportamiento lineal que no permitiría resolver problemas complejos. Esta función altera el resultado obtenido

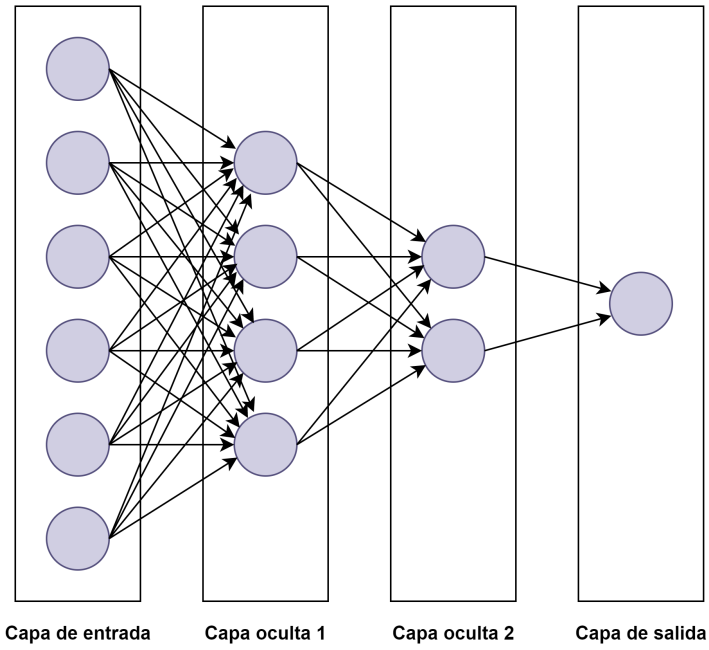

Figura 2.9: Diagrama de las capas de una red neuronal sencilla

por cada neurona para forzar un resultado entre cero y uno. Dependiendo del tipo de la función de activación, le será más fácil aprender a la red, teniendo en cuenta su objetivo.

Backpropagation

Uno de los algoritmos clásicos de optimización de funciones es el descenso del gradiente [Robbins, 2007]. Este algoritmo calcula el vector gradiente de una función coste para ver la dirección e intensidad de la pendiente de la misma función y así poder minimizar el coste. Aplicar este algoritmo a una red neuronal para producir el aprendizaje no es tan sencillo debido a la cantidad de conexiones entre neuronas. Para calcular cómo afecta el coste cuando modificamos algunos de los pesos de la red, necesitamos al algoritmo de *backpropagation*. Este algoritmo nos ahorra tener que calcular por fuerza bruta los cambios en el coste final de la red cuando cambiamos una única neurona. El *backpropagation* distribuye el error obtenido en la salida de la red a las neuronas que han afectado más para obtener dicho error.

Una vez explicado el funcionamiento más general de una red neuronal, se va a proceder a explicar con más detalle las funciones de activación más utilizadas y los diferentes tipos de capas usadas en este trabajo.

Funciones de activación

Como se ha explicado anteriormente, existen múltiples tipos de funciones de activación, entre ellas:

- **Función escalonada**: esta función produce una salida binaria de 0 o 1. No es muy recomendable utilizar esta función ya que no favorece al aprendizaje.

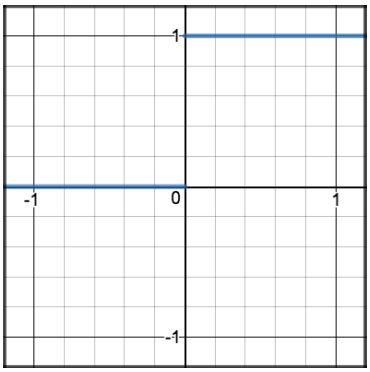

Figura 2.10: Función escalonada [Machine Learning Glossary, 2021]

- **Función sigmoide**: esta función produce una salida entre 0 y 1 donde los valores más bajos son muy cercanos al 0 y los más altos al 1. Esta función se usa mucho en la última capa de las redes neuronales para realizar clasificaciones binarias.

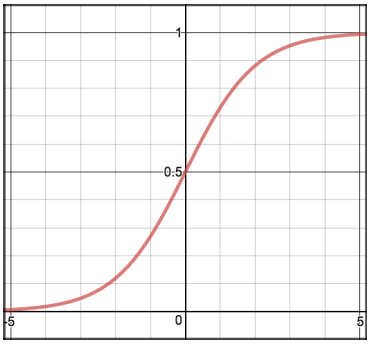

Figura 2.11: Función sigmoide [Machine Learning Glossary, 2021]

- **Función tanh**: esta función produce una salida en el intervalo [-1, 1] en lugar de [0, 1]. Es tan popular como la sigmoide, pero, a diferencia de ella, su salida está centrada en el cero. Por lo tanto, en la práctica siempre se prefiere la no linealidad tanh a la no linealidad sigmoide.

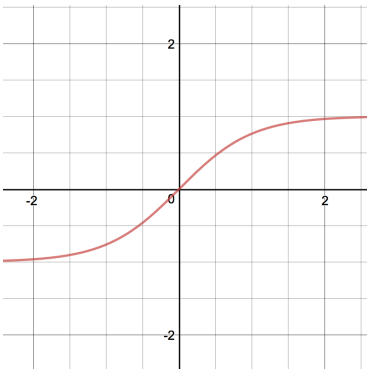

Figura 2.12: Función tanh [Machine Learning Glossary, 2021]

- **Función ReLU**: la función ReLU o *Rectified Linear Unit* es la función de activación más utilizada actualmente. Existen evidencias en LeCun et al. [2012] de que obtiene resultados con menor error que los generados con función sigmoide (que está inspirada en la teoría de la probabilidad) y también de que es más práctica.

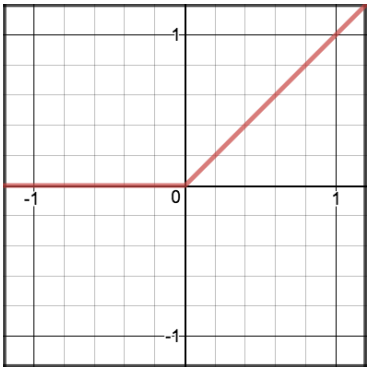

Figura 2.13: Función ReLU [Machine Learning Glossary, 2021]

- **Función softmax**: esta función se utiliza mucho en la última capa para obtener una clasificación entre múltiples clases. Calcula la distribución de probabilidad teniendo en cuenta el número de clases a distinguir.

Perceptrón multicapa

El perceptrón multicapa o MLP (*Multi Layer Perceptron*) [Haykin, 1999] es un tipo de red neuronal artificial compuesta de varias capas que se encuentran totalmente conectadas. Este tipo de red neuronal es una de las redes mas básicas que hay. Aun así, esta red es capaz de extraer características y de dar, en general, un rendimiento mayor al de los clasificadores básicos como Random Forest o kNN.

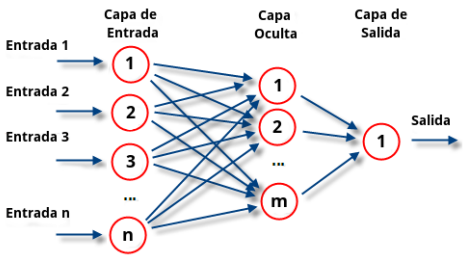

Figura 2.14: Esquema de la arquitectura de un perceptrón multicapa [Wikipedia, 2021]

Como podemos observar en la figura 2.14, el MLP posee n entradas de datos que entran cada una en una de las n neuronas de la capa de entrada. La neurona calcula el dato de salida multiplicando el peso de la variable de entrada con el valor de dicha variable más el sesgo. Este resultado se ve afectado utilizando una función de activación y pasará a todas las neuronas de la siguiente capa como una variable de entrada. Ese proceso, denominado *feedforward*, se repite a lo largo de toda la red neuronal y finalmente se obtiene una salida. Posteriormente se calcula el error obtenido comparando la salida predicha con el valor real correspondiente y se reparte el error hacia atrás con el proceso de *backpropagation*. Con este error se modifican los valores de los pesos y se hace otra iteración. Esto se repite muchas veces y se terminaría el entrenamiento.

Red neuronal convolucional

Las redes neuronales convolucionales son redes neuronales artificiales basadas en el perceptrón multicapa pensadas para tratar matrices bidimensionales, por lo que son muy efectivas para tareas de clasificación o reconocimiento de imágenes. Este tipo de redes neuronales también están inspiradas en un proceso biológico. Según Fukushima [2007], el patrón de conectividad entre las neuronas se asemeja a la organización del córtex visual animal.

Las redes neuronales convolucionales, a pesar de estar pensadas para procesar imágenes o matrices bidimensionales, pueden procesar un vector de datos unidimensional. Este tipo de redes pueden tener múltiples usos, como el reconocimiento automático del habla, monitorización a tiempo real de electrocardiograma o detección del daño de infraestructuras basado en vibración, entre otros, según se expone en Kiranyaz et al. [2021].

Una práctica muy utilizada para la construcción de redes neuronales convolucionales es el *transfer learning* [Yang et al., 2020]. Esta técnica permite construir una

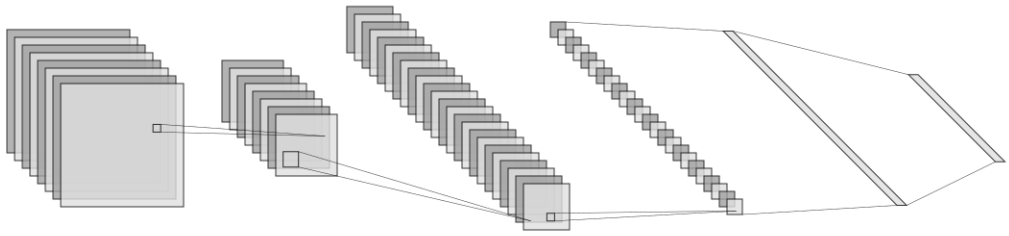

Figura 2.15: Esquema de una arquitectura de una red neuronal convolucional de ejemplo

red neuronal utilizando otras arquitecturas de redes neuronales, en este caso convolucionales, conocidas como Inception [Pandit et al., 2020], ResNet50 [He et al., 2016] y muchas más ya entrenadas con un conjunto de datos conocido, como Imagenet [Deng et al., 2009]. A esta red preentrenada se la suele volver a entrenar conservando los pesos del modelo excepto los de las últimas capas para ajustarse a los nuevos datos.

Tipos de capas presentes en redes convolucionales

Gracias a trabajar con bibliotecas especializadas en redes neuronales se permite abstraer toda la programación de la red neuronal a un alto nivel y trabajar directamente con capas ya preconfiguradas. Estas capas pueden ser de muchos tipos y nuestra labor es la de formar una arquitectura propia de la red juntando varios tipos de capas.

Los tipos de capas utilizados en este trabajo se explican a continuación según la documentación de Keras:

- **Input**: esta capa es la de entrada y, en ella, se especifican las dimensiones de la arquitectura de la red.
- **Dense**: esta capa se caracteriza por ser totalmente conectada o *fully connected*. Se le debe asignar el número de neuronas que se van a incluir en el modelo.
- **Conv1D**: este filtro se utiliza con entradas unidimensionales y crea un *kernel* de convolución que se convoluciona con la entrada de la capa para producir un tensor de salidas.

- **Conv2D**: este filtro se utiliza con entradas bidimensionales y crea un *kernel* de convolución que se convoluciona con la entrada de la capa para producir un tensor de salidas.
- **MaxPooling1D**: este filtro se utiliza con entradas unidimensionales y disminuye la muestra de la representación de entrada tomando el valor máximo en una ventana espacial de tamaño asignado en `pool_size`.
- **MaxPooling2D**: este filtro se utiliza con entradas bidimensionales y reduce la muestra de la entrada a lo largo de sus dimensiones espaciales (altura y anchura) tomando el valor máximo sobre una ventana de entrada (de tamaño definido por `pool_size`) para cada canal de la entrada.
- **BatchNormalization**: durante el entrenamiento, la capa normaliza su salida utilizando la media y la desviación estándar del *batch* actual de entradas.
- **Flatten**: si la entrada es de múltiples dimensiones, esta capa aplana la entrada en una dimensión.
- **Dropout**: la capa Dropout establece aleatoriamente las unidades de entrada a 0 con una frecuencia de tasa en cada paso durante el tiempo de entrenamiento, lo que ayuda a prevenir el sobreajuste de la red.

Red neuronal autoconfigurable

Existe una serie de herramientas que, mediante distintas técnicas, buscan la mejor arquitectura para que una red neuronal artificial alcance el máximo rendimiento con unos datos datos. Se conoce genéricamente como NAS (*Network Arquitecture Search*) a dicha técnica y, para ello, pueden usarse técnicas evolutivas, de búsqueda en cuadrícula. Una de las bibliotecas más utilizadas de este tipo de red para Python es AutoKeras [Jin et al., 2019]. Las herramientas de AutoML se aplican principalmente a redes neuronales convolucionales pero también pueden usarse con otro tipo de redes como se realiza en Charte et al. [2020].

Red neuronal recurrente

Las redes neuronales recurrentes o RNN [Sherstinsky, 2020] son un tipo de red neuronal artificial que permite conectar neuronas de una capa n con otras capas menores, lo que permite crear ciclos y así incluir una variable temporal para que la red

tenga memoria. Gracias a su arquitectura pueden procesar datos con longitud variable. Estas redes dan buenos resultados con el análisis de secuencias, ya sea de texto, sonido o vídeo.

La memoria mencionada anteriormente es estática, ya que mantiene el conocimiento aprendido hasta después del entrenamiento. Esta memoria puede ser útil para recordar estados previos y utilizar esta información para decidir cuál será el siguiente.

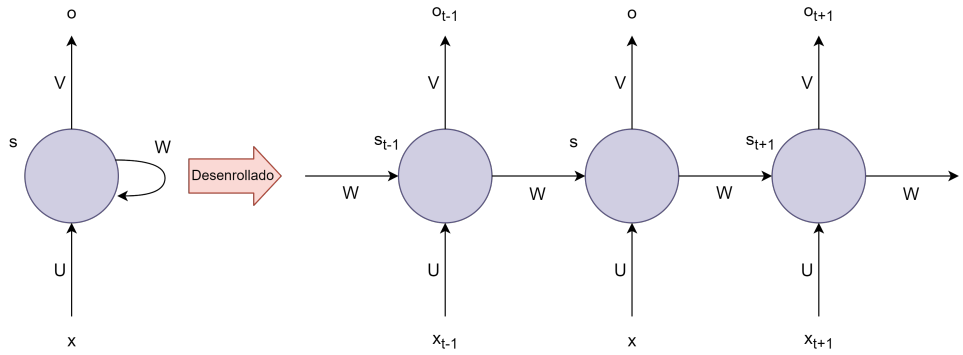

Figura 2.16: Diagrama de una red neuronal recurrente

2.3. Estado del arte

En esta sección se van a presentar algunas de las diferentes técnicas y modelos presentes en la literatura que se han utilizado anteriormente para resolver el problema de extracción de acordes mediante aprendizaje profundo.

Tradicionalmente, la extracción y anotación de los acordes de una pieza musical ha sido y sigue siendo un trabajo completamente manual. Actualmente no existe un método automatizado del todo fiable. Se ha probado con muchos modelos de detección automática, pero ninguno podría considerarse perfecto. La mayoría de las veces son métodos que no utilizan aprendizaje automático, como en Fujishima [1999], que utilizó una técnica de coincidencia de patrones y una heurística para reconocer acordes. Algunos de los métodos funcionales más populares son la página web Chordify o la aplicación móvil de Capo para dispositivos iOS.

En la web de Chordify se menciona:

«*It is close to impossible to observe any meaningful information about chords or beats in a song's waveform directly. Therefore, we convert the audio into a representation that reveals some insights into its musical content: a spectrogram.*»

«Es casi imposible observar directamente cualquier información significativa sobre los acordes o los tiempos en las muestras de audio de una canción. Por ello, convertimos el audio en una representación que revela algunos datos sobre su contenido musical: un espectrograma.»

La mayoría de métodos utilizados previamente trabajan con representaciones del audio como el espectrograma para así extraer los acordes y tratan con un conjunto de acordes bastante reducido. En Osmalskyj et al. [2012] se describe cómo trabajan solo con los 10 acordes más conocidos para conseguir un mayor rendimiento.

«*Since our final goal is not to recognize all the existing chords, but to develop a music recognition system, we can limit chords to the most frequent ones. Therefore, we chose a subset of 10 chords.*»

«Como nuestro objetivo final no es reconocer todos los acordes existentes, sino desarrollar un sistema de reconocimiento musical, podemos limitar los acordes a los más frecuentes. Por lo tanto, elegimos un subconjunto de 10 acordes.»

Este mismo artículo refleja la utilización de un descriptor llamado PCP o *Pitch Class Profile* para identificar los acordes. El PCP es un vector que recoge los tonos de los que se compone un acorde. Para obtener el PCP de un audio se deben de seguir los siguientes pasos:

1. Introducir la señal musical.
2. Realizar un análisis espectral para obtener los componentes de frecuencia del audio.
3. Utilizar la transformada de Fourier para convertir la señal en un espectrograma.
4. Hacer un filtrado de frecuencias. Se utiliza un rango de frecuencias de entre 100 y 5000 Hz.
5. Detectar los picos. Solo se consideran los valores máximos locales del espectro.
6. Realizar el procedimiento de cálculo de la frecuencia de referencia. Estimar la desviación con respecto a 440 Hz.
7. Hacer el *Pitch class mapping* con respecto a la frecuencia de referencia estimada. Se trata de un procedimiento para determinar el valor de la clase del tono a partir de los valores de frecuencia. Se utiliza un esquema de ponderación con la función coseno. Se considera la presencia de frecuencias armónicas teniendo en cuenta un total de 8 armónicos para cada frecuencia. Para asignar el valor en

un tercio de semitono, el tamaño de los vectores de distribución de la clase de tono debe ser igual a 36.

8. Se normaliza la característica *frame* a *frame* dividiendo entre el valor máximo para eliminar la dependencia de la sonoridad global. Entonces, podemos obtener una secuencia PCP.

Todos estos pasos son necesarios para analizar cada pieza musical.

Otro artículo con un buen resultado es Boulanger-Lewandowski et al. [2013]. Aquí se utiliza una red neuronal recurrente para solucionar el problema. Con este tipo de red, más orientada al análisis de secuencias, se consigue un alto rendimiento al analizar el espectrograma extraído.

Como se ha podido observar, esta tarea tiene una gran complejidad y requiere de otro tipo de representación del audio para lograr extraer los acordes.

2.3.1. LEAF

Durante la realización de este trabajo de fin de grado, el 12 de marzo de 2021, Google publica en su blog de inteligencia artificial un artículo llamado *LEAF: A Learnable Frontend for Audio Classification* [Zeghidour et al., 2021]. Este artículo trata de una herramienta para Python basada en aprendizaje profundo que han desarrollado para generar *mel-filterbanks* muy parecidos e incluso mejores que los hechos a mano. Un *mel-filterbank* es una imagen que representa el espectro de frecuencias de un audio de una forma muy parecida a como lo escuchamos los humanos. Los humanos discriminamos mejor las frecuencias bajas y peor las altas. Los *mel-filterbanks* dan una mayor resolución a las frecuencias bajas y menor a las altas tal y como se explica en Haytham Fayek [2016]. Estos filtros han sido muy utilizados sobre todo para el reconocimiento del habla, pero como describen en el artículo:

«In this work we show that we can train a single learnable frontend that outperforms mel-filterbanks on a wide range of audio signals, including speech, music, audio events and animal sounds, providing a general-purpose learned frontend for audio classification.»

«En este trabajo se muestra que podemos entrenar un único *frontend* de aprendizaje que consigue mejores resultados que los *mel-filterbanks* en una extensa gama de señales de audio, que incluyen el habla, la música, los eventos sonoros y los soni-

dos de animales, ya que proporciona un *frontend* de aprendizaje de aplicación general para la clasificación de audio.»

Hasta el momento no existe ningún artículo publicado sobre la extracción de acordes utilizando LEAF para representar el audio.

Capítulo 3

OBJETIVOS

Este trabajo busca cumplir varios objetivos a lo largo de la realización del mismo. Entre ellos se encuentra un objetivo principal y una serie de objetivos específicos que alcanzar.

3.1. Objetivo principal

El objetivo principal que se quiere conseguir en este trabajo es la extracción de acordes de una pieza musical con técnicas de aprendizaje profundo. Para ello, se debe crear un modelo de red neuronal profunda entrenado con un conjunto de datos de piezas musicales con sus respectivos acordes anotados y que obtenga el mejor rendimiento posible al extraer dichos acordes.

