Universidad Politécnica
de Madrid

Escuela Técnica Superior de
Ingenieros Informáticos

Grado en Ingeniería Informática

Trabajo Fin de Grado

Modelo de Red Neuronal Convolucional para la Generación de Música "Lo-Fi"

Autor: Antonio Carpintero Castilla

Tutor(a): Jose María Barambones Ramírez

Madrid, junio de 2023

Este Trabajo Fin de Grado se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

Trabajo Fin de Grado
Grado en Ingeniería Informática
Título: Modelo de Red Neuronal Convolucional para la Generación de Música "Lo-Fi"

junio de 2023
Autor: Antonio Carpintero Castilla

Tutor:

Jose María Barambones Ramírez
Lenguajes y Sistemas Informáticos e Ingeniería de Software
ETSI Informáticos
Universidad Politécnica de Madrid

Resumen

El presente documento recoge toda la información generada en el diseño, desarrollo y pruebas de un modelo de inteligencia artificial capaz de generar canciones del género "*Lo-Fi*". Este modelo, consta de tres partes claramente diferenciadas:

1. Preprocesado: Segmento donde se transforman los archivos de formato MIDI del set de entrenamiento a números entendibles por la red neuronal. Esta transformación se realiza pasando de un formato numérico a palabras y la creación de un vocabulario de manera similar a problemas de procesado natural del lenguaje.

2. Red neuronal: Segmento donde se implementa la arquitectura, con los datos procesados en el segmento anterior, se entrena con una red neuronal basada en capas de memoria a largo plazo. La salida de la red es el índice del vocabulario que se vuelve a convertir a notas y estas a un archivo MIDI.

3. Postprocesado: Segmento donde se realiza la creación de una canción. Mediante la aplicación de un instrumento al archivo generado, la adición de una base rítmica y elementos adicionales, se consigue generar un archivo en formato MP3 que contiene una canción similar a canciones del género "Lo-Fi".

Para la elaboración de este proyecto, se ha realizado un estudio del estado del arte y los distintos acercamientos realizados sobre el mismo problema. Se han considerado distintas arquitecturas como las GAN, Transformers o LSTM; formatos de archivos como MP3, WAV o MIDI y lenguajes como Python, R o Java para resolver el problema. También se ven las distintas estaciones de audio digital de la literatura y se opta por una programable. De cada una de estas decisiones, se explica más en profundidad sus beneficios e inconvenientes.

Se profundiza más en el desarrollo y distintas iteraciones por las que ha pasado el proyecto, explicando que puntos fuertes y puntos débiles que tiene cada uno de estos por separado y en relación entre ellos. Estas tres iteraciones pasan por el entendimiento de una propuesta anterior, al desarrollo de cero de mejoras sobre esta propuesta hasta llegar a una iteración que obtiene beneficios de las dos anteriores.

Además, se mostrarán las evaluaciones realizadas a la salida producida por el modelo. Estas son dos evaluaciones que se realizan desde dos perspectivas. La perspectiva "objetiva": la cantidad de recursos que se gastan en la generación de una canción (tiempo y memoria). Y la perspectiva "subjetiva": mediante un test de Turing se determina la capacidad de las canciones generadas de parecerse a una producción humana.

También viendo los resultados de estas evaluaciones, se sacan conclusiones, viendo que se puede mejorar en los tres segmentos. La mejora del preprocesado para eliminar la perdida de información. La adaptación de un modelo para que resulte óptimo para el problema presentado. O la integración de más opciones creativas como más bases rítmicas o instrumentos, son mejoras que se explican en las líneas futuras y muestran a donde puede apuntar el desarrollo futuro.

Todo esto junto con un capítulo de discusión sobre los puntos más fuertes: alta modularidad del código, diseño encapsulado o potencial de mejora. Y los puntos más débiles: limitada variedad musical, escasez de datos de entrenamiento o sobreajuste de la red neuronal.

Abstract

This document gathers all the information generated in the design, development and testing of an artificial intelligence model capable of generating "Lo-Fi" song fragments, a popular music genre on Internet and commercial platforms. This model consists of three clearly differentiated parts:

1. Preprocessing: Segment where the MIDI format files of the training set are transformed to numbers understandable by the neural network. This transformation is done by going from a numerical format to words and the creation of a vocabulary in a similar fashion to natural language processing problems.

2. Neural network: Segment where the architecture is implemented, with the data processed in the previous segment, it is trained with a neural network based on long-term memory layers. The output of the network is the index of the vocabulary that is converted back to notes and these to a MIDI file.

3. *Post-processing:* Segment where the creation of a song fragment takes place. By applying an instrument to the generated file, adding a rhythmic base and additional elements, an MP3 file is generated containing a song able to be classified in a greater or lesser extent into the "Lo-Fi" genre.

For the development of this project, a study of the state of the art including the different approaches to the same problem has been carried out. Different architectures such as GAN, Transformers or LSTM; file formats such as MP3, WAV or MIDI and languages such as Python, R and Java have been considered to solve the problem. We also look at the different digital audio stations in the literature and choose a programmable one. For each of these decisions, their benefits and drawbacks are explained in more depth.

Next, it is delved into the development and different iterations through which the project has passed, explaining the strengths and weaknesses of each of these separately and in relation to each other. These three iterations go through the understanding of a previous proposal, to the development of zero improvements on this proposal until reaching an iteration that obtains benefits from the two previous ones.

In addition, the evaluations performed on the output produced by the model will be shown. Two evaluations are performed from different perspectives. On the one hand, the "objective" perspective: the amount of resources spent in the generation of a song (time and memory). On the other hand, the "subjective"

perspective: by means of a Turing test, the ability of the generated songs to resemble a human production is determined.