3.2. Objetivos específicos

Como objetivos específicos se pueden distinguir los siguientes:

Acercamiento a la ciencia de datos

Como primer objetivo específico podemos considerar la toma de contacto con técnicas actuales de ciencia de datos y aprendizaje automático. Esto implica aprender cómo funcionan diversas técnicas de aprendizaje automático, específicamente las re-

des neuronales artificiales. Por otro lado, conocer los distintos tipos de redes existentes y utilizar alguna arquitectura más específica para solucionar problemas y, por último, aprender a utilizar las bibliotecas específicas de *machine learning* como pueden ser Tensorflow o Keras o cualquier otra equivalente.

Obtención y adaptación de un conjunto de datos

Otro objetivo es la obtención y adaptación de un conjunto de datos de piezas musicales para entrenar un modelo de red neuronal.

Además, conocer el estado del arte de la extracción de acordes de piezas musicales con aprendizaje automático es importante. Con este, se puede aprender qué técnicas se han realizado anteriormente y saber el tipo de datos y el procesamiento de los mismos que se necesitan para formar un conjunto de datos que permitan entrenar al modelo de forma óptima.

Por otro lado, es importante también la búsqueda de un conjunto de datos ya existente que cumpla las condiciones necesarias para poder adaptarlo y realizar el entrenamiento.

Por último, tenemos la adaptación del conjunto de datos una vez conocemos la información que debemos incluir en el conjunto final.

Diseño de diferentes modelos clasificadores

Para realizar la clasificación del audio se probarán diferentes tipos de modelos clasificadores, básicos y más complejos, como las redes neuronales.

Se construirán una o varias arquitecturas de redes neuronales que sean capaces de obtener un buen rendimiento en la clasificación del audio una vez entrenadas.

Debido a que los modelos de redes neuronales artificiales no suelen obtener un buen rendimiento de primeras, se optimizarán dichos modelos modificando los parámetros necesarios.

Por otro lado, debido a que los medios físicos para realizar la implementación de los modelos es un ordenador personal, se intentará obtener la mayor optimización ejecutando el entrenamiento sobre GPU en vez de CPU.

Evaluación de los modelos

Una vez obtenidos todos los modelos, se evaluarán comparando diferentes métricas oportunas. La comparación se realizará en tablas y en gráficas para así mostrar los resultados de forma visual. También se debe presentar el modelo final que obtenga el mejor rendimiento.

Redacción de la memoria

Para recoger todo el trabajo realizado a lo largo del curso, se ha de redactar una memoria, que debe seguir una estructura similar a la de un artículo científico. El desarrollo de la memoria se realizará en LaTeX para conseguir una maquetación más limpia y presentable.

Capítulo 4

MATERIALES Y MÉTODOS

En este capítulo se va a describir la metodología seguida para realizar la obtención de los datos, su tratamiento, el entrenamiento de modelos de aprendizaje automático y el método de valoración de los modelos. En primer lugar se procederá a describir los conjuntos de datos sobre los que se ha trabajado, su obtención y un análisis exploratorio para conocer bien su contenido. Después se explicarán todas las técnicas de preprocesamiento de los datos aplicadas, detallando los *scripts* creados para este proceso. Posteriormente, se detallará por qué y cómo se ha realizado un análisis exploratorio de los datos finales ya preprocesados. También se describirán los diferentes clasificadores utilizados; entre ellos, clasificadores básicos y modelos de aprendizaje profundo. A continuación se describirá el método y las métricas que se van a evaluar para valorar un modelo de clasificación. Finalmente se definirán todas las herramientas utilizadas para llevar a cabo este trabajo.

4.1. Conjuntos de datos

Como primer paso, hace falta encontrar un conjunto de datos con las características específicas para resolver el problema planteado. Una vez encontrado se procederá a descargarlo y comprobar su utilidad. En este trabajo se han utilizado tres conjuntos de datos de distinta procedencia. La búsqueda del conjunto se ha realizado en Dataset Search. Esta página es un buscador de *datasets* realizado por Google donde podemos encontrar todo tipo de conjuntos de datos. Tras realizar una simple búsqueda encontré Hooktheory.

4.1.1. Hooktheory

El primero que se ha usado para hacer un acercamiento a cómo tratar los datos de forma correcta se llama Hooktheory. El conjunto de datos se obtuvo desde su web el 7 de octubre de 2020, pero actualmente se encuentra inaccesible. Estos datos se encontraban en formato hdf5 y, finalmente, fueron descartados por tener múltiples errores en la clasificación de acordes de las canciones.

Una vez descartado este conjunto se continuó con la búsqueda en Dataset Search y se llegó a McGill Billboard Project.

4.1.2. McGill Billboard Project

El conjunto de datos encontrado en el buscador estaba alojado en Kaggle. Kaggle es una plataforma gratuita donde se encuentran multitud de conjuntos de datos con la intención de que la gente encuentre soluciones a problemas con aprendizaje profundo u otras técnicas similares. Al acceder a dicho dataset, se vio que no era un conjunto de datos perteneciente al autor de la publicación en Kaggle, llamado Jacob van Steyn. En realidad pertenecía al área de tecnología musical de la escuela de música Schulich de la Universidad McGill. Por ello se utilizó el conjunto de datos original McGill Billboard Project [Burgoyne, 2021].

Gracias a un análisis exploratorio preliminar realizado sobre el conjunto de datos descargado se ha conocido más a fondo su contenido. Este conjunto de datos se compone de unas 890 anotaciones de canciones que comprende todo tipo de géneros de música moderna. Estas anotaciones incluyen una marca temporal de inicio de un acorde, la marca final y el nombre del acorde en sí. Desde su web es posible descargar varias versiones del conjunto de datos tan solo variando el número de clases de acordes detectadas. Al disponer de menos clases, agrupa acordes muy parecidos, con lo cual la precisión y exactitud se pierden. La versión más extensa disponía de 975 tipos de acordes diferentes, y la menos extensa, tan solo de 37.

Un problema que presenta este conjunto de datos es que no facilita las canciones anotadas. Para solucionar esto se realizó una lista de reproducción en Spotify donde se incluyeron a mano cada una de las canciones. Una vez completa la lista de reproducción o *playlist*, se comprobó que no era posible descargar una tan grande, así que hubo que dividirla en 9 listas de reproducción distintas de 100 canciones. Posterior-

mente, gracias a la herramienta TuneKeep Spotify Music Converter[1] se descargaron las canciones directamente desde la lista de reproducción.

Como se explicará más adelante, se decidió utilizar otro conjunto de datos para intentar solucionar algunos fallos que tenía el contenido de este. Por ello continuó la búsqueda. Esta vez, a través de Google se encontró la página web de ISMIR (*International Society for Music Information Retrieval*), que dispone de un gran conjunto de *datasets* orientados a la extracción de características de música. Entre ellos encontramos GuitarSet.

4.1.3. GuitarSet

El conjunto de datos GuitarSet [Xi et al., 2018] dispone de 360 audios de guitarra con sus respectivas anotaciones de diferentes características musicales, entre ellas, la anotación de los acordes. Las anotaciones de las características estaban en ficheros de extensión '.jams'. Estos datos distinguen también dos versiones; en una dispone de 42 clases de acordes distintas donde vemos una clasificación más general y, en otra versión, bastantes clases de acordes más, lo que permite que sea mucho más precisa. Se decidió trabajar con las 42 clases de acordes para facilitar el aprendizaje de los modelos a entrenar. La descarga de este conjunto de datos se realiza desde su web.

4.2. Preprocesamiento de los datos

Una vez descargados los conjuntos de datos y realizado un análisis exploratorio preliminar se debe de conocer si existe la necesidad de adaptar o preprocesar los conjuntos de datos que se tratarán. Aquí se observó que tanto el conjunto de datos de McGill Billboard Project como el de GuitarSet requerían una adaptación. El problema es que los datos que se facilitan son tan solo los audios en MP3 y los acordes con unas marcas temporales. Lo que se necesita es un único fichero CSV separado en líneas que representen un fragmento de audio de una duración preestablecida, como 0.25 segundos, con las etiquetas de los acordes y las muestras de audio relativas al acorde a continuación.

[1] Los enlaces a todas las herramientas utilizadas se facilitan en el apartado 4.7.

4.2.1. McGill Billboard Project

Para adaptar el conjunto de McGill Billboard Project se ha requerido realizar una serie de *scripts* en Python que posteriormente se pasaron a un *notebook*. Los programas tenían las diferentes funciones de:

- **Crear una tabla de acordes** con un identificador y su respectivo nombre y exportarlo en un CSV. Esta tabla tiene la utilidad de poder codificar el nombre del acorde con un número. El programa recorre todas las anotaciones de cada canción y añade a una lista todos los acordes distintos que va encontrando. Finalmente guarda la lista con sus índices en un fichero de texto. El esquema de la figura 4.1 representa el proceso descrito.

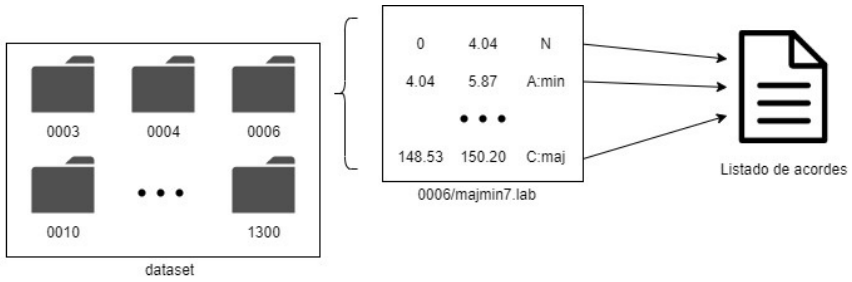

Figura 4.1: Diagrama del *script* para crear una tabla de acordes

- **Renombrar los ficheros de audio de cada canción**. Aquí introduce en el título de cada canción un índice previamente asignado en las anotaciones. De esta manera se unifican títulos y es más sencillo crear el conjunto de datos final. El programa lee una hoja de cálculo que viene en la descarga del conjunto que hace de índice. Esta hoja de cálculo relaciona el título, autor y año de la canción con un identificador que tiene. Luego extrae del título de cada audio su número (número secuencial por orden de descarga), título, extensión y el número de la lista de reproducción. Para reasignar el índice al correspondiente en el conjunto de datos recorre el índice buscando coincidencias de varios campos. Después, reasigna el índice manteniendo varios ceros delante para que la longitud del índice sea de 4 cifras y así se asemeje al índice del conjunto de datos. Por último, renombra los audios con su nuevo índice, título y extensión. El esquema de la figura 4.2 representa el proceso descrito.

- **Cortar los audios de canciones descargadas**. Debido a que las canciones han sido descargadas con una herramienta aparte, la mayoría de las marcas de tiempo estaban desplazadas. Esta herramienta no obtiene directamente el fichero de audio, sino que reproduce el audio a más velocidad que la original y

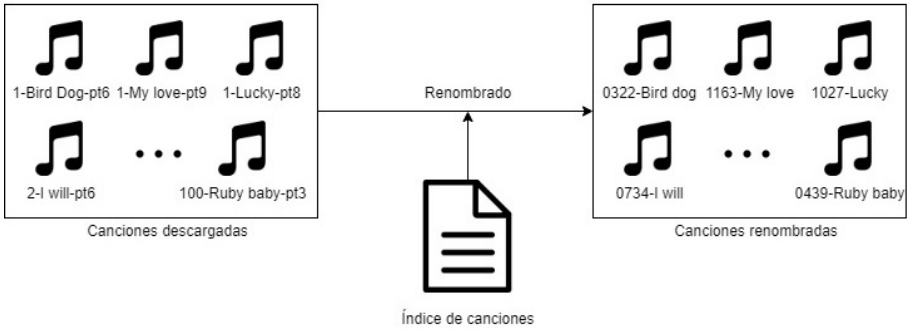

Figura 4.2: Diagrama del *script* para renombrar los audios

graba el sonido reproducido. Esto puede hacer que pierda calidad el audio y a su vez que se desajuste la duración. Para solucionar esto, el programa detecta la duración del silencio inicial del audio y le resta la marca inicial de tiempo del primer acorde detectado para cortar el audio y hacer que comience a la vez que en la anotación. También convierte las canciones estéreo en audio mono para reducir así el número de muestras contenidas. El esquema de la figura 4.3 representa el proceso descrito.

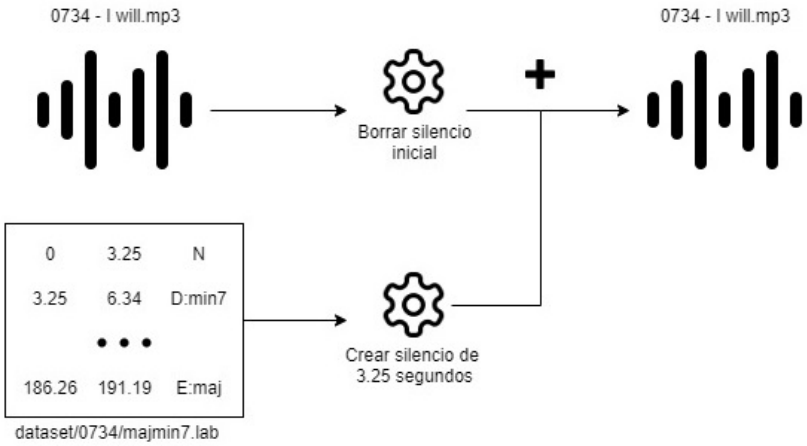

Figura 4.3: Diagrama del *script* para cortar los audios

- **Crear el conjunto de datos final en un CSV**. Aquí se unifica la tabla de acordes con los audios descargados y con las anotaciones de las marcas de tiempo de cada acorde. En cada línea del CSV final se puede encontrar un primer número representando al acorde contenido y luego 11 025 números que corresponden a las muestras de audio durante 0.25 segundos. Estas 11 025 muestras se deben a que el audio dispone de una frecuencia de muestreo de 44 100 Hz. Esto quiere decir que en un segundo existen 44 100 muestras de audio, las cuales dividas entre 4 son 11 025 muestras. Se decidió hacer cada muestra de 0.25 segundos debido a que si es mayor el tiempo, contendría demasiadas variables

para posteriormente hacer la clasificación y, si es menor, posiblemente no pueda identificarse el acorde. El programa carga el listado de acordes y recorre todos los audios, abre el fichero de las anotaciones de cada canción y lee las marcas de tiempo finales. Mientras esta marca es mayor al instante de tiempo actual se particiona la canción otros 0.25 segundos donde se incluye el índice de la anotación del acorde correspondiente. Una vez que es mayor a la marca final de tiempo avanza con el siguiente acorde. Finalmente lo guarda en un fichero CSV. El esquema de la figura 4.4 representa el proceso descrito.

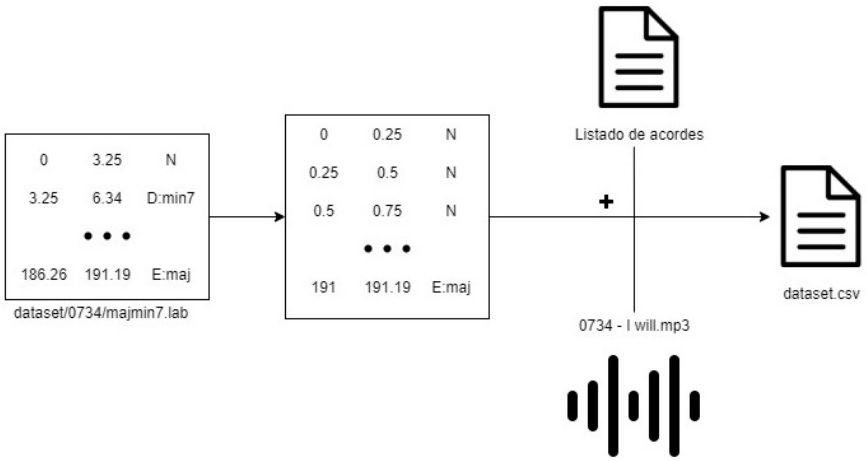

Figura 4.4: Diagrama del programa para generar el fichero CSV

4.2.2. GuitarSet

El preprocesamiento realizado en este conjunto de datos ha sido muy parecido al del conjunto anterior. Esta vez se hicieron unos pequeños cambios, como juntar todo el preprocesamiento en un solo *notebook* para hacer la ejecución más sencilla. En este conjunto no era necesario cortar los audios debido a que él mismo nos los facilitaba y, de esta manera, las marcas de audio coincidían bien. Luego se renombraron los audios para darles un formato más útil para localizarlos. Se realizó una simple comprobación de la frecuencia de muestreo, ancho de muestra y el número de canales de cada canción. Después se separó de los ficheros '.jams' la información necesaria entre todas las anotaciones y se guardaron en ficheros separados con el mismo formato indicado para el otro conjunto, marca temporal de inicio, marca temporal de fin y el nombre del acorde. A continuación, se obtuvo el listado de acordes recorriendo todas los ficheros generados con las marcas de tiempo y acordes para añadir a una lista todos los acordes distintos y posteriormente guardarlos en un archivo CSV. Por último, se generó el conjunto de datos final. Esto se realiza igual que en el conjunto anterior.

Solo cambia que, para avanzar de acorde, la marca de tiempo final debe ser menor que el instante actual más la mitad de la duración del fragmento (0.25 segundos).

Sobre este conjunto de datos también se formó otro conjunto final donde se redujeron las clases de acordes que podíamos encontrar, agrupando los acordes más parecidos. Todos los acordes que antes denotaban si eran mayores, menores, con 7ª, disminuidos y demás, 42 en total, se reducen a tan solo 12, que solo muestran la tónica del acorde. Por ejemplo C:min $\rightarrow$ C o G#:dim7 $\rightarrow$ G#. Esto se realizó para analizar el impacto que puede producir en la precisión final de la clasificación al tener menos clases. Para esto hubo que generar un nuevo índice de acordes, el cual se realizó a mano. En este podemos encontrar las 12 clases de acordes con sus índices y modificar la generación del CSV para poder asignar correctamente los índices a las muestras de audio.

4.2.3. LEAF

Como se ha explicado en el capítulo anterior, LEAF es un *frontend* para generar *mel-filterbanks* de un audio. En este trabajo se va a utilizar LEAF para obtener un conjunto de imágenes y entrenar una red neuronal convolucional con ellas. Primero se ha tenido que estudiar con más profundidad el código de GitHub para comprender el funcionamiento del *frontend*. Para la utilización de leaf-audio ha sido necesario instalarlo en WSL (*Windows Subsystem for Linux*) debido a problemas con alguna librería al hacerlo en Windows. Se han instalado también una serie de dependencias necesarias para su funcionamiento. La librería de LEAF da la opción de entrenar su modelo adaptándolo a otro conjunto de datos o de usar una serie de modelos preentrenados para generar imágenes de manera muy sencilla. Esta biblioteca permite generar los *mel-filterbanks* con distintas configuraciones. En la figura 4.6 se representa el procesamiento del audio realizado por el frontend de LEAF para cada configuración.

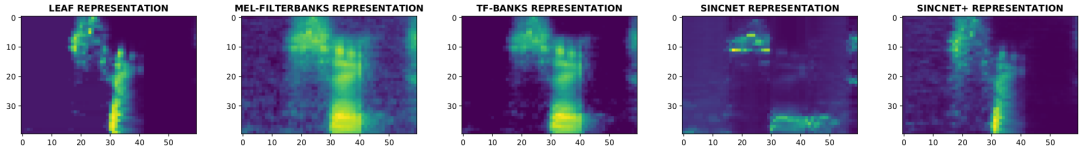

Figura 4.5: Imágenes resultantes de cada configuración de LEAF preentrenada [Zeghidour et al., 2021].

Para aplicar LEAF al conjunto de datos de GuitarSet, necesitamos hacer un preprocesamiento extra con el conjunto de datos y así trabajar directamente con las imágenes. Partimos del CSV obtenido en el último paso del preprocesamiento. Ahora programamos un *notebook* y en él cargamos el CSV final. Este conjunto de datos lo generamos de nuevo 3 veces. Con una duración de fragmento de 0.25 segundos y 42 clases, con una duración de fragmento de 1 segundo y 42 clases y con una duración de fragmento de 1 segundo y 12 clases. El hecho de cambiar la duración del fragmento a 1 segundo crea 44 100 variables en vez de 11 025. Pero debido a que vamos a clasificar imágenes con una red neuronal convolucional, no es tan importante el número de parámetros que contenga, no como con el audio en bruto. También al observar que un acorde suele durar en torno a los 2 segundos de media en estos audios, se pensó que con 1 segundo de duración de fragmento era mucho más representativo el acorde.

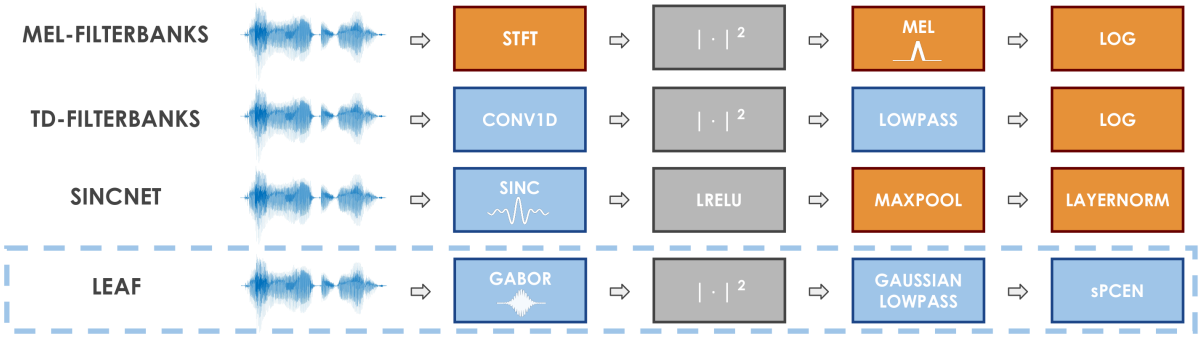

Figura 4.6: Desglose del procesamiento requerido para la obtención de cada configuración. Los recuadros naranjas son fijos, mientras que los procesamientos en recuadros azules son entrenables. Los recuadros grises representan las funciones de activación [Zeghidour et al., 2021].

El *notebook* carga en memoria el conjunto completo y los separa en 4 conjuntos disjuntos: `X_train`, `y_train`, `X_test` e `y_test`. En los conjuntos `X` encontramos en cada línea tan solo las 11 025 o 44 100 variables que componen el fragmento de audio. Y en los conjuntos `y` encontramos en cada línea el índice de la clase de acorde correspondiente a cada línea del conjunto `X`. Para hacer la separación de los datos, se mezclan las líneas y un 80 % de los fragmentos de tiempo se incluyen en los conjuntos de *train* o entrenamiento, y el otro 20 %, en test o prueba.

Una vez que esté todo cargado, se recorren los conjuntos `X` y, con la herramienta proporcionada por Google, se genera una imagen por fragmento que guardamos aparte. Guardamos también en ficheros separados los conjuntos de `y_train` e `y_test`. Por último, se añade un módulo opcional en el *notebook*, que permite separar las imáge-

nes generadas en distintas carpetas para facilitar un método de carga de las imágenes en el clasificador.

4.3. Análisis exploratorio

Antes de comenzar a implementar clasificadores de los datos, es crucial conocerlos bien. Para ello se realiza un análisis exploratorio de los datos. Este consiste en hacer un estudio de las características más importantes del conjunto de datos. Se ha realizado un análisis de estas características:

- Número de muestras
- Número de variables por línea
- A qué corresponde cada variable
- Número de clases que clasificar
- Número de muestras por clase

El análisis exploratorio ha sido realizado sobre un *notebook* de Python. Ahora se mostrarán los resultados de los análisis exploratorios realizados a cada conjunto de datos obtenido.

4.3.1. McGill Billboard Project

El conjunto de datos en formato CSV es una tabla de 743 108 filas y 11 026 columnas. Cada fila se compone de un número que representa a un acorde y de 11 025 variables restantes se que corresponden con cada muestra de audio. Si cada fila equivale a 0.25 segundos de audio, se dispone de un total de 185 777 segundos, lo que son 51 horas, 36 minutos y 17 segundos de audio. El conjunto de datos en formato CSV ocupa un total de 39.1 GB y para poder cargarlo en memoria se tiene que dividir en una serie de *chunks* o secciones debido a que solo disponemos de 16 GB de RAM.

Podemos observar en el análisis exploratorio que tenemos 85 clases de acordes diferentes. Como primer acorde tenemos 'N', que representa la ausencia de sonido, y otro 'X', que representa la imposibilidad de asociar un acorde al sonido. En la figura

4.7 podemos ver una gráfica donde se expone el número de muestras por cada acorde. Como aquí se refleja, hay una serie de acordes que predominan, entre los que encontramos los acordes mayores como do mayor, fa mayor, re mayor o sol mayor, entre otros, y también un gran número de 'N' y de 'X', que supera a cualquier clase. Los acordes menos populares suelen ser los más complejos, como disminuidos, con 7ª mayor y acordes sobre notas con sostenido o bemol. Esta gráfica refleja una realidad bastante interesante sobre la utilización de estos acordes en la música moderna de distintos géneros. Con esto se resalta el claro desequilibrio de clases del conjunto de datos.

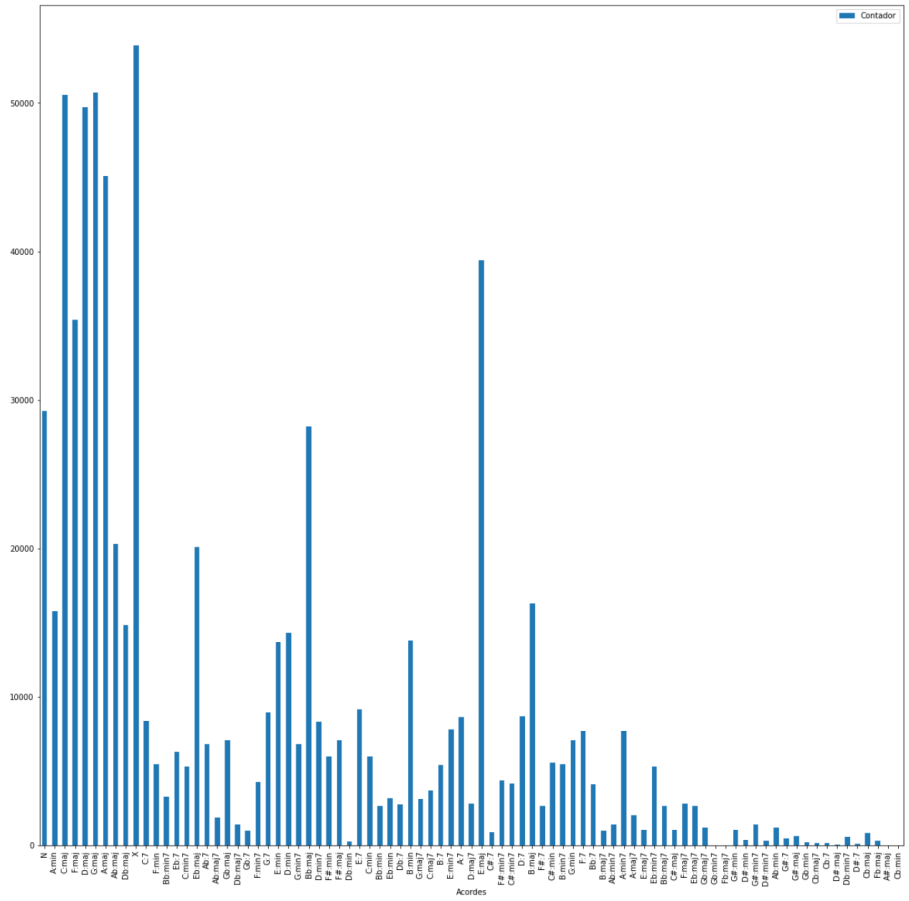

Figura 4.7: Distribución de muestras por acorde en el conjunto de datos de McGill

4.3.2. GuitarSet - Fragmentos de 0.25 segundos - 42 clases

Este conjunto de datos es una tabla de 22 038 filas y 11 026 columnas. Las variables son las mismas que en el conjunto de McGill. Como disponemos de 0.25 segundos de audio por muestra, obtenemos un total de 5509.5 segundos, que se traducen en 1 hora, 31 minutos y 49.5 segundos de audio. Como podemos observar, este conjunto de datos es mucho más pequeño que el anterior, ya que solo ocupa 1.05 GB en

formato CSV. Una ventaja es que no hay que particionarlo para cargarlo en memoria.

Este conjunto dispone de 42 acordes distintos y no encontramos ninguna clase 'X' o 'N'. Estos audios se diferencian mucho de los anteriores debido a que son solamente grabaciones de guitarra sin producción, ecualización y masterización, mientras que los otros eran canciones bien producidas. En la figura 4.8 se puede observar una tendencia parecida al conjunto anterior, solo que esta vez son más representativos algunos acordes mayores sobre notas con sostenido. Esto se debe al número reducido de audios que hay y a que muchas están en tonalidades poco comunes donde esos acordes ejercen funciones tonales muy importantes. También hace falta resaltar que existe un desequilibrio muy claro en los datos, donde no todas las clases de acordes están representadas por igual. Este hecho suele repercutir negativamente en el rendimiento de los modelos de aprendizaje automático.

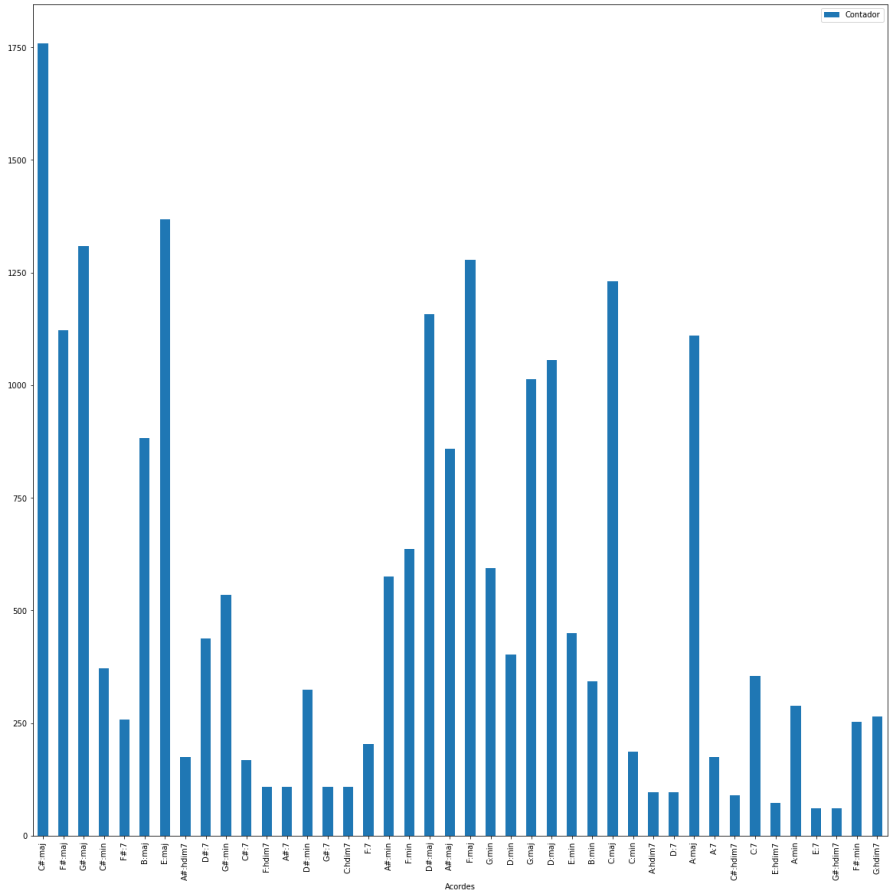

Figura 4.8: Distribución de muestras por acordes en GuitarSet

4.3.3. GuitarSet - Fragmentos de 1 segundo - 42 clases

Este conjunto de datos es una tabla de 5484 filas y 44 101 columnas. La primera variable de una línea sigue siendo la representación numérica de un acorde, y las 44 100 restantes son las muestras de audio. Esta es la única diferencia con el conjunto de datos GuitarSet con fragmentos de 0.25 segundos y 42 clases. Este conjunto al mantener la misma distribución de acordes sigue estando desequilibrado.

4.3.4. GuitarSet - Fragmentos de 1 segundo - 12 clases

Este conjunto de datos se diferencia de los anteriores en la reducción de clases que tiene. La tabla es la misma, tan solo cambian el índice de los acordes. Tal y como se representa en la figura 4.9, se puede observar que los 12 acordes tienen una presencia bastante igualada, a diferencia de los dos anteriores conjuntos, este modifica su distribución de clases y consigue un mayor equilibrio entre el número de representaciones de cada clase. Por otra parte, la duración de los fragmentos es de 1 segundo.

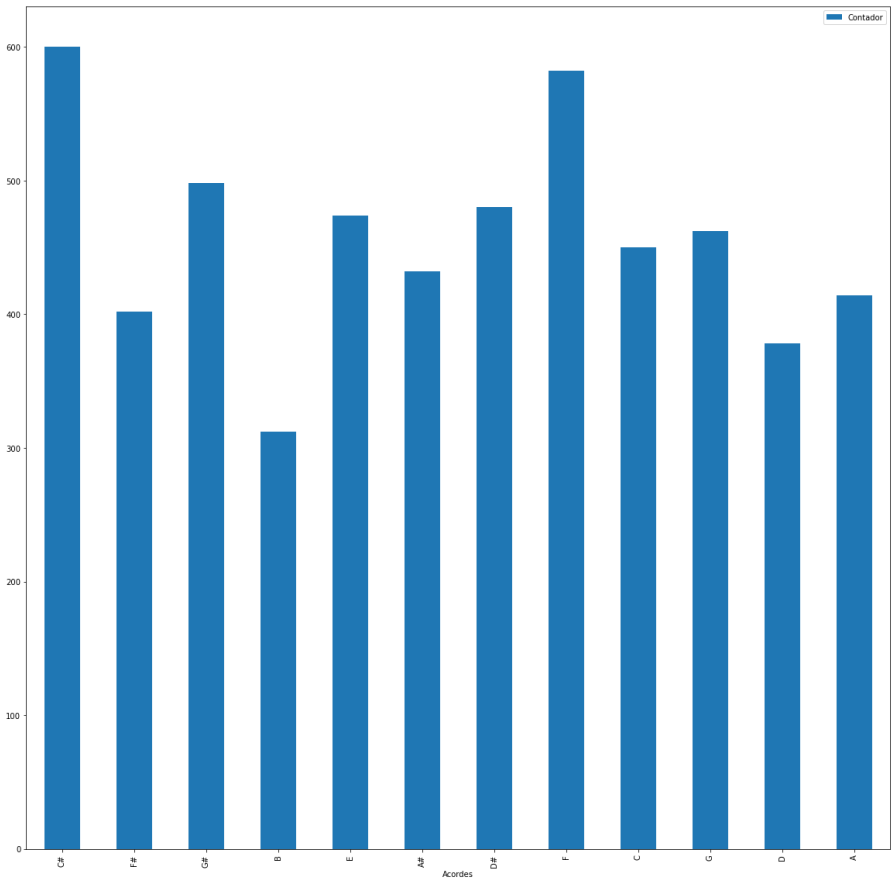

Figura 4.9: Distribución de muestras por acordes reducidos en GuitarSet

4.3.5. Comparación general

La tabla 4.1 representa la comparación entre los diferentes conjuntos de datos tratados.

Dataset	Nº clases	Duración fragmento	Nº fragmentos	Nº muestras
McGill	85	0.25 s	743108	11 025
GuitarSet	42	0.25 s	22038	11 025
		1 s	5484	44 100
	12	0.25 s	22038	11 025
		1 s	5484	44 100

Tabla. 4.1: Comparación entre diferentes conjuntos de datos

Aquí se observa que el conjunto de datos McGill es unas 33.71 veces más extenso que GuitarSet. A su vez, tiene más del doble de acordes distintos que clasificar.

4.4. Clasificadores básicos

Una vez se conocen los datos que se van a tratar, planteamos cómo sería su clasificación sin utilizar aprendizaje profundo, solamente usando clasificadores básicos, como Random Forest, kNN o regresión logística. Esto ha sido programado también en un *notebook* de Python.

En primer lugar, se ha cargado el conjunto de datos o una sección o *chunk* en memoria, en función del tamaño. Se debe recordar que el conjunto de McGill tiene un tamaño de 39.1 GB y es necesario dividirlo en partes más pequeñas. También es muy importante separar los datos en cuatro conjuntos disjuntos: `X_train`, `y_train`, `X_test` e `y_test`. Como se explicó anteriormente, vamos a utilizar los conjuntos de entrenamiento para alimentar a nuestro clasificador con ejemplos para que aprenda a clasificar. El conjunto `X` contiene las diferentes muestras de audio, y el `y`, los índices de los acordes que se han de identificar. El 80 % del conjunto de datos está destinado al conjunto de entrenamiento, y el otro 20 %, al conjunto de test.

- **Random Forest**: para utilizar Random Forest con nuestros conjuntos de datos solo hace falta crear el modelo de Random Forest dado por la librería `sklearn` en Python. A continuación, hay que introducir los conjuntos *train* en el clasificador para entrenarlo con el método `fit`. Entonces, comenzará a entrenarse durante

un periodo de tiempo y posteriormente se deberá medir con los conjuntos de test o validación cómo de buena es la clasificación que ha aprendido a realizar el modelo. El resultado obtenido ha sido bajo en ambos conjuntos.

- **kNN**: un kNN o *k nearest neighbor* necesita normalizar previamente los datos con los que va a tratar; por ello, se pueden utilizar diversos tipos de normalizadores. En este caso usamos `MinMaxScaler`. Este transforma los datos dejándolos a escala en un determinado rango de valores. Para utilizar kNN se debe definir el modelo y ajustar algunos parámetros, como el tamaño del vecindario, entrenarlo con el método `fit` e introducir los conjuntos de entrenamiento ya normalizados. El rendimiento superó muy levemente al Random Forest en ambos modelos, pero seguía siendo bajo.

- **Regresión logística**: para ejecutar una regresión logística se puede utilizar un modelo ya dado por `sklearn` o simplemente hacer el modelo de una red neuronal simple con una sola neurona. En este caso hemos usado la dada por `sklearn`. Aquí, una vez más, definimos el modelo, se configura algún parámetro como el número de iteraciones máximas y se entrena introduciendo los conjuntos de entrenamiento. El rendimiento en ambos modelos ha sido más bajo aún que el Random Forest.

4.5. Redes neuronales

El siguiente paso es la clasificación del conjunto de datos entrenando redes neuronales de diversos tipos. Estas redes neuronales pueden haber sido definidas desde cero o utilizando redes neuronales conocidas para diversos usos y adaptándolas a los datos de uno. Se comenzó a probar por redes neuronales básicas, redes convolucionales profundas sobre audio y redes convolucionales profundas sobre imágenes.

4.5.1. Perceptrón multicapa

Se comenzó realizando una red neuronal básica, como el perceptrón multicapa, para ver cómo afrontaba el problema y ver hasta qué punto era necesario utilizar aprendizaje profundo. La carga de los datos y el procesamiento previo no difiere de los clasificadores básicos; la única diferencia es que, después de separar los diferentes conjuntos de datos, cada línea de `y_test` e `y_train` se transforma en un vector

binario, mientras que antes eran números enteros. Esto se realiza para facilitar la tarea de clasificación a las redes neuronales. Se han hecho múltiples arquitecturas para intentar mejorar la precisión final de la red, todas ellas sobre un modelo secuencial y con múltiples capas `Dense`. La arquitectura final para el conjunto de McGill ha sido de cinco capas `Dense` con activación ReLU, de las cuales las tres primeras con 11 025 neuronas, la siguiente con 5440 neuronas, después una de 680 neuronas y, por último, una capa final con el número de neuronas igual al de clases, 85. La activación de esta última capa es *softmax*, ya que es mucho mejor para hacer la clasificación. El rendimiento de la red es incluso más bajo que los clasificadores; por ello, se decide probar con otra arquitectura. En cuanto al conjunto de GuitarSet, las pruebas se hicieron con arquitecturas similares, adaptando el número de neuronas en la capa final al número de clases de este conjunto de datos. Tras múltiples intentos cambiando el número de capas y el número de neuronas por capa, observamos un rendimiento superior al obtenido por los clasificadores básicos. El diagrama de la figura 4.10 representa la arquitectura de la red.