From the results of these evaluations, conclusions can be drawn, showing that there is room for improvement in all three segments. The improvement of the preprocessing to eliminate the loss of information. Adapting a model to make it more optimal for the problem presented. Or the integration of more creative options such as more rhythmic bases or instruments, are improvements that are explained in future lines and show where future development can point to.

All of this wrapped up together with a discussion chapter on the strongest points: high modularity of the code, encapsulated design or potential for improvement. And the weakest points: limited musical variety, scarcity of training data or overfitting of the neural network.

Agradecimientos

A mis amigos, presentes durante todo el desarrollo, que han supuesto un pilar donde apoyarme en los momentos más bajos. En especial a los que día tras día han escuchado mis alegrías y mis lamentos en llamada.

A mi pareja que día tras día me daba ánimos y me preguntaba religiosamente el progreso de mi trabajo. Sin la preocupación, atención y dedicación que le pone a cada cosa que hago, probablemente no hubiese sido capaz de llegar a donde estoy.

A mi familia, que ha visto de primera mano el proceso completo. En especial a mis padres por su disposición a ayudar y las recomendaciones que me daban para mejorar. Han servido de más utilidad de lo que esperaban, tanto a nivel del proyecto como a nivel personal.

A mi familia académica, tanto personal, profesores y alumnos por formar el ambiente donde he podido conseguir realizarme como ingeniero y persona. En especial a los profesores que se dedican en cuerpo y alma para que las futuras generaciones salgamos mejor de como entramos a la facultad.

A Txema, mi tutor académico y quien más me ha aguantado por escuchar mis inseguridades, preocupaciones y preguntas. Tendría la mitad de lo conseguido, si no fuese gracias a las reuniones, guías, artículos y atención dada.

Gracias a todos.

Tabla de contenidos

Tabla de Figuras

1 Introducción

1.1 Motivación y necesidad del proyecto

Durante los últimos años, el uso de la inteligencia artificial (IA) en la vida cotidiana ha crecido exponencialmente [1] [2]. Por consecuencia, ha surgido una competición por conseguir desarrollar el modelo que mejor se ajuste a la realidad. Junto con otros campos, el de la música no se queda detrás y ya existen propuestas que muestran modelos capaces de crear canciones completas [3] [4]. Estos están entrenados con grandes capacidades computacionales y millones de datos de entrenamiento para poder conseguir resultados muy efectivos. Debido a las restricciones de tiempo, capacidad de desarrollo y de computación, es imposible plantear llegar a ese mismo nivel de arquitecturas respaldadas por empresas con equipos de desarrollo detrás. Es por esa razón que el presente trabajo de fin de grado se enfoca en la creación de un modelo de inteligencia artificial más simple, para poder contrastar si las diferencias entre unos modelos y otros son tan grandes como la diferencia de los costes. Esta comparación servirá para ver la viabilidad de implementar modelos más sencillos en elementos de bajo poder computacional como pueden ser los componentes de los sistemas del *Internet of Things* (IoT), así como en proyectos con bajo presupuesto.

Con respecto al estilo musical escogido, se ha elegido el "Lo-Fi" debido a varias razones. El uso de música del género se utiliza comúnmente entre la comunidad de internet para poner de fondo mientras se estudia o se relaja, incluso llegando a haber un directo activo 24h al día [5]. Esto, junto a la simplicidad a nivel de producción de sonido que conlleva, ha sido el factor que ha llevado a la inclusión de este en el proyecto. Esta decisión permite que el trabajo contenga una línea de procesamiento completa donde se produzca como salida final una canción más que un producto intermedio como puede ser solo la melodía.

1.2 Objetivos

Después de ver el razonamiento detrás de la necesidad de crear este proyecto, se puede inferir que el objetivo principal es la implementación de un modelo de inteligencia artificial sencillo capaz de generar música, específicamente del género "Lo-Fi". De esta meta principal, surgen los siguientes subobjetivos:

1. Aprendizaje de los sistemas de generación de audio para el óptimo diseño de modelos aplicando técnicas utilizadas por el estado del arte.

2. Diseño estructural del modelo aplicando los conocimientos adquiridos del estudio de la literatura.

3. Implementación y entrenamiento del modelo para su evaluación en comparación con otros acercamientos al problema.

1.3 Planificación

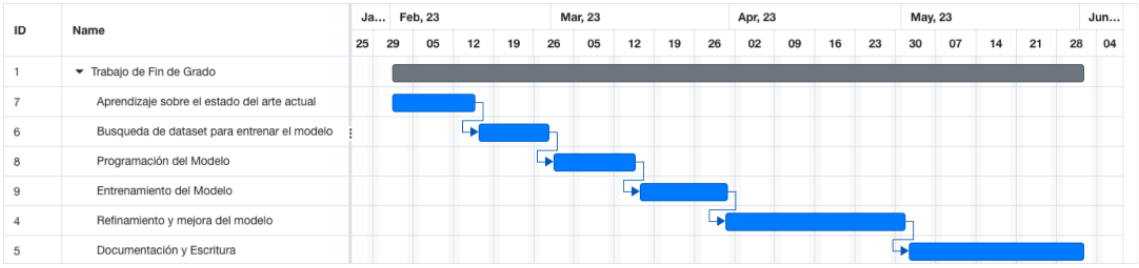

Figura 1: Diagrama Gantt realizado durante la planificación de trabajo

Para la planificación de desarrollo del modelo a lo largo de los cuatro meses de duración del trabajo, inicialmente se distribuyó siguiendo el diagrama de Gantt mostrado en la Figura 1. Este, se puede desglosar en seis tareas principales que se agrupan en tres fases:

1. Una primera fase de investigación sobre otros modelos y *datasets* disponibles para tener un mejor entendimiento del estado del arte y otros posibles enfoques del proyecto.