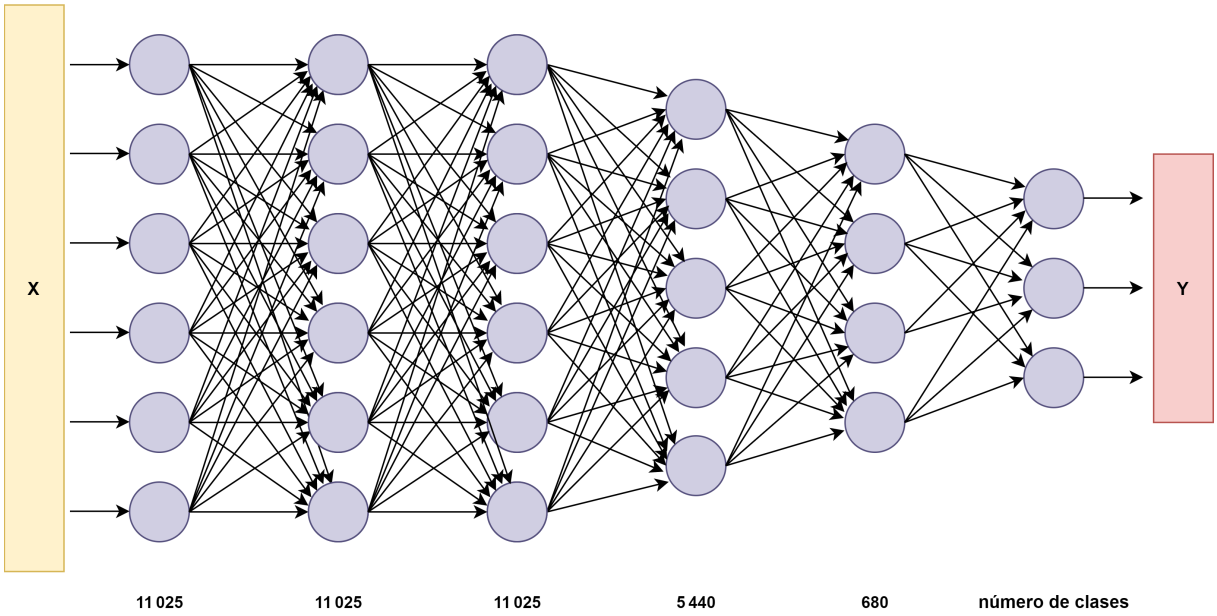

Figura 4.10: Diagrama del perceptrón multicapa utilizado

4.5.2. Redes Neuronales Convolucionales sobre audio en bruto

La red neuronal convolucional fue la arquitectura de red elegida para probar la mejora de precisión de la red. Esta arquitectura, basada en una parte de convoluciones y *max-pooling* y en otra de neuronas totalmente conectadas, es capaz de extraer carac-

terísticas de los datos y abstraerlas en un menor espacio. De esta manera se pensó como la mejor opción que probar. La carga de los datos era prácticamente igual que en la red neuronal básica, la única diferencia es que hubo que redimensionar los conjuntos `X_train` y `X_test` a (`n_timesteps`, `n_features`, 1) para poder utilizar la capa de `Conv1D` de keras. Un ejemplo de arquitectura de la red estaría formado por un par de capas `Conv1D` con 64 filtros, un `Dropout` a 0.5, un `MaxPooling1D`, `Flatten`, una `Dense` con 100 neuronas y otra `Dense` con el mismo número de neuronas que clases a clasificar. Tras múltiples configuraciones con el conjunto de McGill, se sigue obteniendo un rendimiento demasiado bajo, menor incluso al de los clasificadores básicos.

En este momento solo se estaba utilizando el conjunto de datos de McGill y se pensó que el problema era de los datos, ya fuera por errores de cálculo del umbral al cortar los audios o bien porque, al descargar por nuestra cuenta las canciones, muchas de ellas no se encontraban en su versión original y se tuvieron que descargar en su versión remasterizada. Esto podría suponer que al grabarlas de nuevo se cambiaran los tiempos de acordes o que incluso se cambiase la tonalidad y esto produjera otros acordes distintos. Como era un conjunto de datos demasiado extenso para poder hacer las comprobaciones y las correcciones a mano, se decidió buscar otro. Ahí fue cuando entró GuitarSet. A partir de este momento queda descartado el *dataset* de McGill y solo se utiliza GuitarSet.

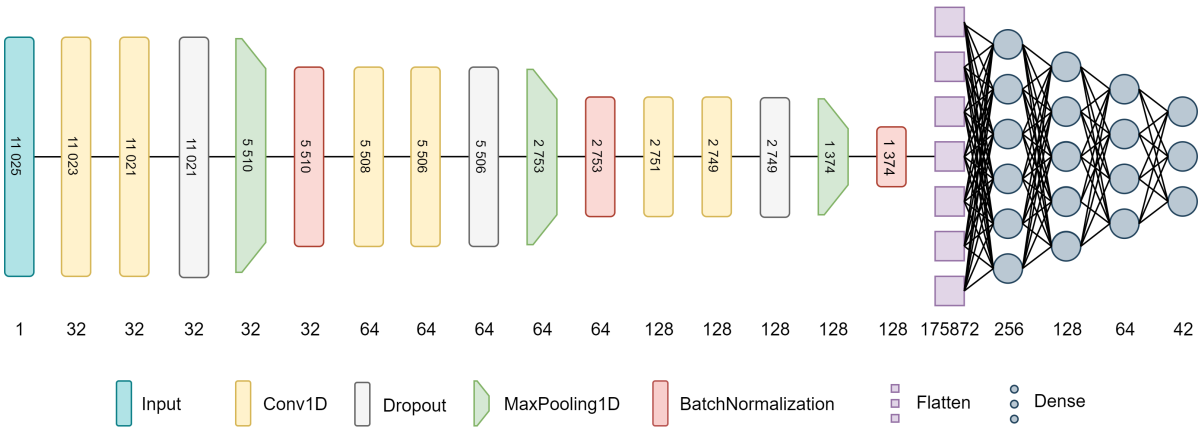

Figura 4.11: Diagrama del CNN utilizado sobre muestras de audio

En el conjunto de datos de GuitarSet los resultados superaban a los de McGill, pero aun así se seguía obteniendo un rendimiento bajo, un poco inferior al perceptrón multicapa. Por ello se decidió seguir mejorando la arquitectura de la red. Su forma final fue esta: un bloque de dos capas `Conv1D` de 32 filtros, un `Dropout` de 0.5, un `MaxPool1D` y un `BatchNormalization`. Este bloque se repite 2 veces más cambiando

los filtros convolucionales a 64 y luego a 128. Luego un `Flatten` y un bloque de 3 `Dense` con 256, 128 y 64 neuronas respectivamente con un `Dropout` entre ellas y finalmente un `Dense` con el mismo número de neuronas que de clases. Se obtuvo también un rendimiento bajo. En la figura 4.11 se puede visualizar la arquitectura de la red.

En la fecha en la que se estaban realizando estos experimentos, se publicó el artículo de LEAF, por lo que se decidió utilizarlo para mejorar el rendimiento.

4.5.3. Redes Neuronales Convolucionales sobre imágenes

Una vez generado el nuevo conjunto de imágenes, se carga en el *notebook*. Esta vez la carga de datos ha sido diferente, ya que se ha utilizado ImageDataGenerator para crear un generador de imágenes y así no tener que cargarlas todas en memoria. Una vez cargado, hemos probado con tres arquitecturas diferentes. Inicialmente la arquitectura consistía en: una capa `Conv2D` con 32 filtros, `BatchNormalization`, `MaxPooling2D`, otra `Conv2D` con 64 filtros, `MaxPooling2D`, `Conv2D` de 128 filtros, `MaxPooling2D`, `Flatten` y un bloque de cuatro `Dense` de 1024, 512, 256 y 128 neuronas, `BatchNormalization`, `Dropout` de 0.5 y una capa final `Dense` con el mismo número de neuronas que de clases. Todas las funciones de activación a excepción de la ultima capa son ReLU. La última es softmax.

A continuación, se probó con un modelo consistente en: un bloque de `Conv2D` y `MaxPooling2D` repetido cuatro veces cambiando el número de filtros de 32, 64, 128 y 256. Un `Flatten` y cuatro `Dense` con 1024, 512, 256 y 128 y un `Dense` final con el mismo número de neuronas que de clases. Todas las funciones son ReLU excepto la última que es softmax.

Finalmente se llegó a un modelo que consistía en: `BatchNormalization`, tres bloques de `Conv2D` y `MaxPooling2D` con 32, 64 y 128 filtros. Un `Flatten`, cuatro `Dense` con 1024, 512, 256 y 128 neuronas (esta última sin función de activación), `BatchNormalization`, `Activation` de tipo ReLU, `Dropout` de 0.2 y una `Dense` final con el mismo número de neuronas que de clases. Todas las activaciones son ReLU excepto la última que es softmax. Este diagrama se puede visualizar en la figura 4.12

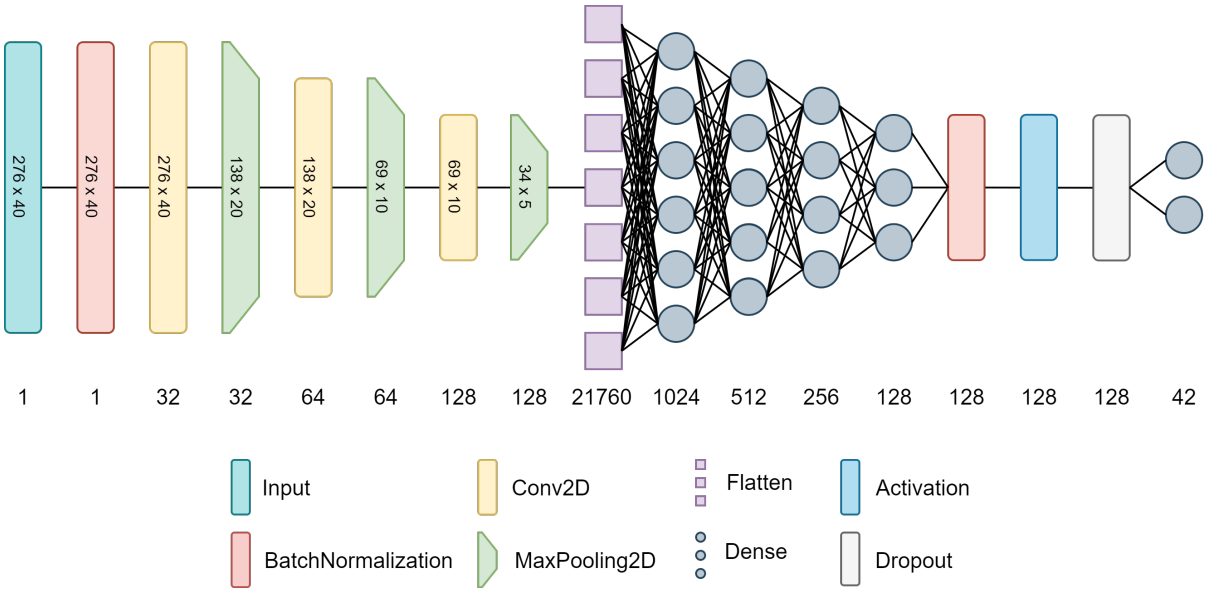

Figura 4.12: Diagrama de la arquitectura de red convolucional final

Tras múltiples intentos obteniendo resultados desfavorables, finalmente se obtuvo un modelo con un buen rendimiento. Los datos tenían la configuración de una duración de fragmento de 1 segundo y las 42 clases de acordes. Una vez se llegó a obtener tal rendimiento se dio por finalizada la experimentación.

4.6. Valoración de modelos

Para poder valorar un modelo de clasificación, primero se ha de conocer una serie de términos. Entre ellos podemos considerar como más importante el término de pérdida o error y el de exactitud.

El término de pérdida o error (*loss* en inglés), es un valor numérico obtenido como resultado de una función de pérdida. Hay múltiples funciones de pérdida, pero todas reflejan la diferencia entre un valor esperado y un valor predicho. Este término nos interesa porque es muy importante para el aprendizaje de nuestro modelo. El modelo se debe encargar de minimizar la pérdida en cada iteración del aprendizaje.

El término de exactitud (*accuracy* en inglés), es otro valor numérico representado normalmente en tanto por uno que refleja el porcentaje de casos que el modelo ha acertado. Se calcula como el número de predicciones correctas entre el número total

de predicciones. No debe confundirse con la precisión.

$$\text{Exactitud} = \frac{\text{VP} + \text{VN}}{\text{VP} + \text{VN} + \text{FP} + \text{FN}} \tag{4.1}$$

VP significa verdaderos positivos, VN, verdaderos negativos, FP, falsos positivos y FN, falsos negativos.

En función a estos dos términos podemos evaluar de manera general un modelo de clasificación, obteniendo los valores de pérdida y exactitud del conjunto de entrenamiento y comparándolo con los del conjunto de test. Si estos términos son similares en ambos conjuntos, la exactitud es alta y la pérdida baja, podemos pensar *a priori* que tenemos un buen modelo de clasificación. Dependiendo del problema consideramos que una exactitud es suficientemente alta o no. Determinar el rendimiento de un modelo mediante la exactitud puede darnos una idea equivocada. Si un conjunto de datos, como en este caso, está desequilibrado, la exactitud no es una medida representativa, ya que puede hacer que un modelo parezca mejor de lo que es.

Para conocer mejor los resultados y ver con más exactitud lo buena que es la clasificación del modelo debemos conocer algunos términos más.

El término de precisión (*precision* en inglés) es un valor numérico representado normalmente en tanto por uno que recoge la proporción de valores correctos positivos medidos entre el número total de positivos predichos. Esto quiere decir que si en una canción hemos detectado ocho acordes de do mayor y siete de ellos eran do mayor, el modelo ha tenido una precisión de 7/8 = 0.875. Suponiendo que en la canción había en total 12 acordes de do mayor, el término de precisión no recoge este fallo. Para ello requerimos el término de exhaustividad.

$$\text{Precisión} = \frac{\text{VP}}{\text{VP} + \text{FP}} \tag{4.2}$$

El término de exhaustividad o sensibilidad (*recall* en inglés) refleja la proporción de los valores correctos medidos entre todos los valores correctos. Como hemos mencionado anteriormente, si suponemos que se detectan ocho acordes de do mayor entre los 12 que hay pero de ellos solo se detectan correctamente siete, podemos decir que la exhaustividad es de 7/12 = 0.583.

$$\text{Exhaustividad} = \frac{\text{VP}}{\text{VP} + \text{FN}} \tag{4.3}$$

Para comparar bien ambos términos tenemos el término valor-f (*f-score* en inglés). Este valor es una media armónica entre la precisión y la exhaustividad.

$$\mathrm{F}_1 = 2 * \frac{\text{Precisión} \cdot \text{Exhaustividad}}{\text{Precisión} + \text{Exhaustividad}} \tag{4.4}$$

Estos valores se calculan para cada clase a clasificar y, posteriormente, se puede realizar una media de estos, de forma ponderada o no. La ponderación se hace teniendo en cuenta el número de veces que se representaba en el conjunto de datos cada clase. Debido al desequilibrio de los conjuntos de datos se tendrá en cuenta la media ponderada. Podemos decir que, cuanto más alto sea el valor-f, mayor rendimiento tiene el modelo.

Otra métrica que se debe tener en cuenta es el coeficiente kappa de Cohen. Este valor representa si el resultado ha sido correcto por azar o por la distribución de los datos o si, por el contrario, ha sido por todo el conocimiento aprendido en el conjunto de entrenamiento.

$$k = \frac{\text{Pr(a)} - \text{Pr(e)}}{1 - \text{Pr(e)}} \tag{4.5}$$

Pr(a) corresponde al acuerdo observado relativo entre los observadores, y Pr(e) es la probabilidad hipotética de acuerdo por azar, utilizando los datos observados para calcular las probabilidades de que cada observador clasifique aleatoriamente cada categoría.

Como última métrica que se ha de tener en cuenta para conocer el rendimiento de los modelos de clasificación tenemos la matriz de confusión. Esta herramienta permite visualizar de forma gráfica el rendimiento de un modelo de clasificación. Esto permite ver fácilmente si el modelo está confundiendo dos clases. Cada columna de la matriz representa el número de predicciones de cada clase, mientras que cada fila representa las instancias en la clase real.

4.7. Herramientas utilizadas

Para la realización de todo el proceso de experimentación se han utilizado multitud de herramientas que han permitido hacer todas las tareas necesarias. A continuación, se procederá a explicar para qué sirve y por qué se ha elegido cada herramienta.

- HDFView es una herramienta que se ha utilizado para poder abrir el fichero hdf5 del primer conjunto de datos. HDFView es gratuita y ejerce la función de abrir ficheros en este formato de manera muy simple.

- Overleaf es un editor de texto online que trabaja con LaTeX. La redacción de esta memoria en Latex se podría haber hecho con otro programa en local, pero se ha decidido usar Overleaf para poder redactar en línea de manera colaborativa.

- TuneKeep Spotify Music Converter es un programa para descargar cualquier canción de Spotify en formato MP3. Esta aplicación permite también la descarga de listas de reproducción. Se ha elegido esta aplicación por permitir descargar todas las canciones de una lista de reproducción y, a su vez, por poder formatear el nombre del fichero de cada canción libremente o utilizando índices.

- Python es uno de los lenguajes de programación más extendidos del mundo. Este lenguaje de muy alto nivel incluye muchas funciones muy útiles que permiten abstraerse considerablemente de la programación a bajo nivel. Se ha elegido este lenguaje porque hay abundancia de librerías para realizar el tratamiento necesario con los datos y, a su vez, es el lenguaje por excelencia donde se programan redes neuronales. También es un lenguaje que conocía previamente y con el que ha sido mucho más fácil empezar a trabajar.

- Jupyter es un entorno de trabajo para Python que permite programar en *notebooks*. Esto es lo mismo que un *script* de Python cualquiera pero con la posibilidad de ejecutarse por bloques de código y donde se guardan los resultados de las ejecuciones como si se tratase de un documento de texto. También es posible mezclar bloques de Python con bloques en formato Markdown. Se ha elegido hacerlo en *notebooks* por su facilidad para guardar los resultados y por no tener que ejecutar el *script* entero cada vez.

- Sklearn es una librería de Python donde se encuentran los modelos clasificadores básicos como Random Forest, kNN y regresión logística, además de funciones como `train_test_split` para separar los datos en diferentes conjuntos. Existen otras librerías con funciones bastante similares a las descritas, pero se ha decidido elegir estas por la confianza de la comunidad de *machine learning* en ella y su gran soporte en foros.

- Tensorflow es la librería de Python por excelencia de aprendizaje automático. Esta librería permite abstraerse de la programación a bajo nivel de las redes neuronales, como la función de *backpropagation*, y ofrece múltiples funciones muy útiles para su control y evaluación. Existen otras herramientas también útiles,

pero se ha decidido elegir esta por su gran popularidad y por estar familiarizado anteriormente con ella.

- Keras es una librería basada en Tensorflow que abstrae aún más la programación de redes neuronales de lo que hace Tensorflow. Esta librería se encuentra incluida dentro del mismo Tensorflow, lo que la hace muy cómoda y fácil de usar.
- Autokeras es una de las librerías de AutoML que existen. Esta librería es muy fácil de utilizar y no requiere de mucho conocimiento previo. Se ha decidido utilizar esta por sus facilidades al crear modelos autoconfigurables.
- Leaf-audio es la librería que se ha creado para el proyecto LEAF. Esta es una herramienta para poder obtener *mel-filterbanks* de audio de manera sencilla. Se ha elegido esta herramienta por ser la única en generar *mel-filterbanks* de manera automática, con buen rendimiento y de manera sencilla.
- Pydub es una librería de procesamiento de audio. Esta librería ofrecía funciones como la de obtener el umbral de ruido que no ofrecía otra librería de forma tan sencilla y por ello se ha elegido.
- Python Imaging Library o PIL ha sido la librería de procesamiento de imágenes seleccionada. Hay varias librerías que han podido utilizarse, entre ellas OpenCV. Debido a un error de compatibilidad con OpenCV y otras librerías, que era la librería que más conocía, se decidió cambiar a PIL para realizar la misma tarea.
- Jams es una librería de Python que permite cargar ficheros '.jams' que almacenan características de audio en JSON. Esta librería permite extraer estos datos de forma sencilla.

Capítulo 5

RESULTADOS

En este capítulo se reflejan los resultados obtenidos en las diferentes pruebas de clasificación realizadas. Se tendrá en cuenta el método de valoración de los resultados para medir cómo de bueno es el rendimiento de un modelo. A continuación, se procederá a describir los resultados de los diferentes conjuntos de datos siguiendo un orden cronológico de realización.

5.1. Conjunto de datos McGill Billboard Project

En primer lugar, se describirán los resultados de las mejores pruebas realizadas sobre el *dataset* de McGill. Debido a la falta de memoria RAM, se han cargado solamente en memoria 80 000 fragmentos aleatorios de 0.25 segundos de duración. Entre ellos, 64 000 pertenecen al conjunto de entrenamiento y 16 000 al de test. Todas las métricas mostradas se corresponden con las obtenidas con el conjunto de test. Dichas métricas serán: la exactitud, la media ponderada de la precisión, de la exhaustividad y del valor-f, el coeficiente de Cohen Kappa y la matriz de confusión.

Primero se han realizado los modelos de clasificadores básicos para ver qué rendimiento pueden obtener en este problema. Se probó inicialmente con Random Forest, kNN y regresión logística.