2. Una segunda fase de creación del modelo donde se hace el desarrollo principal de la red neuronal y del postprocesado.

3. Una tercera fase de refinamiento y documentación donde se recoge todo el trabajo realizado y se realizan las evaluaciones.

Basándose esta organización se ha realizado el trabajo y ha servido para marcar fácilmente cuanto tiempo debe de requerir cada una de las partes del desarrollo, simplificando la distribución de tareas a lo largo del tiempo.

1.4 Estructura de la memoria

A continuación, en el apartado 2 veremos el estado del arte de la generación de música mediante inteligencia artificial y las distintas tecnologías empleadas y las que han sido descartadas; en el apartado 3 ahondaremos en el modelo realizado y su arquitectura; en el apartado 4 se explicará el desarrollo así como las distintas iteraciones que ha tenido el proceso de creación; en el apartado 5 resultados sacados del desarrollo del sistema; en el apartado 6 las conclusiones en base a estos resultados; en el apartado 7 una sección de discusión sobre el producto final del proyecto; en el apartado 8 aportaciones futuras y en el apartado 9 el análisis de impacto que implica la creación de un sistema como este. Para concluir, en los apartados 10 y 11 la bibliografía y anexos.

2 Estado del arte

En el presente capítulo se realiza un análisis de las distintas plataformas, herramientas y sistemas existentes que permiten la generación de música (ya sea del género *"Lo-Fi"* u otros), adicionalmente se hace una descripción del género "Lo-Fi" para establecer una definición consistente durante el trabajo de fin de grado. Cabe destacar que pocas arquitecturas del estado del arte vienen respaldadas de un estudio académico así que las referencias de la bibliografía serán a sus respectivas páginas principales como fuentes.

2.1 Definición del género *"Lo-Fi"*

El género de música conocido como "*Lo-Fi*" o *Low-Fidelity* consiste en una rama de la música basada en el empleo de medios de baja fidelidad o anticuados [6]. En lo que respecta a la melodía, normalmente suele ser un conjunto de notas a piano (o similares) de corta duración, repetido durante toda la canción. Este tipo de música ha explotado en popularidad durante las últimas décadas [7]. Aunque cualquier grabación del siglo XX o anterior se puede considerar música del género "*Lo-Fi*" (por ser grabada con equipos de grabación antiguos) realmente la definición se empieza a usar a partir de los años 2000, tras la aparición del casete y otras herramientas, que permitían grabar con mejor calidad los instrumentos. La principal cualidad que hacen a una canción "*Lo-Fi*" es la decisión deliberada de reducir la calidad de grabación mediante elementos antiguos o edición digital.

Específicamente un subgénero salido de éste es el "*Lo-Fi hip-hop"* o *chillhop* que se han adueñado por completo del nombre de "*Lo-Fi*" general. Algunas canciones de este subgénero pueden ser [8] o [9], en estas se puede ver el carácter relajante que intentan mostrar. Este tipo de música se caracteriza por tener una base rítmica, una melodía sencilla que se repite varias veces y elementos adicionales como puede ser lluvia, estática de un reproductor de vinilo u otros elementos que produzcan ruido. Para el presente proyecto, se va a trabajar sobre esta definición de "*Lo-Fi Hip-Hop*" como representación del estilo *"Lo-Fi"*, esto es categorizando una canción como *"Lo-Fi"* si contiene una melodía tocada a piano (o similares) de carácter relajado, una base rítmica y algún elemento de ruido blanco.

2.2 Acercamientos a la generación de música

Habiendo asentado las bases de la definición del género *"Lo-Fi"*, se procederá a mostrar distintas herramientas, tecnologías y sistemas presentes en el estado del arte.

El primer caso es la generación de música con inteligencia artificial a nivel general, en la que los actuales modelos se encuentran divididos entre tres categorías de afrontar el problema: procesamiento directo del archivo,

procesamiento de archivos de notación y procesamiento directo de ritmos. Nótese que van en orden de generalidad del procesado.

Adicionalmente y no contemplados en los tres subapartados, existen otros modelos que hasta el día de escritura de este documento siguen sin revelar su arquitectura por lo que no se puede ver en cuál de estas tres categorías entran. Una de las herramientas más poderosas de generación en este sentido es la herramienta de Aiva [3] desarrollada por AIVA Technologies que mantiene su arquitectura sin revelar. Debido a que en la aplicación se pueden editar notas e instrumentos, probablemente entraría dentro de la segunda manera (procesamiento de archivos de notación).

2.2.1 Procesamiento directo del archivo

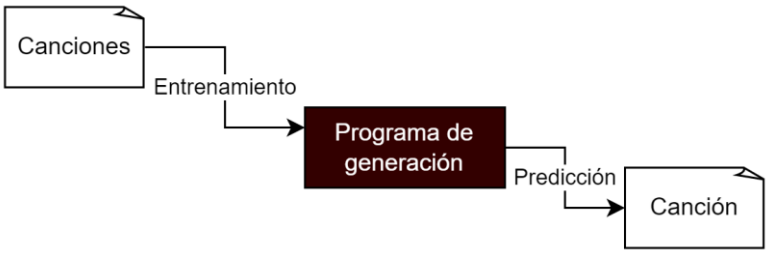

Figura 2: Diagrama general del procesado directo del archivo

Las arquitecturas de la primera categoría entrenan sus sistemas pasando el archivo de la canción en bruto y completo, sin ninguna división por pistas (como puede ser separar el tambor de la guitarra o de la voz, por ejemplo). En la Figura 2 se puede ver un esquema general que sigue la generación de estos programas. Con los archivos completos y mediante grandes capacidades computacionales y volúmenes de datos, consiguen entrenar una inteligencia artificial capaz de generar ondas de sonido para hacer música o adaptar letras a distintos géneros. Debido a que la generación de la canción surge desde un archivo de ruido blanco, que posteriormente se va refinando, se pueden oír pequeños segmentos de estática en las canciones generadas que nos muestran la utilización de este método. De estos sistemas destaca Jukebox [4], la propuesta de OpenAI.

2.2.2 Procesamiento de archivos de notación

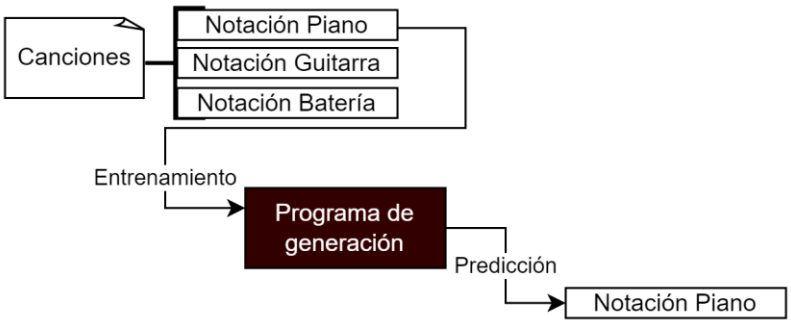

Figura 3: Diagrama general del procesado de archivos de notación

La segunda categoría de afrontar el problema de generación de música es utilizando archivos de notación como pueden ser los archivos MIDI. En la Figura 3 se puede apreciar mejor el funcionamiento de extraer un archivo de notación (el piano en este caso) con el cual se entrena al programa y puede predecir otro archivo de notación. Este acercamiento es más simple puesto que, al tener todas las notas, estas se pueden pasar a números de manera mucho más sencillas y permite dividir las canciones en distintas pistas para poder distinguir qué instrumentos están sonando. En este sector existen propuestas como MuseNet [10] de OpenAI o magenta [11] de Google.

2.2.3 Procesamiento directo de ritmos

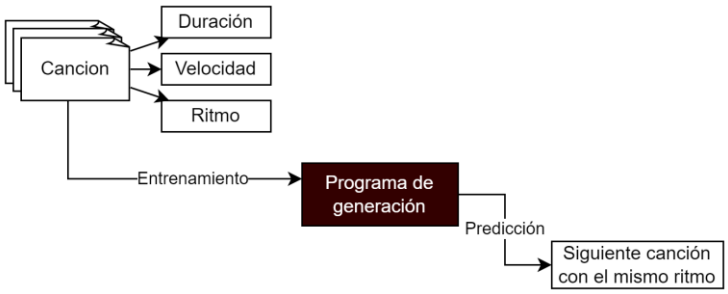

Figura 4: Diagrama general del procesado de archivos de notación.

La tercera categoría, y la más sencilla a nivel de complejidad de procesado, es la utilización de modelos de inteligencia artificial para realizar la tarea de orquestación y sincronización de diferentes pistas. En la Figura 4 se muestra el procesado de esta manera, donde la simplicidad proviene de que en la salida solamente se pide una canción que tenga el mismo ritmo y velocidad. La tarea final de estos proyectos consiste en la realización de un paso gradual de un tipo de sonido a otro utilizando distintas pistas con el mismo ritmo. En este sentido solamente se ha podido encontrar el proyecto de Larnii [12].

2.3 Métodos de representación de sonidos digitalmente

A la hora de almacenar los distintos elementos que conforman una canción en forma de ficheros, existen distintas maneras de guardarlos. Estas distintas maneras, difieren dependiendo de la cantidad de información que se quiera perder al momento de la compresión. A continuación, se verán por encima los distintos tipos de archivos que se pueden manejar a la hora de realizar un proyecto de este tipo.

2.3.1 MP3, WAV y FLAC

MPEG-1 Audio Layer III (MP3) es un formato de compresión de audio con pérdida, que reduce el tamaño del archivo sacrificando cierta información de este. Utilizando algoritmos de compresión que eliminan partes del espectro de audio menos perceptibles, consigue crear archivos más pequeños que los demás formatos y es por esto que suele ser la opción más común para trabajar con archivos de sonido que no requieren alta fidelidad con la fuente original (como pueden ser mensajes de audio en una aplicación de mensajería).

A un lado diametralmente opuesto del espectro, esta *Waveform Audio File Format* (WAV) que consiste en un formato de archivo de audio sin comprimir, en bruto; haciéndolo así más fácilmente ajustable a distintas plataformas y aplicaciones. La forma de almacenamiento de estos archivos es en forma de onda directa sin comprimir, haciéndolo así compatible con diferentes tasas de muestreo y profundidades de bits.

Junto con WAV también surge otro formato como *Free Lossless Audio Codec* (FLAC). Creado con la finalidad de permitir la compresión de archivos sin sacrificar calidad (como ocurre en MP3). El uso de este especialmente se ve en proyectos de creación musical, debido al requerimiento de muchos más elementos sonoros en el proceso de creación. Es una opción bastante beneficiosa a la hora de manejar varios archivos de sonido puesto que tenerlos en formato WAV requeriría más espacio. Debido a que como máximo se utilizan tres archivos de sonido durante el procesado de la canción del proyecto, la utilización de este formato de archivo no ofrecía mucha ventaja en comparación con WAV y no se terminó utilizando.