5.1.1. Random Forest

El clasificador básico de Random Forest ha obtenido los siguientes resultados:

	Exactitud	Precisión	Exhaustividad	Valor-F	Kappa
Random Forest	0.1916	0.28	0.19	0.15	0.1158

Tabla. 5.1: Tabla de resultados de Random Forest en el conjunto de datos de McGill

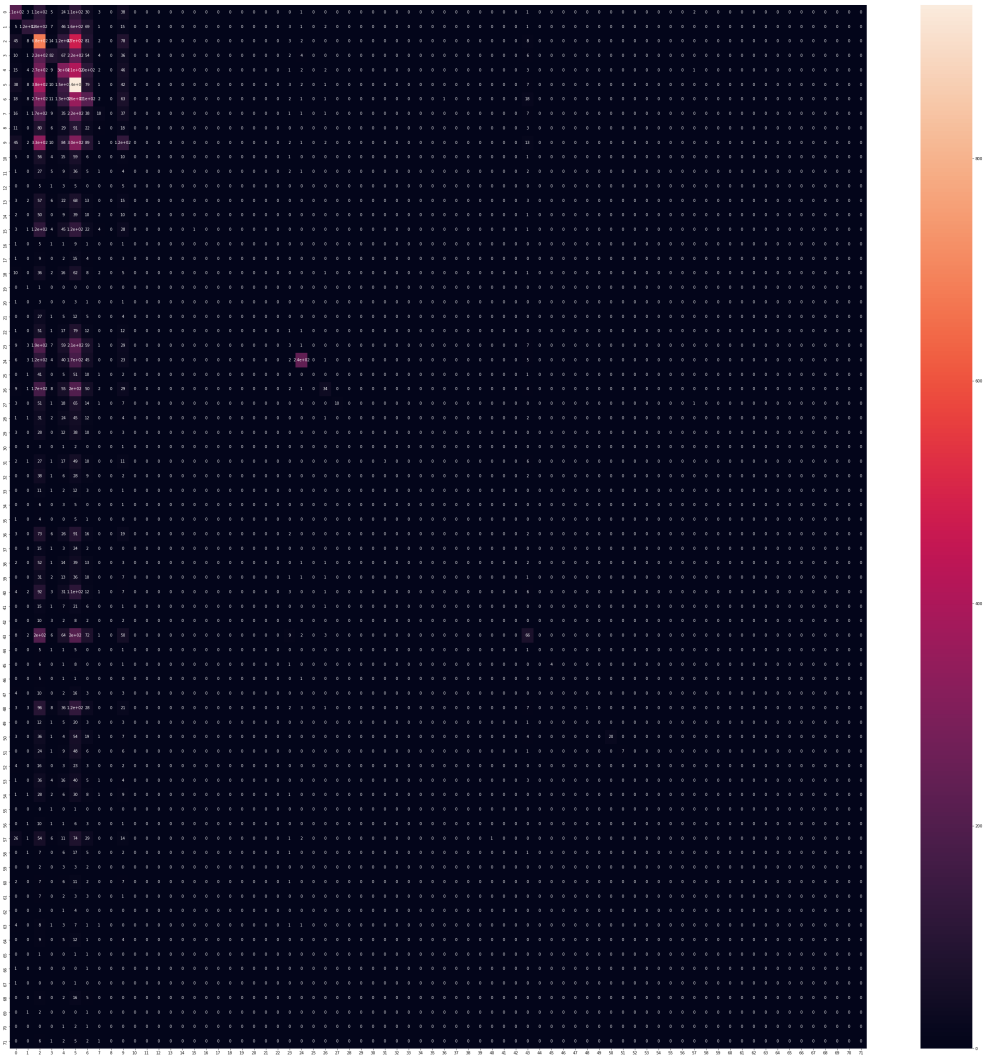

Figura 5.1: Matriz de confusión obtenida en el modelo de random forest sobre el conjunto de McGill

5.1.2. kNN

Con un vecindario de **15 vecinos**, el clasificador básico de kNN o k vecinos más próximos ha obtenido los siguientes resultados:

	Exactitud	Precisión	Exhaustividad	Valor-F	Kappa
kNN	0.1766	0.24	0.18	0.17	0.1286

Tabla. 5.2: Tabla de resultados de kNN en el conjunto de datos de McGill

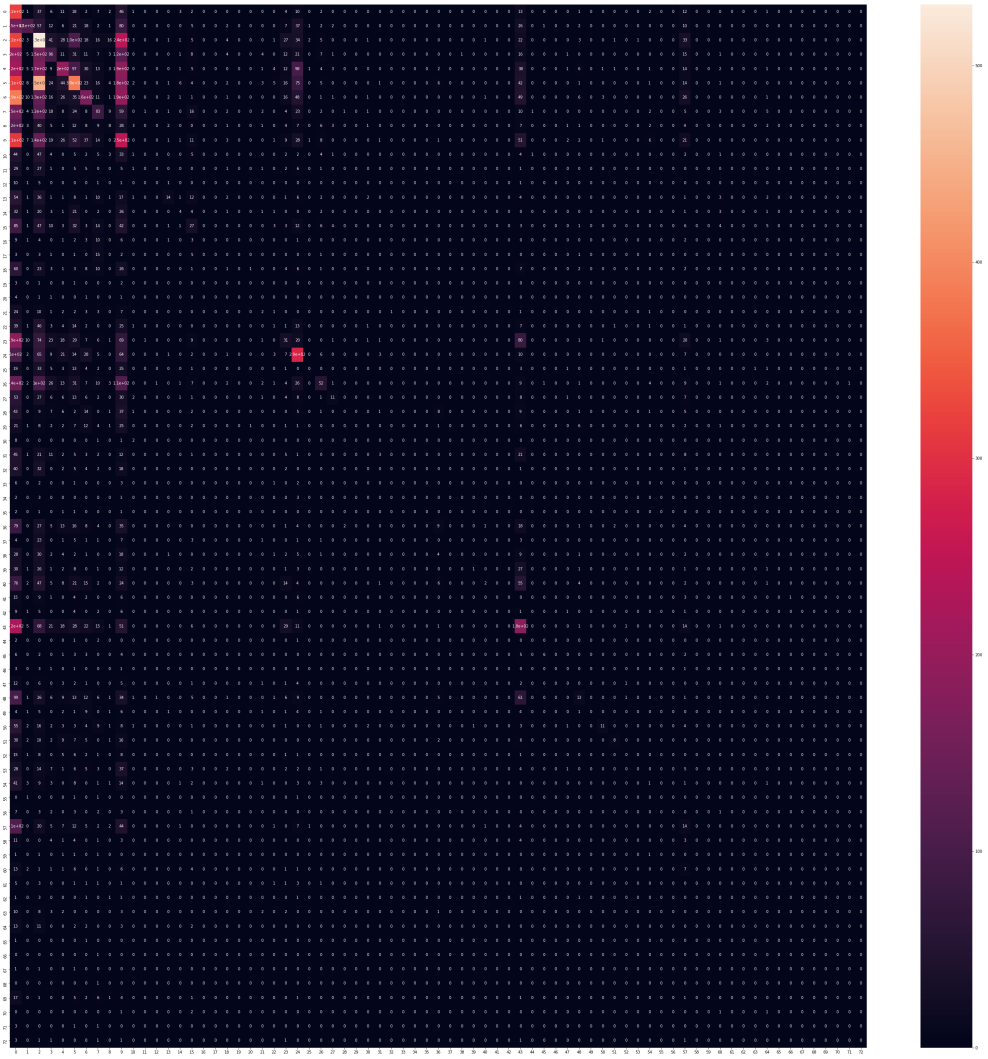

Figura 5.2: Matriz de confusión obtenida en el modelo de kNN sobre el conjunto de McGill

5.1.3. Regresión logística

Con un número de **100 iteraciones máximas**, el clasificador básico de regresión logística ha obtenido los siguientes resultados:

	Exactitud	Precisión	Exhaustividad	Valor-F	Kappa
Regresión logística	0.0867	0.09	0.09	0.09	0.0427

Tabla. 5.3: Tabla de resultados de regresión logística en el conjunto de datos de McGill

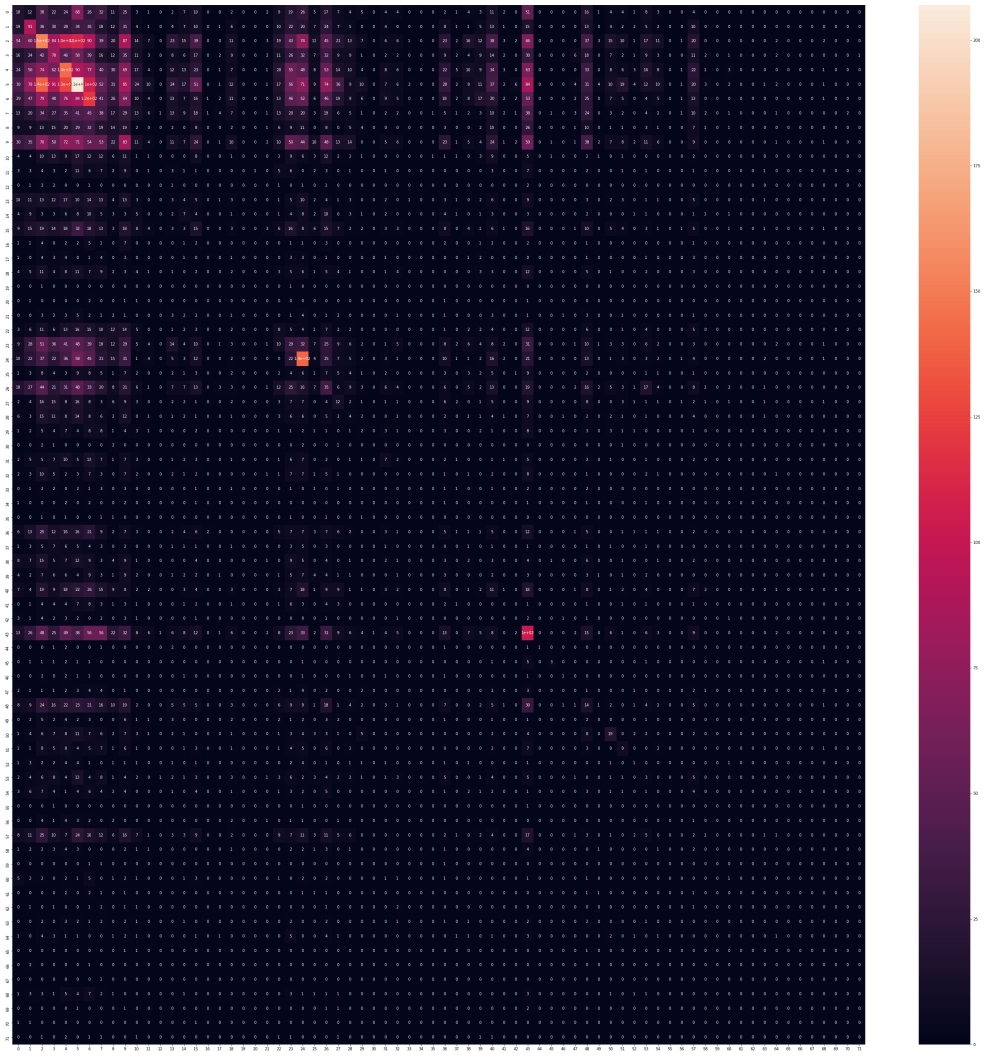

Figura 5.3: Matriz de confusión obtenida en el modelo de regresión logística sobre el conjunto de McGill

Ya obtenidos los resultados con los clasificadores básicos, se procede a probar con arquitecturas de redes neuronales básicas para así ver su rendimiento.

5.1.4. Perceptrón multicapa

Con un número de **50 epochs**, el modelo del perceptrón multicapa ha obtenido los siguientes resultados:

	Exactitud	Precisión	Exhaustividad	Valor-F	Kappa
Perceptrón multicapa	0.1045	0.01	0.10	0.02	0.00

Tabla. 5.4: Tabla de resultados del perceptrón multicapa en el conjunto de datos de McGill

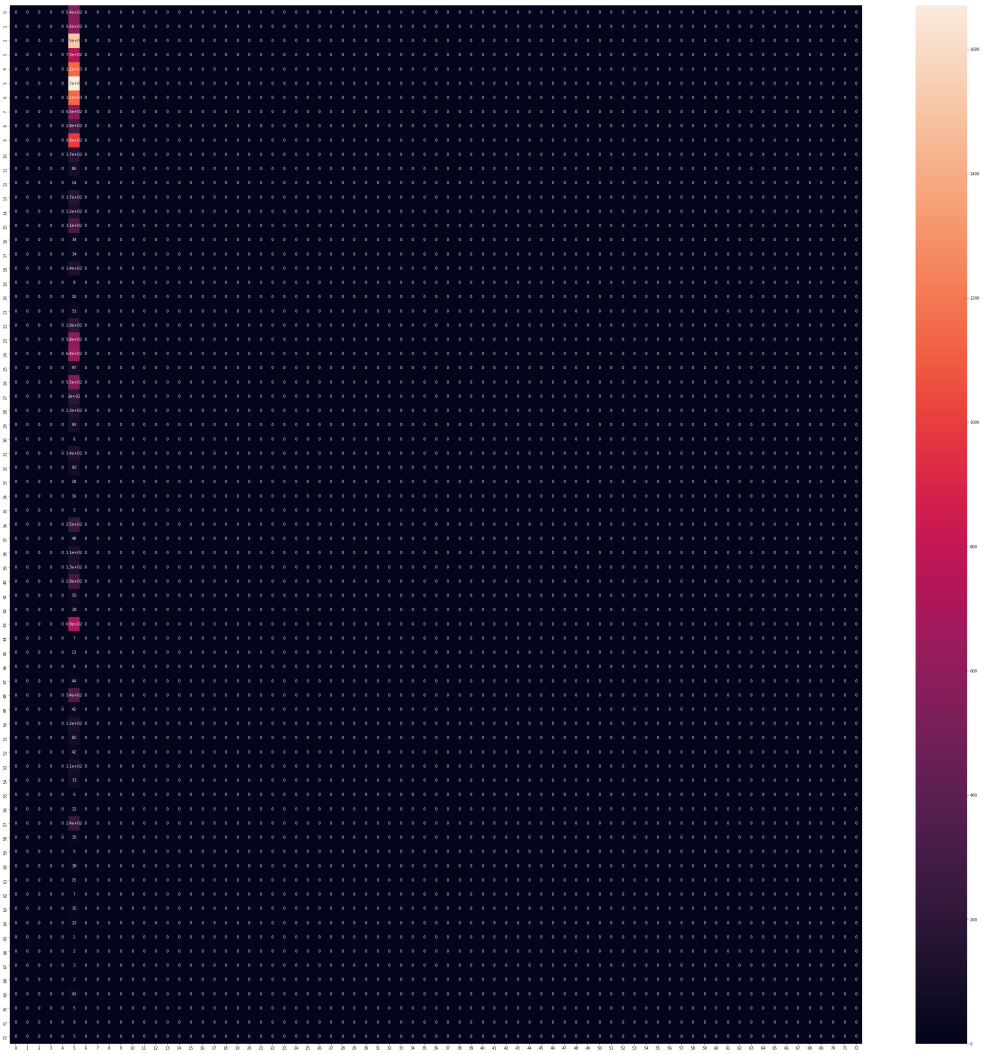

Figura 5.4: Matriz de confusión obtenida en el perceptrón multicapa sobre el conjunto de McGill

Debido a la obtención de un rendimiento tan malo, se ha decidido probar con alguna arquitectura más específica, como puede ser una red neuronal convolucional.

5.1.5. Red neuronal convolucional sobre audio

Este modelo de red neuronal convolucional trata directamente con las muestras de audio. Con un número de **30 epochs**, el modelo de red ha obtenido los siguientes resultados:

	Exactitud	Precisión	Exhaustividad	Valor-F	Kappa
CNN	0.1115	0.14	0.11	0.03	0.0098

Tabla. 5.5: Tabla de resultados de CNN en el conjunto de datos de McGill

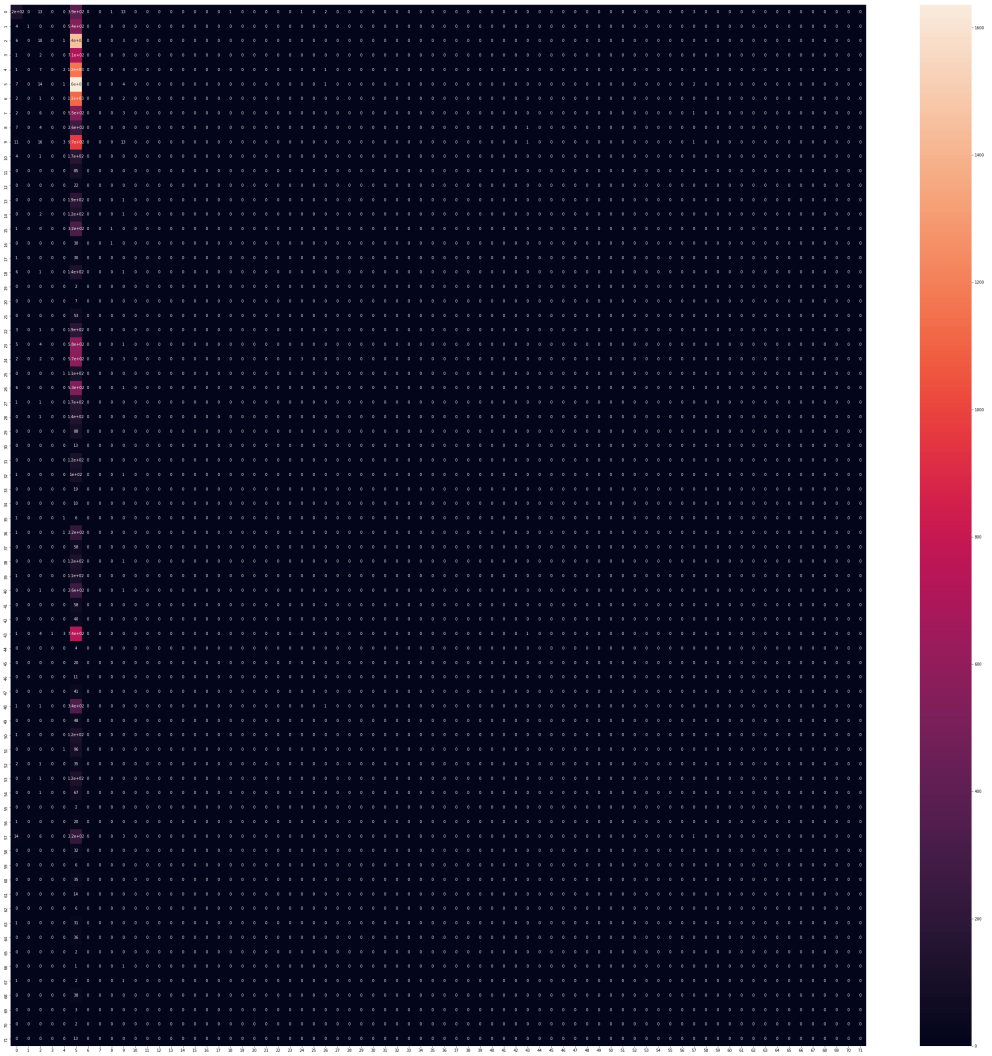

Figura 5.5: Matriz de confusión obtenida en el CNN sobre el conjunto de McGill

Se piensa que existe un problema con los datos y, al ver la mala puntuación que se obtiene, se abandona este conjunto de datos y se pasa a buscar otro.

5.2. Conjunto de datos GuitarSet

Ahora analizaremos los resultados de las mejores pruebas realizadas sobre el *dataset* de GuitarSet. En primer lugar, veremos los clasificadores básicos y posteriormente las redes neuronales. En esta ocasión se ha podido cargar en memoria RAM todo el *dataset*. Los resultados serán la exactitud, la media ponderada de la precisión, de la exhaustividad y del valor-f, el coeficiente de Cohen Kappa y la matriz de confusión.

Para comenzar, realizaremos lo mismo que en el conjunto de datos anterior y probaremos el rendimiento de clasificadores básicos como Random Forest, kNN o Regresión logística.