2.3.2 MIDI

Musical Instrument Digital Interface (MIDI) es un protocolo estándar que permite la comunicación y control de instrumentos musicales electrónicos, computadoras y otros dispositivos relacionados con la música. En vez de almacenar la onda de sonido de maneras comprimidas, el formato MIDI guarda la información de eventos musicales como notas, duración, intensidad y otros parámetros relacionados con la interpretación musical. Específicamente, la información de los archivos MIDI, se almacena como una secuencia de mensajes que indican cómo debe reproducirse la música. Son estos mensajes que tienen parámetros como qué notas deben sonar, durante cuánto tiempo o con qué

intensidad. Es por esto que los archivos MIDI son pequeños en tamaño y son elegidos para la representación musical debido a su alta flexibilidad y usabilidad.

2.4 Lenguajes para el desarrollo de modelos de IA

En el momento de realización de un proyecto de programación, uno puede pensar que cualquier lenguaje de alto nivel puede servir para completar esta tarea, pero a no ser que se disponga de una gran cantidad de tiempo y habilidades suele ser preferible sopesar la viabilidad de los distintos lenguajes para poder optimizar el desarrollo. En lo que respecta al desarrollo de modelos neuronales a continuación se mostrarán tres de los lenguajes más usados para tal fin.

2.4.1 Java

Java es un lenguaje de programación ampliamente utilizado y reconocido en el mundo del desarrollo de software [13]. Conocido por su portabilidad, rendimiento y enfoque en la orientación a objetos, Java ha dejado una huella significativa en la industria. Desde su lanzamiento en 1995, ha sido adoptado por programadores y empresas de todo el mundo, convirtiéndose en un pilar fundamental en el desarrollo de aplicaciones.

Aunque pueda requerir más líneas de código en comparación con otros lenguajes como Python, Java se destaca por su velocidad y escalabilidad. Con bibliotecas como *Deeplearning4j*, Java ofrece la posibilidad de crear modelos de *Deep learning*. Su enfoque en la compilación en *bytecode* y su sólido sistema de gestión de memoria optimizan el rendimiento de las aplicaciones. Sobre todo, en proyectos de gran envergadura, la arquitectura orientada a objetos de Java facilita la organización y estructuración del código.

2.4.2 R

R es un lenguaje de programación estadística ampliamente utilizado y altamente valorado en el ámbito del análisis de datos y la investigación estadística [14]. Con su rico conjunto de bibliotecas y funciones especializadas, R ofrece a los usuarios una amplia gama de herramientas para el procesamiento, análisis y visualización de datos. Desde su introducción en 1993, R se ha convertido en un estándar de facto en la comunidad estadística y científica, proporcionando una plataforma flexible y poderosa para realizar análisis estadísticos complejos y generar gráficos de alta calidad.

Debido a la especialización en estadísticas y análisis de datos que ofrece R y lo diferente que puede llegar a ser comparado con otros lenguajes de programación, sería de esperar que no resultase eficiente a la hora de desarrollar modelos de inteligencia artificial. Pero es debido a su gran capacidad de análisis estadístico y de datos que le da una ventaja sobre otras opciones. Principalmente utilizado

en la comunidad académica y en la rama de investigación de la Inteligencia Artificial.

2.4.3 Python

Python es un lenguaje de programación versátil y de alto nivel que se ha ganado una enorme popularidad en diversos campos [15]. Con una sintaxis clara y legible, Python se destaca por su facilidad de uso y su enfoque en la legibilidad del código. Desde su creación en la década de 1990, Python ha ganado impulso y se ha convertido en uno de los lenguajes más utilizados en el desarrollo de aplicaciones web, análisis de datos, inteligencia artificial y automatización de tareas.

Aunque sea más lento en comparación a Java y R, Python es uno de los lenguajes más escogidos para el desarrollo de IA debido a su mejor legibilidad y facilidad de uso junto con otras ventajas. Una de estas ventajas es que Python cuenta con bibliotecas que sirven de gran utilidad debido al alto nivel en el que operan como *TensorFlow* y *PyTorch*. Es en parte por estas razones y por cuestiones académicas que para este proyecto se eligió el desarrollo del modelo en Python.

2.5 Librerías para el desarrollo de inteligencia artificial

Habiendo introducido en la sección anterior las dos principales librerías de Python utilizadas en el desarrollo de modelos de Inteligencia Artificial, se va a hacer más hincapié en ambas, los puntos fuertes y débiles de cada una y en la elección final de una sobre la otra para este proyecto.

2.5.1 PyTorch

PyTorch es una biblioteca de aprendizaje automático desarrollada por *Facebook's AI Research Lab* que se basa en la utilización de estructuras dinámicas para realizar las computaciones [16]. Algunas de las características destacadas de PyTorch son:

- Flexibilidad: Permite un mayor control y flexibilidad en la construcción de modelos con el uso del enfoque imperativo, lo que significa que se pueden definir y modificar los modelos en tiempo real con facilidad. Esto lo hace especialmente útil en la investigación y el desarrollo de prototipos rápidos.

- Grafos computacionales dinámicos: Procesamiento mediante la utilización de un grafo computacional dinámico capaz de modificarse durante el tiempo de ejecución. Este procesamiento permite una mayor flexibilidad en la construcción de modelos que pueden tener estructuras cambiantes en distintas iteraciones.

- Comunidad activa: Debido al auge de popularidad de la inteligencia artificial, cuenta con una comunidad activa de desarrolladores que contribuyen con nuevas ideas, extensiones y soluciones.