5.2.1. Random Forest

El clasificador básico de Random Forest se ha realizado sobre la configuración del conjunto de datos de 0.25 segundos por fragmento y 42 clases de acordes. Este clasificador ha obtenido los siguientes resultados:

	Exactitud	Precisión	Exhaustividad	Valor-F	Kappa
Random Forest	0.2849	0.28	0.28	0.23	0.2418

Tabla. 5.6: Tabla de resultados de Random Forest en el conjunto de datos de GuitarSet

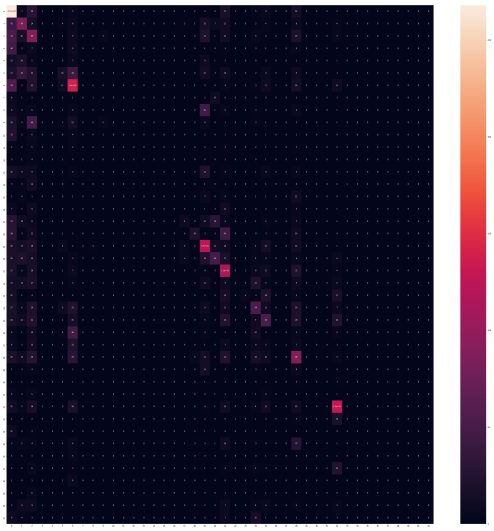

Figura 5.6: Matriz de confusión obtenida en el modelo de random forest sobre el conjunto de GuitarSet

5.2.2. kNN

El clasificador básico de kNN con **15 vecinos** se ha realizado sobre la configuración del conjunto de datos de 0.25 segundos por fragmento y 42 clases de acordes. La configuración adicional del modelo es:

	Exactitud	Precisión	Exhaustividad	Valor-F	Kappa
kNN	0.2670	0.24	0.27	0.24	0.2275

Tabla. 5.7: Tabla de resultados de kNN en el conjunto de datos de GuitarSet

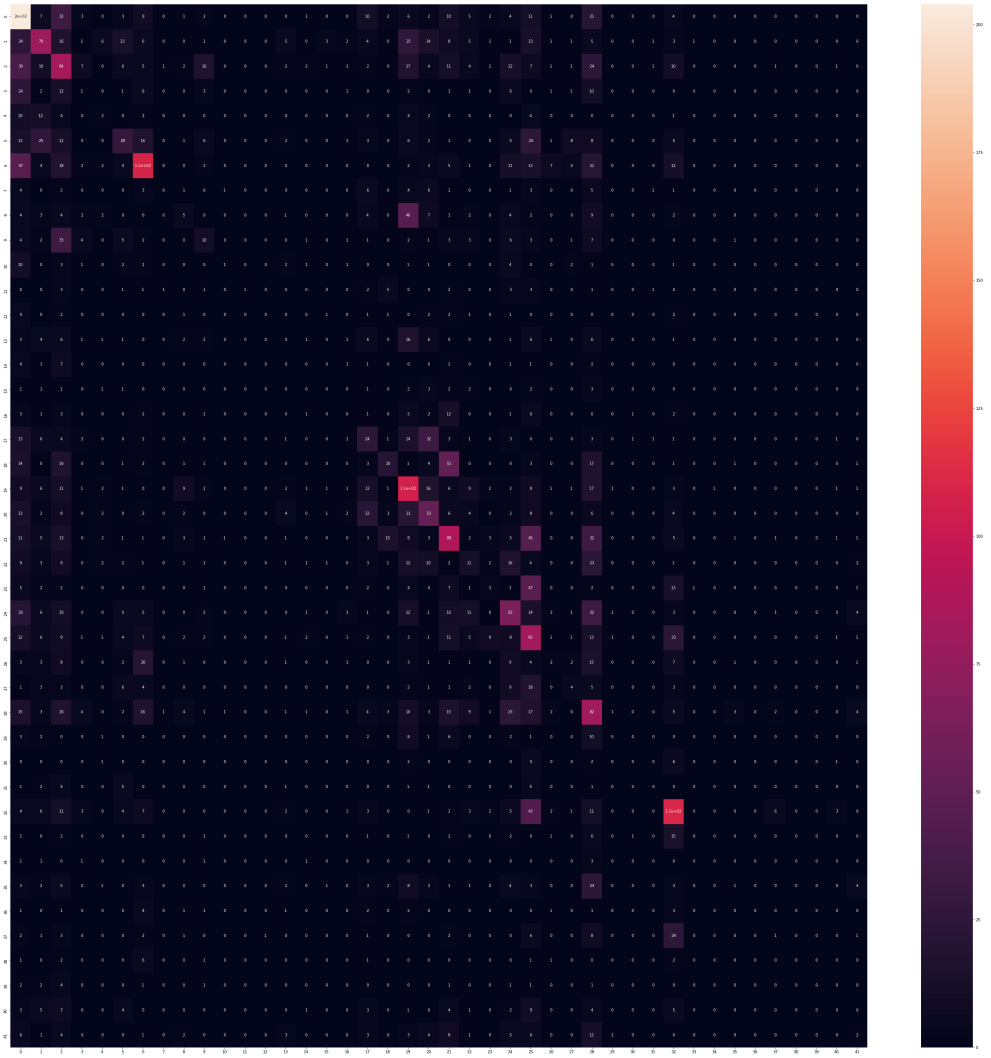

Figura 5.7: Matriz de confusión obtenida en el modelo de kNN sobre el conjunto de GuitarSet

5.2.3. Regresión logística

El clasificador básico de regresión logística con un número de **100 iteraciones máximas** se ha realizado sobre la configuración del conjunto de datos de 0.25 segundos por fragmento y 42 clases de acordes. La configuración adicional del modelo es:

	Exactitud	Precisión	Exhaustividad	Valor-F	Kappa
Regresión logística	0.0673	0.07	0.07	0.07	0.0264

Tabla. 5.8: Tabla de resultados de regresión logística en el conjunto de datos de GuitarSet

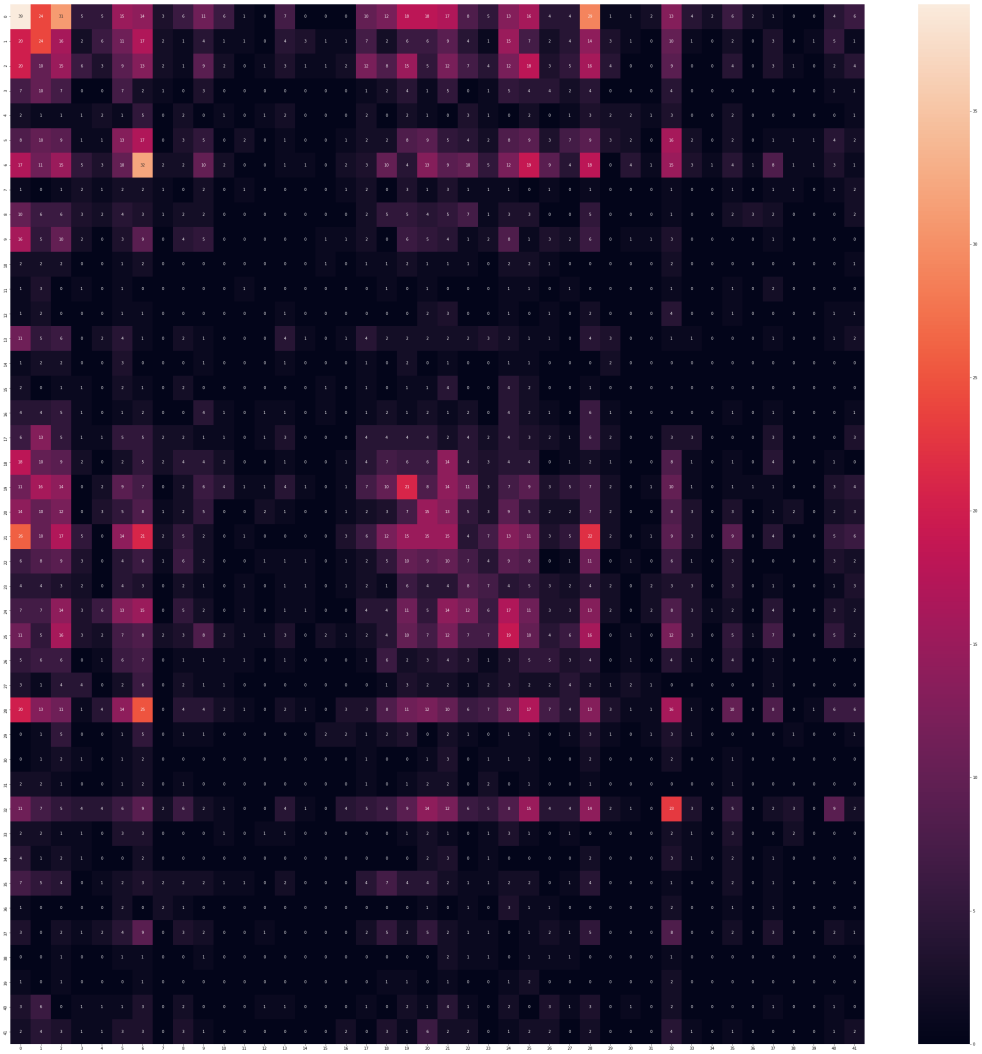

Figura 5.8: Matriz de confusión obtenida en el modelo de regresión logística sobre el conjunto de GuitarSet

Una vez probados los clasificadores básicos, se seguirá con arquitecturas de redes

neuronales básicas para comprobar su rendimiento.

5.2.4. Perceptrón multicapa

Con un número de **10 epochs**, el modelo del perceptrón multicapa con fragmentos de 0.25 segundos y 42 clases ha obtenido los siguientes resultados:

	Exactitud	Precisión	Exhaustividad	Valor-F	Kappa
Perceptrón multicapa	0.3201	0.29	0.32	0.29	0.28

Tabla. 5.9: Tabla de resultados del perceptrón multicapa en el conjunto de datos de GuitarSet

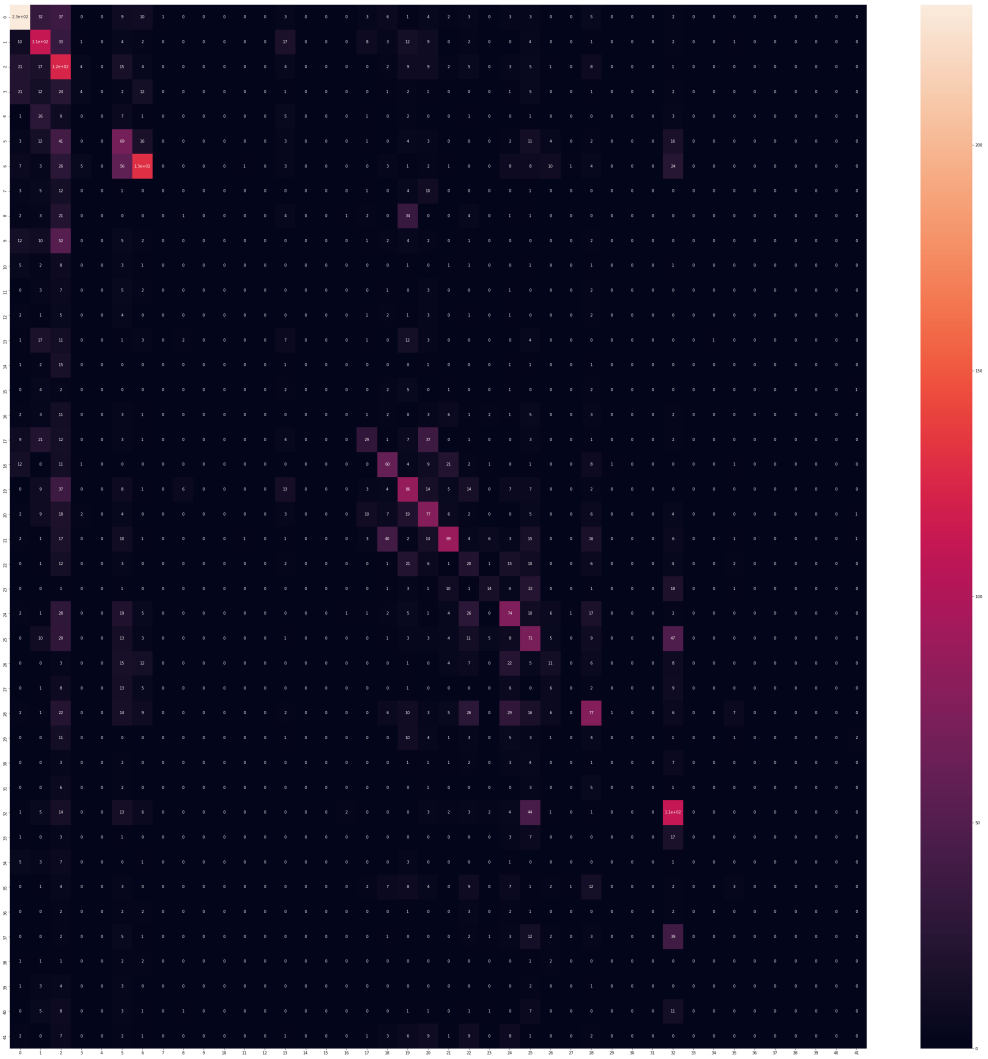

Figura 5.9: Matriz de confusión obtenida en el perceptrón multicapa sobre el conjunto de GuitarSet

Debido al bajo rendimiento de la red probamos una arquitectura un poco más específica como una red neuronal convolucional.

5.2.5. Red neuronal convolucional sobre audio

Con la misma arquitectura de red se entrenarán tres modelos con configuraciones del conjunto de datos distintas. Una con una duración de fragmento de audio de 0.25 segundos y 42 clases, otra con una duración de fragmento de audio de 1 segundo y 42 clases y una última configuración con una duración de fragmento de 1 segundo y 12 clases que distinguir. Todos estos modelos tratan directamente con las muestras de audio.

Fragmento de 0.25 segundos y 42 clases

Con un número de **30 epochs**, el modelo de red ha obtenido los siguientes resultados:

	Exactitud	Precisión	Exhaustividad	Valor-F	Kappa
Red convolucional	0.2238	0.26	0.22	0.22	0.19

Tabla. 5.10: Tabla de resultados de CNN, 0.25s y 42 acordes en el conjunto de datos de GuitarSet

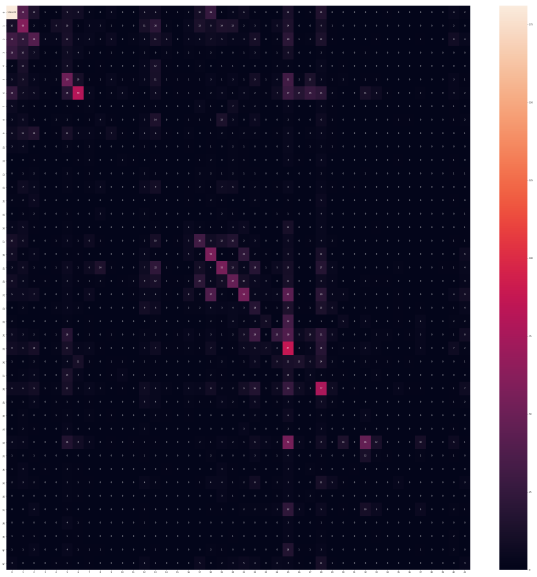

Figura 5.10: Matriz de confusión del CNN sobre audio con la configuración de 0.25 segundos y 42 clases de GuitarSet

Fragmento de 1 segundo y 42 clases

Con un número de **30 epochs**, el modelo de red ha obtenido los siguientes resultados:

	Exactitud	Precisión	Exhaustividad	Valor-F	Kappa
Red convolucional	0.0702	0.06	0.07	0.06	0.0222

Tabla. 5.11: Tabla de resultados de CNN, 1s y 42 acordes en el conjunto de datos de GuitarSet

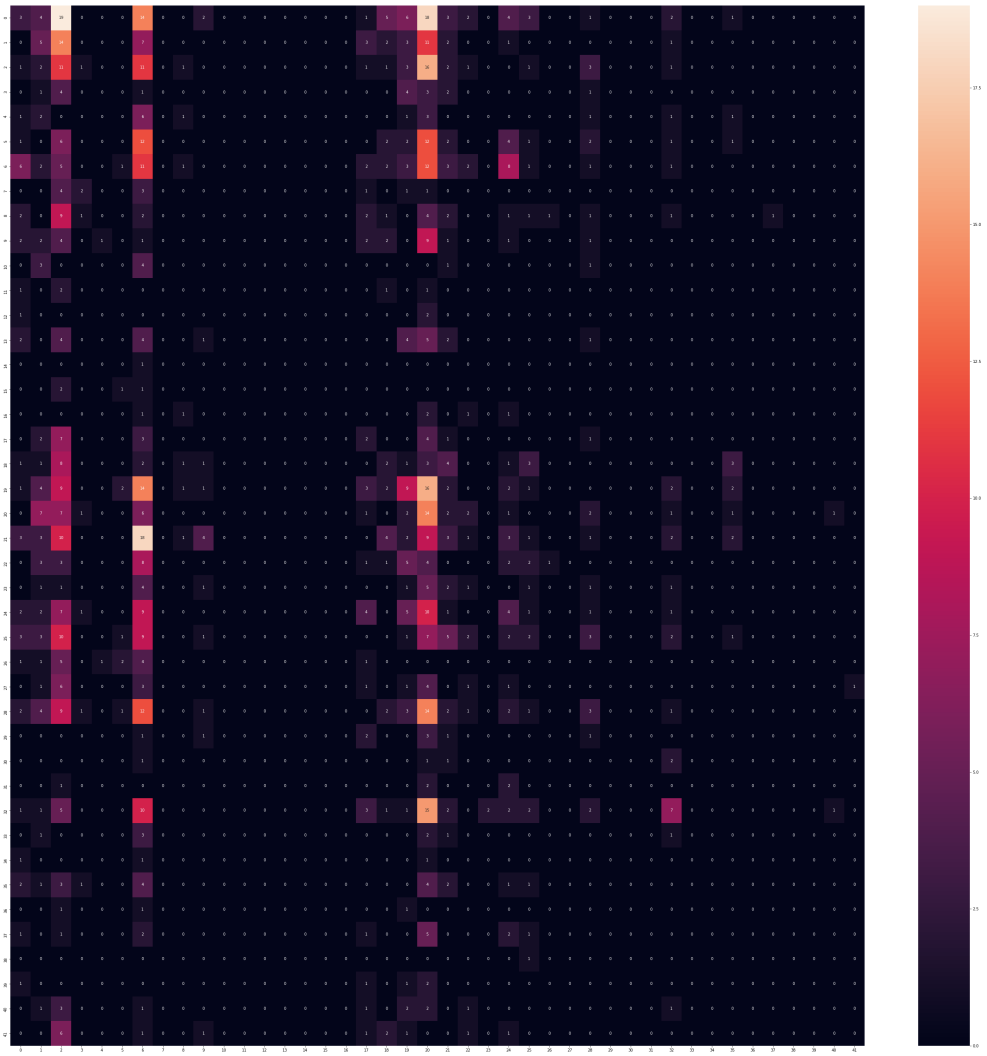

Figura 5.11: Matriz de confusión del CNN sobre audio con la configuración de 1 segundo y 42 clases de GuitarSet

Fragmento de 1 segundo y 12 clases

Con un número de **30 epochs**, el modelo de red ha obtenido los siguientes resultados:

	Exactitud	Precisión	Exhaustividad	Valor-F	Kappa
Red convolucional	0.1321	0.19	0.13	0.12	0.0429

Tabla. 5.12: Tabla de resultados de CNN, 1 s y 12 acordes en el conjunto de datos de GuitarSet

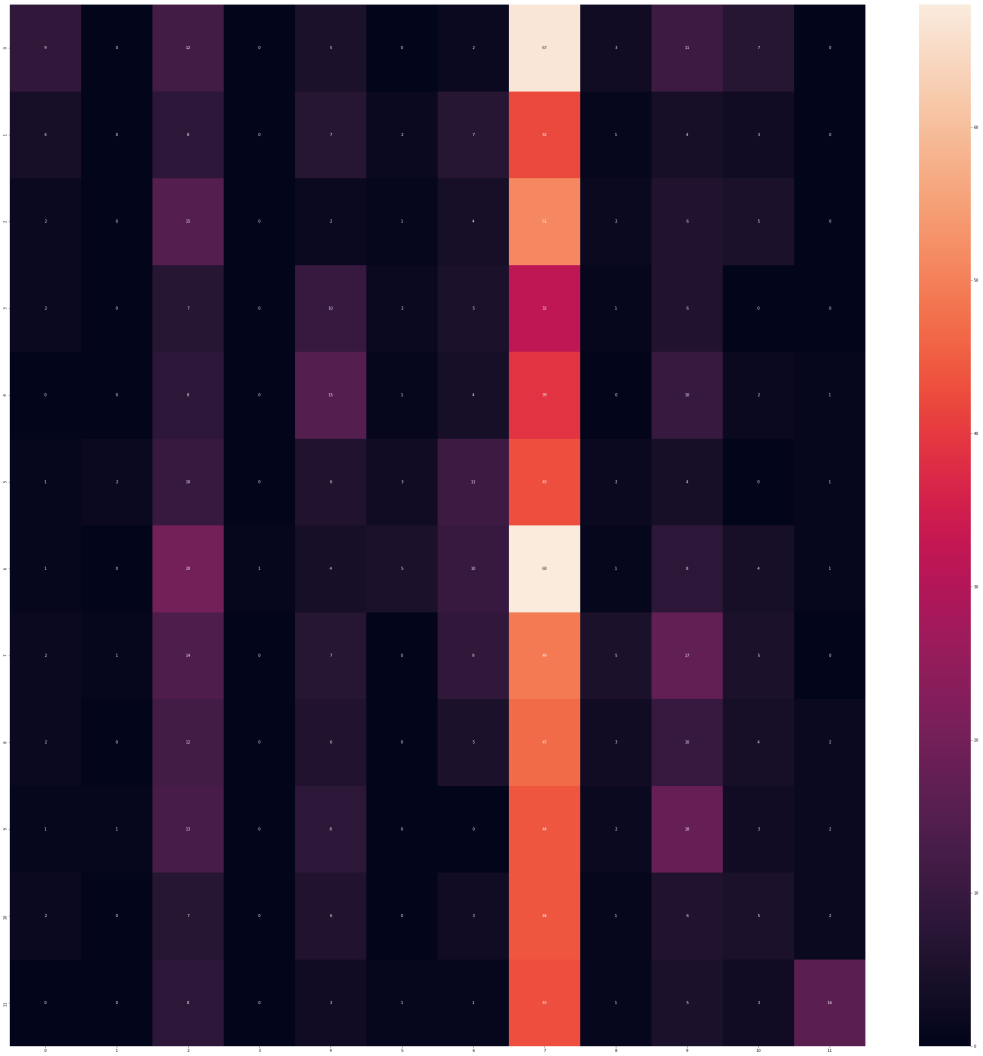

Figura 5.12: Matriz de confusión del CNN sobre audio con la configuración de 1 segundo y 12 clases de GuitarSet

5.2.6. Red neuronal convolucional sobre imágenes

Esta vez se entrenarán dos modelos con distintas configuraciones del conjunto de datos. Una con 0.25 segundos de duración de fragmento de audio y una segunda con 1 segundo. Ambas constan de 42 clases que distinguir. Todos estos modelos tratan directamente con las imágenes obtenidas con la herramienta LEAF.

Fragmento de 0.25 segundos y 42 clases

Con un número de **50 epochs**, el modelo de red ha obtenido los siguientes resultados:

	Exactitud	Precisión	Exhaustividad	Valor-F	Kappa
CNN LEAF	0.2391	0.21	0.24	0.20	0.1988

Tabla. 5.13: Tabla de resultados de CNN LEAF, 0.25 s y 42 acordes en el conjunto de datos de GuitarSet

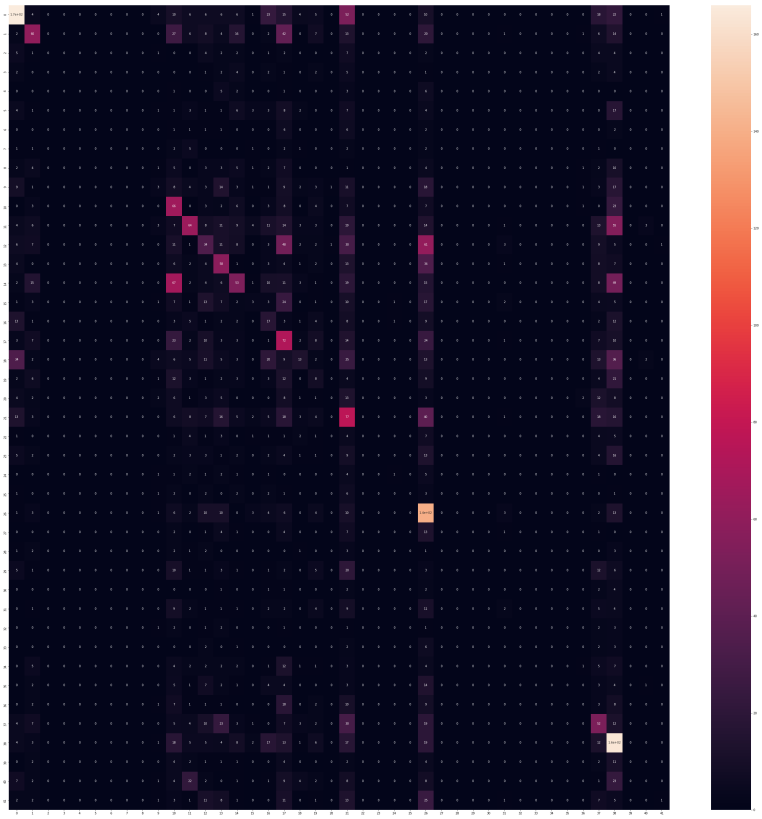

Figura 5.13: Matriz de confusión del CNN LEAF con la configuración de 0.25 segundo y 42 clases de GuitarSet

Fragmento de 1 segundo y 42 clases

Con un número de **10 epochs**, el modelo de red ha obtenido los siguientes resultados:

	Exactitud	Precisión	Exhaustividad	Valor-F	Kappa
CNN LEAF	0.8162	0.83	0.82	0.82	0.8078

Tabla. 5.14: Tabla de resultados de CNN LEAF, 1 s y 42 acordes en el conjunto de datos de GuitarSet

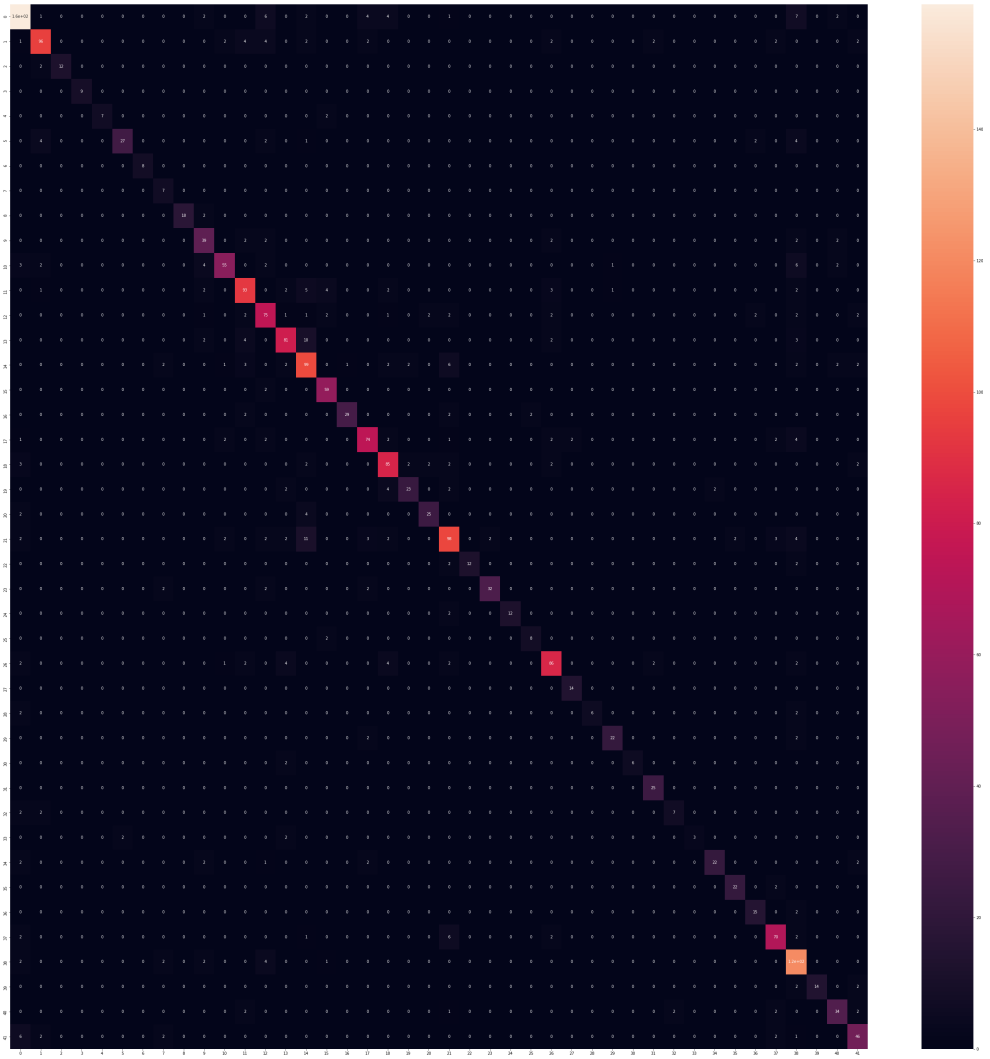

Figura 5.14: Matriz de confusión del CNN LEAF con la configuración de 1 segundo y 42 clases de GuitarSet

Una vez se obtuvo este rendimiento se consideró suficiente para dar por finalizada la sección de experimentación.

5.3. Comparación de los resultados

En esta sección se van a comparar todos los resultados obtenidos desde diferentes enfoques; en primer lugar, se van a comparar los del conjunto de datos de McGill con los obtenidos de GuitarSet y, posteriormente, se compararán los resultados entre las distintas configuraciones de GuitarSet.

5.3.1. Comparación entre conjuntos de datos

Se realizará una comparación entre los diferentes tipos de modelos comunes. Como Random Forest, kNN, regresión logística, perceptrón multicapa y CNN.

Random Forest

Random Forest	Exactitud	Precisión	Exhaustividad	Valor-F	Kappa
McGill	0.1916	0.28	0.19	0.15	0.1158
GuitarSet	0.2849	0.28	0.28	0.23	0.2418

Tabla. 5.15: Comparación de resultados obtenidos en Random Forest

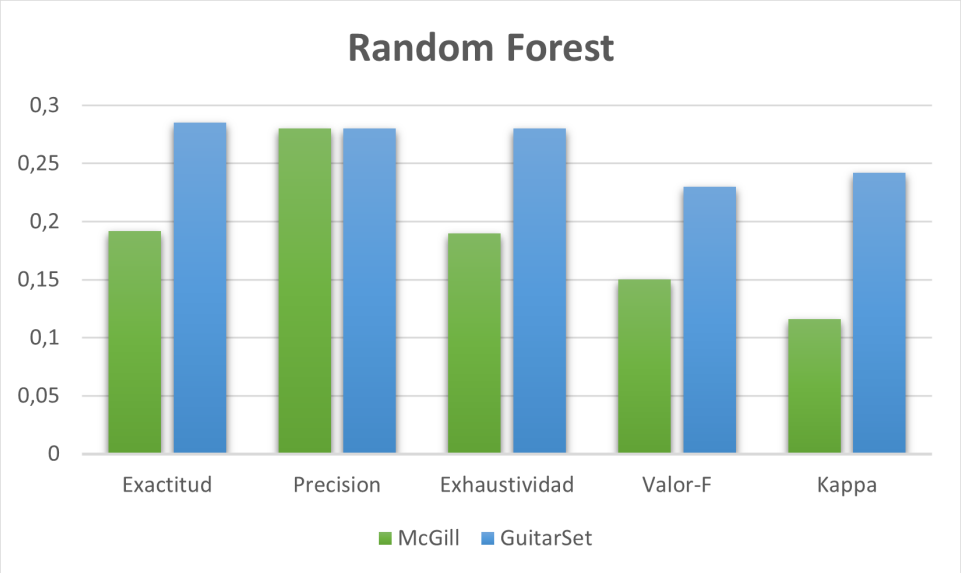

Figura 5.15: Gráfica comparativa de Random Forest entre los distintos conjuntos de datos

kNN

kNN	Exactitud	Precisión	Exhaustividad	Valor-F	Kappa
McGill	0.1766	0.24	0.18	0.17	0.1286
GuitarSet	0.2670	0.24	0.27	0.24	0.2275

Tabla. 5.16: Comparación de resultados obtenidos en kNN

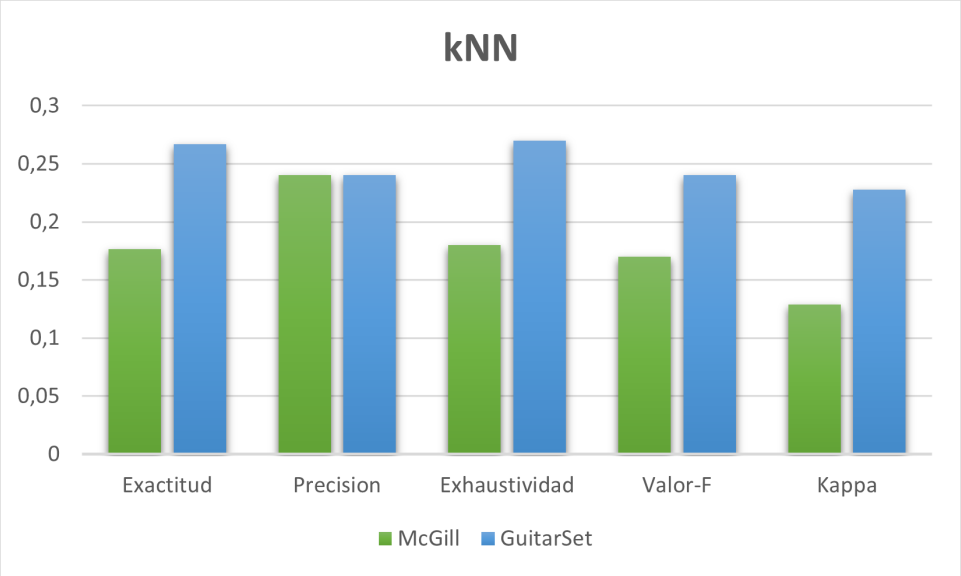

Figura 5.16: Gráfica comparativa de kNN entre los distintos conjuntos de datos

Regresión logística

Regresión Logística	Exactitud	Precisión	Exhaustividad	Valor-F	Kappa
McGill	0.0867	0.09	0.09	0.09	0.0427
GuitarSet	0.0673	0.07	0.07	0.07	0.0264

Tabla. 5.17: Comparación de resultados obtenidos en regresión logística

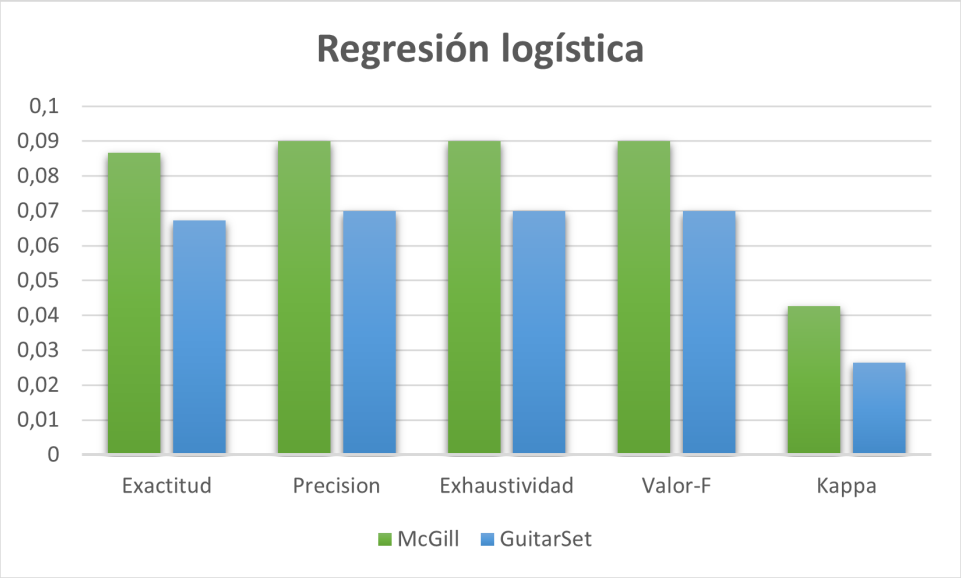

Figura 5.17: Gráfica comparativa de regresión logística entre los distintos conjuntos de datos

Perceptrón multicapa

MLP	Exactitud	Precisión	Exhaustividad	Valor-F	Kappa
McGill	0.1045	0.01	0.10	0.02	0.00
GuitarSet	0.3201	0.29	0.32	0.29	0.28

Tabla. 5.18: Comparación de resultados obtenidos en el perceptrón multicapa

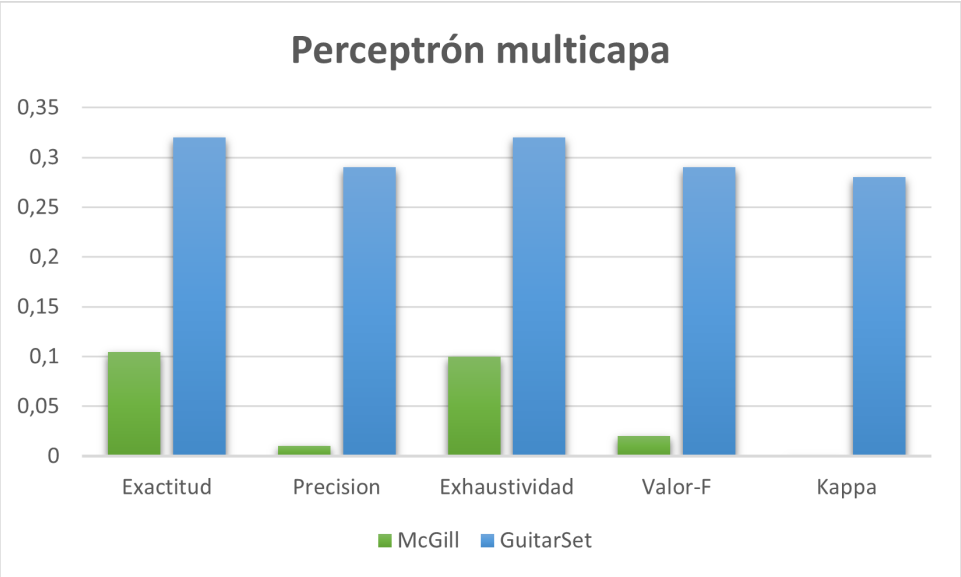

Figura 5.18: Gráfica comparativa del perceptrón multicapa entre los distintos conjuntos de datos

Red neuronal convolucional

CNN	Exactitud	Precisión	Exhaustividad	Valor-F	Kappa
McGill	0.1115	0.14	0.11	0.03	0.0098
GuitarSet	0.2238	0.26	0.22	0.22	0.19

Tabla. 5.19: Comparación de resultados obtenidos en la red neuronal convolucional

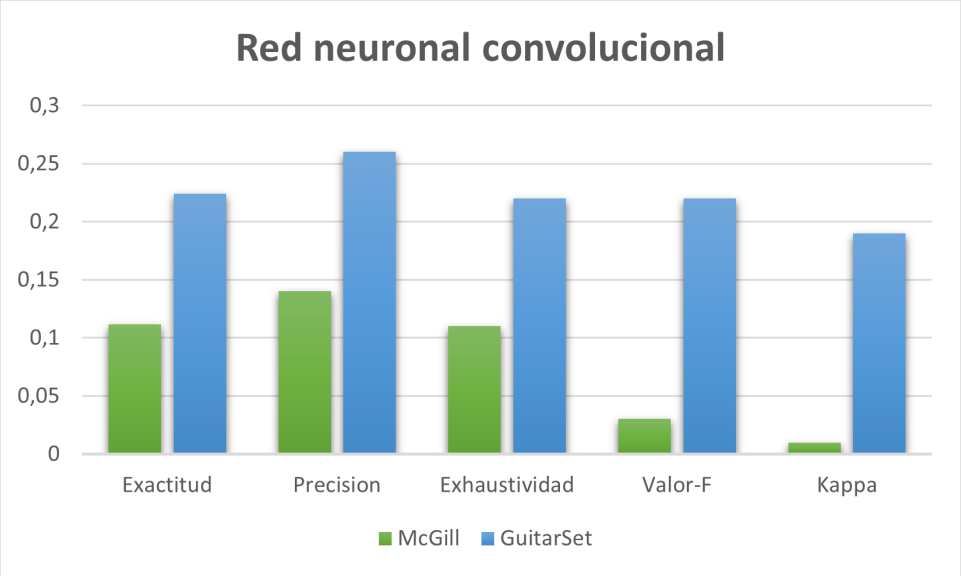

Figura 5.19: Gráfica comparativa de la red neuronal convolucional entre los distintos conjuntos de datos

Como se ha podido observar, los clasificadores básicos obtienen un rendimiento muy parecido. En cambio, al realizar modelos de clasificación basados en aprendizaje profundo, el conjunto de datos GuitarSet obtiene un mayor rendimiento que McGill. Esto se puede deber al gran número de clases que se han de distinguir en el conjunto de McGill o a un fallo en los datos, como se comentó en el punto 4.5.2.

5.3.2. Comparación entre configuraciones de GuitarSet

En este apartado se realizará una comparación entre las diferentes configuraciones de GuitarSet teniendo en cuenta la duración del fragmento de audio y el número de clases que se distinguen. Para realizar la comparación se tendrán en cuenta los modelos CNN sobre audio y sobre imágenes.