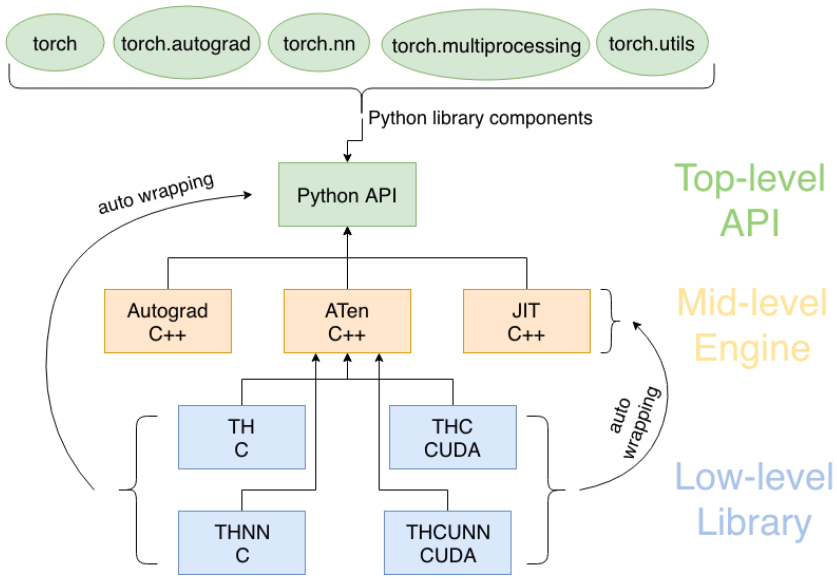

Figura 5: Modelo Modulo Estructura de [17]

Más específicamente se puede ver la organización de PyTorch en la Figura 5. Se puede ver que la biblioteca proporciona una interfaz fácil de usar en Python, pero los cálculos se realizan en motores de cálculo eficientes escritos en C++. Estos motores se dividen en tres niveles: *autograd* para cálculos y diferenciación automática, *JIT compiler* para optimizar pasos de cálculo, y *ATen*, una biblioteca de tensores en C++ que envuelve una biblioteca de bajo nivel para PyTorch. Las bibliotecas de bajo nivel realizan la mayoría de los cálculos intensivos asignados por la API de alto nivel, proporcionando estructuras de datos eficientes y funciones de redes neuronales. *ATen* envuelve las bibliotecas de bajo nivel y las expone a la API de Python de alto nivel, manteniendo el código desacoplado y facilitando el desarrollo.

2.5.2 Keras

Keras es una biblioteca de alto nivel que, aunque se pueda utilizar de forma independiente, es una capa de abstracción sobre otras bibliotecas de aprendizaje automático como *TensorFlow* [18]. Algunas de las características destacadas de Keras son:

- Simplicidad: Está diseñado para ser fácil de usar y permite desarrollar modelos de manera rápida y sencilla. Proporciona una API intuitiva y una sintaxis clara, lo que facilita la construcción y el entrenamiento de redes neuronales.

- Modularidad: Basado en una estructura modular que permite a los desarrolladores construir modelos de forma incremental. Estos se crean mediante la combinación de capas, lo que facilita la construcción y experimentación con diferentes arquitecturas.

- Amplia compatibilidad: Keras permite la integración con múltiples motores de *backend*, incluido *TensorFlow*, lo que permite aprovechar todas las capacidades de estas bibliotecas subyacentes. Esto proporciona una gran flexibilidad para elegir el *backend* según las necesidades del proyecto.

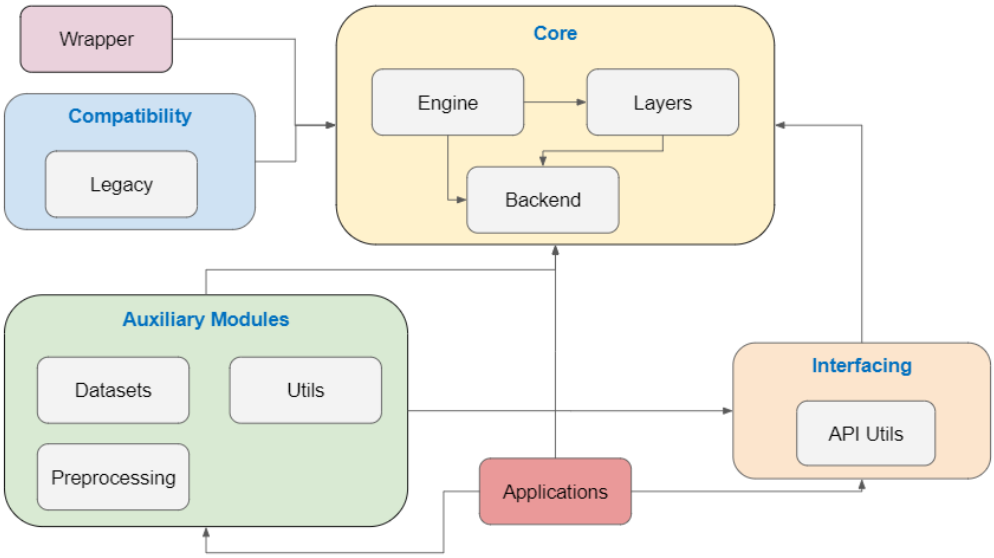

Figura 6: Modelo Módulo Estructura de [19]

Entrando más en profundidad sobre la organización de Keras, los módulos representados en la Figura 6 muestran las categorías de las funciones que implementa. El *core* se encarga de implementar las funcionalidades esenciales para definir arquitecturas de modelos, mientras que los módulos auxiliares complementan esta implementación con funciones adicionales de preprocesamiento y *datasets*. Por último, La compatibilidad se encarga de soportar versiones anteriores de Keras, las aplicaciones proporcionan implementaciones de modelos pre-entrenados populares y la interfaz ofrece una API para acceder a las funcionalidades del *core*.

2.6 Arquitecturas consideradas

A continuación, se procederá a ver y explicar las tres principales arquitecturas sopesadas durante el desarrollo del proyecto. Posiblemente existan otras arquitecturas que, tras más tiempo de investigación o más destreza en la materia, estarían presentes en este listado.

2.6.1 LSTM

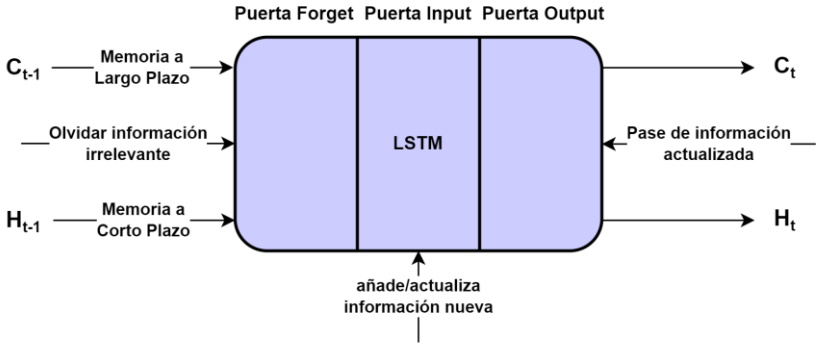

Figura 7: Estructura general de una neurona LSTM

Las *Long Short-Term Memory* (LSTM), son un tipo de arquitectura de redes neuronales recurrentes (RNN) que, a diferencia de las redes neuronales tradicionales, están diseñadas para capturar y almacenar información relevante a largo plazo en secuencias de datos. Esto las hace especialmente adecuadas para tareas de procesamiento del lenguaje natural y otras secuencias temporales. Esta capacidad de retener información a largo plazo y gestionar las dependencias temporales ha hecho que las LSTM sean ampliamente utilizadas en tareas como el reconocimiento de voz, la traducción automática y la generación de texto.

En la Figura 7, se puede ver la estructura de una neurona de una capa LSTM. Cada neurona está dividida en tres partes: una puerta *forget* que permite olvidar información irrelevante de la anterior neurona, una puerta *input* que recibe la información de la secuencia y una puerta *output* que pasa la información a la siguiente neurona. Al igual que las RNN, cada neurona tiene un estado *hidden* (H) y un estado de *cell* (C). Ambos estados representan la memoria a corto plazo y la memoria a largo plazo respectivamente. Y es debido a la capacidad de modelar y entender el contexto secuencial que consigue esta arquitectura, que las LSTM han sido fundamentales para avances significativos en campos como el del procesamiento del lenguaje natural.

2.6.2 GAN

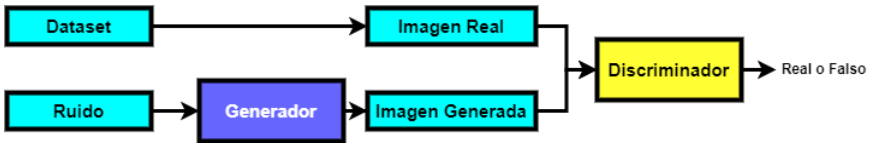

Figura 8: Esquema general de la Arquitectura de una GAN

Las *Generative Adversarial Networks* (GAN) o Redes Generativas Adversarias, son un tipo de arquitectura de redes que consiste en la existencia de dos redes de neuronas: el generador y el discriminador. En la Figura 8 se puede ver con más detalle el funcionamiento de ambos. El generador toma como entrada un

vector de ruido (entiéndase ruido como valores aleatorios para cada parámetro) y trata de generar datos sintéticos que se asemejen a los datos reales de entrenamiento. Por otro lado, el discriminador actúa como un clasificador que intenta distinguir entre los datos reales y los generados por el generador. Ambos componentes se entrenan de manera adversarial, es decir, el generador busca engañar al discriminador, mientras que el discriminador intenta ser cada vez más preciso en la detección de datos falsos. A medida que ambos se entrenan juntos, las GAN aprenden a generar datos realistas y de alta calidad que son indistinguibles de los datos reales. Este tipo de arquitecturas ha demostrado ser especialmente eficaz en la generación de imágenes, texto y otros tipos de datos complejos. Debido a la necesidad de crear dos redes neuronales, se descartó esta idea, también por el alto riesgo de que los modelos aprendiesen los datos de entrenamiento e hiciesen *overfitting* (o sobreajuste) debido a la poca cantidad de datos de la que se disponía.

2.6.3 Transformers

A partir de la publicación del paper *Attention Is All You Need* [20], ha surgido un crecimiento considerable en el desarrollo de modelos que contienen capas de atención. Estos fragmentos son los *Transformers* que están detrás de tecnologías como *ChatGPT* [21] y han revolucionado el campo de la inteligencia artificial, ofreciendo una alternativa potente y eficiente para el procesamiento de secuencias en diversos dominios de aplicación.

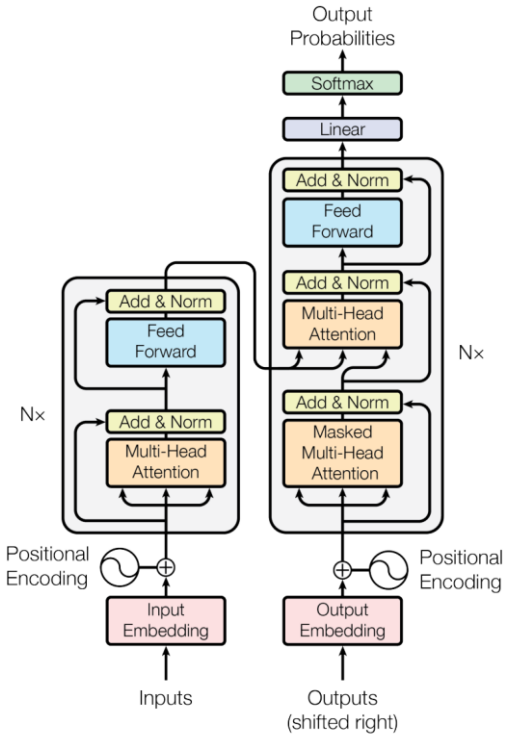

Figura 9: Estructura de un *Transformer* obtenida de *Attention Is All You Need* [20]