CNN

CNN	Exactitud	Precisión	Exhaustividad	Valor-F	Kappa
0.25s - 42 clases	0.2238	0.26	0.22	0.22	0.19
1s - 42 clases	0.0702	0.06	0.07	0.06	0.0222
1s - 12 clases	0.1321	0.19	0.13	0.12	0.0429

Tabla. 5.20: Tabla comparativa de configuraciones de GuitarSet en CNN sobre audio

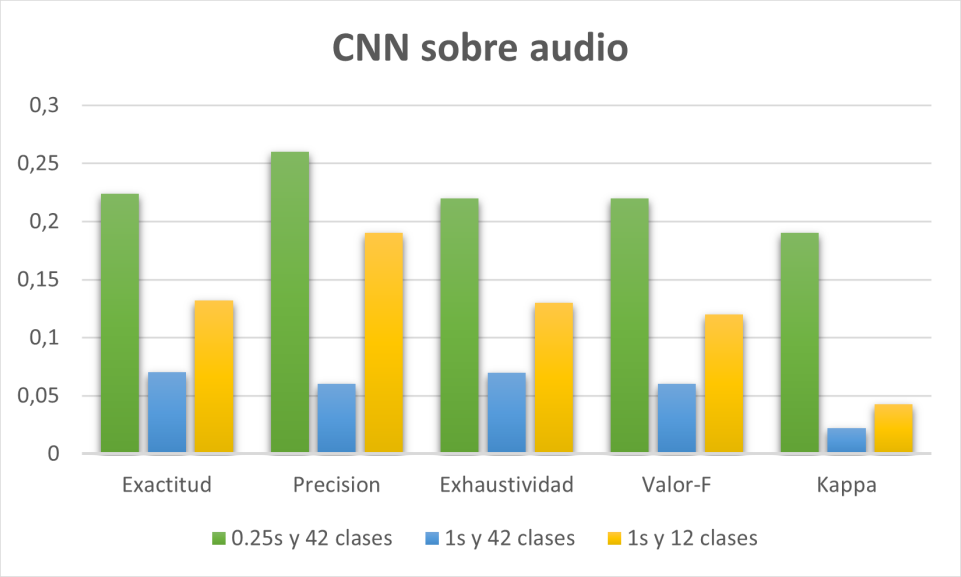

Figura 5.20: Gráfica comparativa del rendimiento de una red neuronal convolucional en función de las configuraciones del conjunto de datos

CNN LEAF

CNN LEAF	Exactitud	Precisión	Exhaustividad	Valor-F	Kappa
0.25s - 42 clases	0.2391	0.21	0.24	0.20	0.1988
1s - 42 clases	0.8162	0.83	0.82	0.82	0.8078

Tabla. 5.21: Tabla comparativa de configuraciones de GuitarSet en CNN LEAF

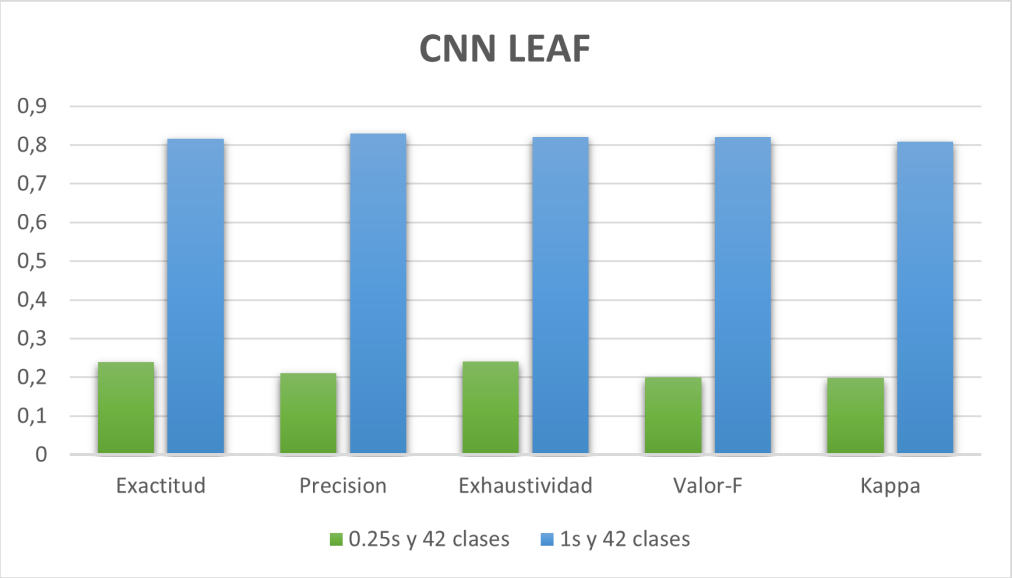

Figura 5.21: Gráfica comparativa del rendimiento de una red neuronal convolucional sobre imágenes dependiendo de las configuraciones del conjunto de datos

Comprobamos que, entre los modelos de red neuronal convolucional sobre las muestras de audio, el modelo que trabaja con una duración de fragmento de audio de 0.25 segundos obtiene un rendimiento claramente superior. Esto se debe a que tiene que aprender a extraer los acordes en cuatro veces menos variables que con la duración de fragmento igual a 1 segundo. Por otra parte, al comparar el modelo de 42 clases y de 12, obtiene mejor rendimiento el de 12 por comprimir el número de clases y así facilitar la clasificación.

Entre los modelos que tratan con imágenes obtenidas por LEAF, es claramente superior el de 1 segundo de duración. Se observa que, dividiendo los fragmentos en 0.25 segundos, existen muchos fragmentos donde no se puede distinguir si está sonando un acorde u otro. El hecho de dividirlo en 1 segundo no repercute negativamente como en los modelos anteriores debido a que trata directamente con imágenes y no con variables.

5.3.3. Comparación entre los mejores modelos

En esta última comparación medimos el rendimiento obtenido por las mejores configuraciones de cada tipo de modelo utilizando el conjunto de datos de GuitarSet. En la tabla 5.22 se muestran los diferentes modelos ordenados de peor a mejor rendimiento.

	Exactitud	Precisión	Exhaustividad	Valor-F	Kappa
Regresión logística	0.0673	0.07	0.07	0.07	0.0264
CNN	0.2238	0.26	0.22	0.22	0.19
kNN	0.2670	0.24	0.27	0.24	0.2275
Random Forest	0.2849	0.28	0.28	0.23	0.2418
MLP	0.3201	0.29	0.32	0.29	0.28
CNN LEAF	0.8162	0.83	0.82	0.82	0.8078

Tabla. 5.22: Tabla comparativa de los mejores modelos sobre el conjunto de datos de GuitarSet

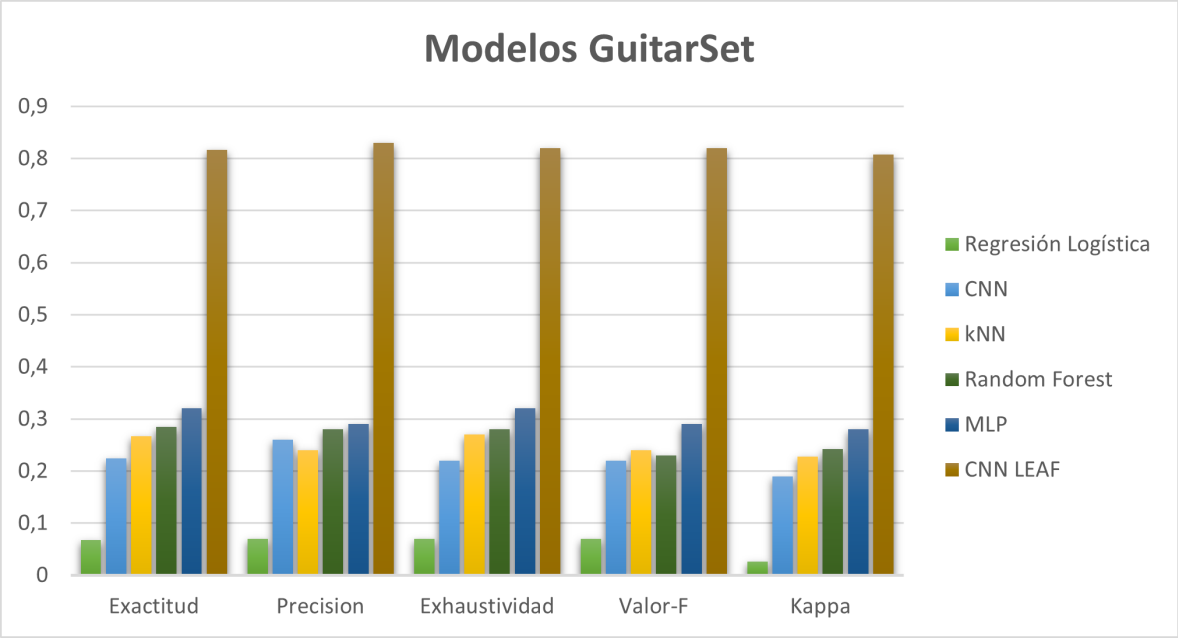

Figura 5.22: Gráfica comparativa entre los diferentes modelos sobre GuitarSet

A excepción de la regresión logística, que obtiene un rendimiento muy malo, los modelos que son entrenados con las muestras de audio directamente tienen un rendimiento bajo y muy similar. En cambio, gracias a la representación del audio en imágenes, el modelo de red neuronal convolucional es capaz de aprender a clasificar los 42 acordes propuestos con una tasa de acierto razonablemente alta. Esto lo convierte en el mejor modelo obtenido.

Capítulo 6

CONCLUSIONES

En este capítulo final se va a describir como se ha distribuido la realización de este trabajo de fin de grado, exponer una visión general del trabajo realizado, señalar los resultados obtenidos y proponer cómo podría continuarse este trabajo en el futuro.

6.1. Distribución del trabajo realizado

El trabajo ha sido realizado en un periodo de tiempo de unos 10 meses. En este tiempo se han realizado diferentes fases del trabajo. La **primera fase** consistió en un estudio del estado del arte realizando una búsqueda de la bibliografía acerca del tema de estudio y una toma de contacto con el aprendizaje profundo donde se estudiaron las diferentes redes neuronales y su funcionamiento.

En la **segunda fase** se realizó la búsqueda, obtención y análisis preliminar de tres conjuntos de datos, de los cuales solo se trabajó con dos que cumplían las características requeridas. Aquí también se preprocesaron los conjuntos de datos y se realizó otro análisis exploratorio una vez obtenidos los conjuntos finales de datos.

La **tercera fase** consistió en la experimentación con modelos de aprendizaje automático. Aquí se hicieron las pruebas con los clasificadores básicos Random Forest, kNN y regresión logística y con las redes neuronales profundas, como un perceptrón multicapa y una red neuronal convolucional sobre audio, y otra sobre imágenes obtenidas gracias a la herramienta LEAF.

La **cuarta y última fase** consistió en la obtención de resultados y redacción de la memoria final. En ella se obtuvo como mejor modelo la red neuronal convolucio-

nal sobre imágenes capaz de distinguir entre 42 clases de acordes con un 80 % de exactitud.

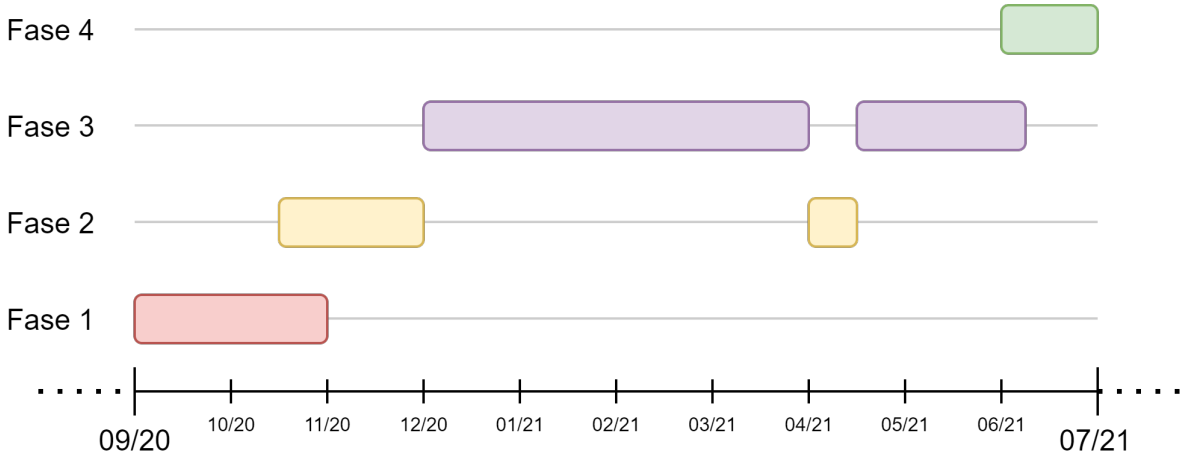

Figura 6.1: Cronograma de las fases del trabajo

6.2. Lecciones aprendidas

En este trabajo he aprendido acerca de múltiples ámbitos como pueden ser el aprendizaje automático y el funcionamiento de las redes neuronales, especialmente las redes neuronales convolucionales. También he aprendido a solucionar un problema mediante ciencia de datos realizando la obtención de los datos, su análisis exploratorio, su preprocesamiento y a aplicar técnicas de aprendizaje profundo sobre dichos datos.

Uno de los mayores retos ha sido encontrar un modelo que tenga un buen rendimiento, ya que se han tenido que probar muchos de ellos cambiando pequeños detalles hasta encontrar uno funcional. Este proceso ha consistido en probar diferentes técnicas de aprendizaje automático sobre los diferentes conjuntos de datos con los que se ha tratado y en evaluar dichos modelos para conocer los fallos de cada uno e intentarlo con otra técnica o configuración para conseguir mejores resultados. Se ha probado con tres clasificadores básicos: random forest, kNN y regresión logística, y con dos técnicas de aprendizaje profundo: MLP y CNN. Tras ver que los resultados obtenidos al tratar con las muestras de audio eran malos, se decidió probar una CNN sobre una representación en imagen del audio, *mel-filterbanks*. Tras varias pruebas se obtuvo un rendimiento razonablemente bueno y se dio por terminada la experimentación.

Otro gran reto ha sido redactar esta memoria, ya que nunca antes había escrito un trabajo con enfoque científico y ha sido realizada en LaTeX, sistema de composición de textos que no había utilizado anteriormente.

En resumen, me ha parecido una gran experiencia realizar este trabajo de fin de grado, ya que considero que me ha enriquecido mucho en diferentes ámbitos.

6.3. Trabajo futuro

Por último, me gustaría plantear de cara al futuro diferentes formas de continuar este trabajo.

6.3.1. Mejora del modelo

Una de ellas sería una posible forma de mejorar los resultados actuales utilizando modelos autoconfigurables. Estos modelos, como los presentados por la biblioteca de AutoKeras, son capaces de encontrar una arquitectura óptima en el mismo entrenamiento. De esta forma, posiblemente encuentre una mejor arquitectura de red neuronal que la presentada anteriormente y sea capaz de mejorar el rendimiento del modelo.

6.3.2. Desarrollo de un programa

También sería interesante realizar una implementación de la red neuronal final en un programa para poder introducir audios diferentes a los del conjunto de datos y que devuelva los acordes encontrados de una forma bastante visual e intuitiva. Este programa podría hacerse en forma de página web, aplicación móvil o aplicación de escritorio y le aportaría utilidad a la red entrenada.

Bibliografía

N. Boulanger-Lewandowski, Y. Bengio, and P. Vincent. Audio chord recognition with recurrent neural networks. *Proceedings of the 14th International Society for Music Information Retrieval Conference, ISMIR 2013*, pages 335–340, 2013.

L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, oct 2001. ISSN 08856125. doi: 10.1023/A:1010933404324. URL https://link.springer.com/article/10.1023/A:1010933404324.

A. Burgoyne. The McGill Billboard Project · DDMAL, 2021. URL https://ddmal.music.mcgill.ca/research/The_McGill_Billboard_Project_(Chord_Analysis_Dataset)/.

F. Charte, A. Rivera-Rivas, F. Martínez, and M. J. del Jesus. Evoaaa: An evolutionary methodology for automated neural autoencoder architecture search. *Integrated Computer-Aided Engineering*, 27(3):211–231, 05/2020 2020. doi: https://doi.org/10.3233/ICA-200619.

Clases de guitarra Lima. Notación musical para acordes, 2021. URL https://clasesdeguitarralima.com/2016/03/12/notacion-musical-para-acordes/.

W. contributors. Random forest - Wikipedia, 2021. URL https://en.wikipedia.org/wiki/Random_forest. [Online; accessed 27/06/2021].

J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009. doi: 10.1109/CVPR.2009.5206848.

T. Fujishima. Realtime chord recognition of musical sound: a system using common lisp music. In *ICMC*, 1999.

K. Fukushima. Neocognitron. *Scholarpedia*, 2(1):1717, Jan. 2007. doi: 10.4249/scholarpedia.1717.

S. Haykin. *Neural networks*. Prentice Hall [1999], 1999.

Haytham Fayek. Speech Processing for Machine Learning: Filter banks, Mel-Frequency Cepstral Coefficients (MFCCs) and What's In-Between | Haytham Fayek, 2016. URL https://haythamfayek.com/2016/04/21/speech-processing-for-machine-learning.htmlhttps://haythamfayek.com/2016/04/21/speech-processing-for-machine-learning.html%0Ahttps://haythamfayek.com/2016/04/21/speech-processing-for-machine-learning.html#fn:2%0Ahttps://haythamfayek.com/2016/04/21/speech-processing-for-machine-learning.html.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2016-Decem, pages 770–778. IEEE Computer Society, dec 2016. ISBN 9781467388504. doi: 10.1109/CVPR.2016.90. URL http://image-net.org/challenges/LSVRC/2015/.

H. Jin, Q. Song, and X. Hu. Auto-keras: An efficient neural architecture search system. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1946–1956. Association for Computing Machinery, jun 2019. ISBN 9781450362016. doi: 10.1145/3292500.3330648. URL http://arxiv.org/abs/1806.10282.

A. Kaplan and M. Haenlein. Siri, Siri, in my hand: Who's the fairest in the land? On the interpretations, illustrations, and implications of artificial intelligence, jan 2019. ISSN 00076813.

S. Kiranyaz, O. Avci, O. Abdeljaber, T. Ince, M. Gabbouj, and D. J. Inman. 1d convolutional neural networks and applications: A survey. *Mechanical Systems and Signal Processing*, 151:107398, 2021. ISSN 0888-3270. doi: https://doi.org/10.1016/j.ymssp.2020.107398. URL https://www.sciencedirect.com/science/article/pii/S0888327020307846.

Y. A. LeCun, L. Bottou, G. B. Orr, and K. R. Müller. Efficient backprop. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7700 LECTU:9–48, 2012. ISSN 16113349. doi: 10.1007/978-3-642-35289-8_3.

Machine Learning Glossary. Activation Functions — ML Glossary documentation, 2021. URL https://ml-cheatsheet.readthedocs.io/en/latest/activation_functions.html.

J. Osmalskyj, J. J. Embrechts, and M. V. Droogenbroeck. Neural networks for musical chords recognition. *Osmalskyj, J., Embrechts, J. J., & Droogenbroeck, M. Van.*, 1 (January):46, 2012.

T. Pandit, A. Kapoor, R. Shah, and R. Bhuva. Understanding inception network architecture for image classification, 02 2020.

W. Piston and M. DeVoto. *Harmony*. Gollancz, 1994.

J. R. Quinlan. Learning decision tree classifiers. *ACM Comput. Surv.*, 28(1):71–72, Mar. 1996. ISSN 0360-0300. doi: 10.1145/234313.234346. URL https://doi.org/10.1145/234313.234346.

H. Robbins. A stochastic approximation method. *Annals of Mathematical Statistics*, 22:400–407, 2007.

S. J. Russell. *Artificial intelligence*. Prentice Hall, 1995.

J. Schmidhuber. Deep Learning in neural networks: An overview, jan 2015. ISSN 18792782.

A. Schönberg and R. Barce. *Tratado de armonía*. Real Musical, 2004.

A. Sherstinsky. Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) network. *Physica D: Nonlinear Phenomena*, 404, aug 2020. ISSN 01672789. doi: 10.1016/j.physd.2019.132306. URL http://arxiv.org/abs/1808.03314http://dx.doi.org/10.1016/j.physd.2019.132306.

A. M. G. Vico, P. G. García, and C. J. C. del Jesus. Modelos descriptivos basados en aprendizaje supervisado para el tratamiento de grandes volúmenes de datos y flujos continuos de datos. Technical report, 2018.

Wikipedia. k vecinos más próximos - Wikipedia, la enciclopedia libre, 2020. URL https://es.wikipedia.org/wiki/K_vecinos_m{á}s_pr{ó}ximos. [Internet; descargado 27-junio-2021].

Wikipedia. Perceptrón multicapa - Wikipedia, la enciclopedia libre, 2021. URL https://es.wikipedia.org/wiki/Perceptr{ó}n_multicapa. [Internet; descargado 27-junio-2021].

Q. Xi, R. M. Bittner, J. Pauwels, X. Ye, and J. P. Bello. Guitarset: A dataset for guitar transcription. In *Proceedings of the 19th International Society for Music Information Retrieval Conference, ISMIR 2018*, pages 453–460, 2018. ISBN 9782954035123. URL https://github.com/marl/GuitarSet.

Q. Yang, Y. Zhang, W. Dai, and S. J. Pan. *Transfer Learning*. Cambridge University Press, 2020. doi: 10.1017/9781139061773.

YouTube. La escala cromática explicada en 7 minutos - Tonos y semitonos - YouTube, 2013. URL https://www.youtube.com/watch?v=IbgOcXar9UA.

J. Zamacois. *Tratado de armonía*. SpanPress, 1997.

N. Zeghidour, O. Teboul, F. de Chaumont Quitry, and M. Tagliasacchi. Leaf: A learnable frontend for audio classification. *ICLR*, 2021.