Para ser más específicos, los *Transformers* son un tipo de arquitectura que se basan en el concepto de atención, permitiendo que los modelos capturen las relaciones y dependencias entre las diferentes partes de una secuencia de entrada; y capaces de procesar múltiples elementos de entrada simultáneamente a diferencia de las RNN. En la Figura 9 se puede ver la arquitectura que tiene una red neuronal que es un Transformer. Este diagrama está dividido en dos partes: la parte izquierda que hace de codificador y la derecha de decodificador. Los distintos rectángulos de colores representan las capas que contiene, la capa importante a la que hay que prestar atención, son las tres de color naranja *(Multi-Head Attention*, y *Masked Multi-Head Attention*) que son las que le dan la cualidad de capturar las relaciones y dependencias de distintas partes de la secuencia. Esta cualidad ha conseguido catapultar en eficacia a los *Transformers* en tareas de procesamiento del lenguaje natural, como la traducción automática y el análisis de sentimientos. Debido a la complejidad de sistemas de este estilo y las reducidas capacidades de desarrollo y tiempo, no se optó por esta arquitectura.

2.7 Métodos de preprocesado de archivos de sonido considerados

Tras ver las distintas arquitecturas comúnmente utilizadas para este y otros problemas, puede surgir la consideración de aprovechar los avances realizados en otros campos de la inteligencia artificial (como el procesado del lenguaje natural o la generación de imágenes) para poder potenciar y guiar el desarrollo del proyecto. Específicamente, a la hora de introducir la información a un modelo de inteligencia artificial, el paso de eventos musicales (archivos MIDI) a números que pueda entender la red puede ser realizado de distintas maneras.

2.7.1 De formato MIDI a imagen

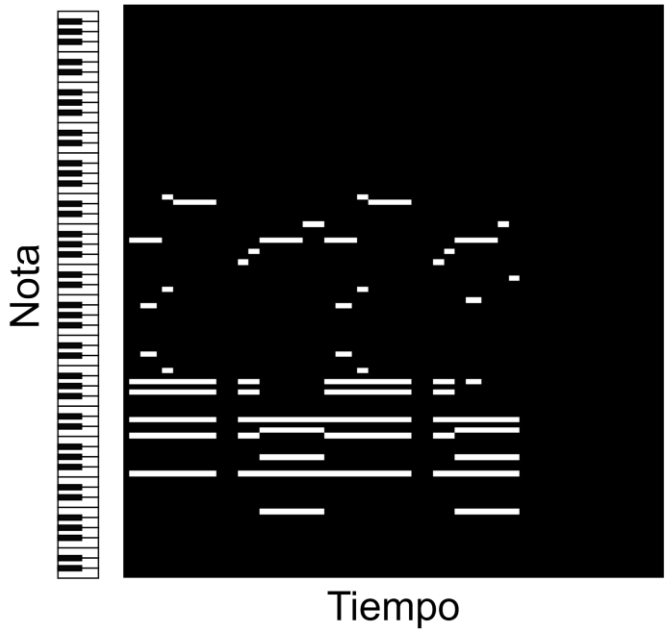

Figura 10: Representación de archivo MIDI en Imagen

Habiendo visto los recientes avances en la generación de imágenes con herramientas como *DALL-E* [22] o *Stable Diffusion* [23]. Se podrían utilizar elementos y progresos de este campo para poder generar mejores canciones. Es por esto que se consideró el desarrollo de un transformador de archivos MIDI a imágenes (formato PNG o JPG) para poder introducirlos a un modelo capaz de generar una imagen parecida y acto seguido pasarla de vuelta a MIDI.

Como se puede ver en la Figura 10, este sería un ejemplo de un archivo pasado por este proceso. En el eje Y de la imagen se representan todas las notas del piano, mientras que en el eje X sería su duración y aparición en el tiempo. Una de las restricciones de este proceso es que elimina el resto de información contenida en los archivos MIDI como puede ser la intensidad, presión o velocidad. Finalmente se descartó por la posible dificultad que supondría la generación de una canción con un sentido melódico dado el tiempo y conocimiento en la materia.

2.7.2 De formato MIDI a palabras

El otro campo de la inteligencia artificial donde se ha visto gran progreso es el procesado de lenguaje natural o *Natural Languaje Processing* (NLP) con modelos como *ChatGPT* [21] o *Github Copilot* [24].

Debido a la estructura de mensajes de los archivos MIDI, los mensajes contienen un atributo que dictamina qué notas que suenan en ese instante de tiempo. Cada nota puede convertirse al valor de su tono y estos números codificarse como palabras. Por lo directa que resulta esta conversión y lo desarrollada que esta esta rama de la inteligencia artificial, es una opción

bastante buena y por la cual se ha optado en el desarrollo de este proyecto como se explicará más adelante en profundidad en el capítulo 3.

2.8 Tecnologías empleadas

A parte del formato de archivo, lenguaje, librería, arquitectura y método de procesado de sonido elegidos, durante el desarrollo del proyecto se ha utilizado una herramienta adicional, la estación de trabajo de audio digital.

En el mundo de la producción musical la herramienta principal utilizada son los *Digital Audio Workstation* (DAW), tecnología capaz de poder asignar instrumentos a archivos MIDI, sobreponer archivos de sonido y sintetizarlo todo en una canción final. Es por esto que para la parte de post procesado se necesitaba utilizar uno para poder resultar en una canción un poco más realista que el simple archivo MIDI generado.

2.8.1 DAWs interactivos

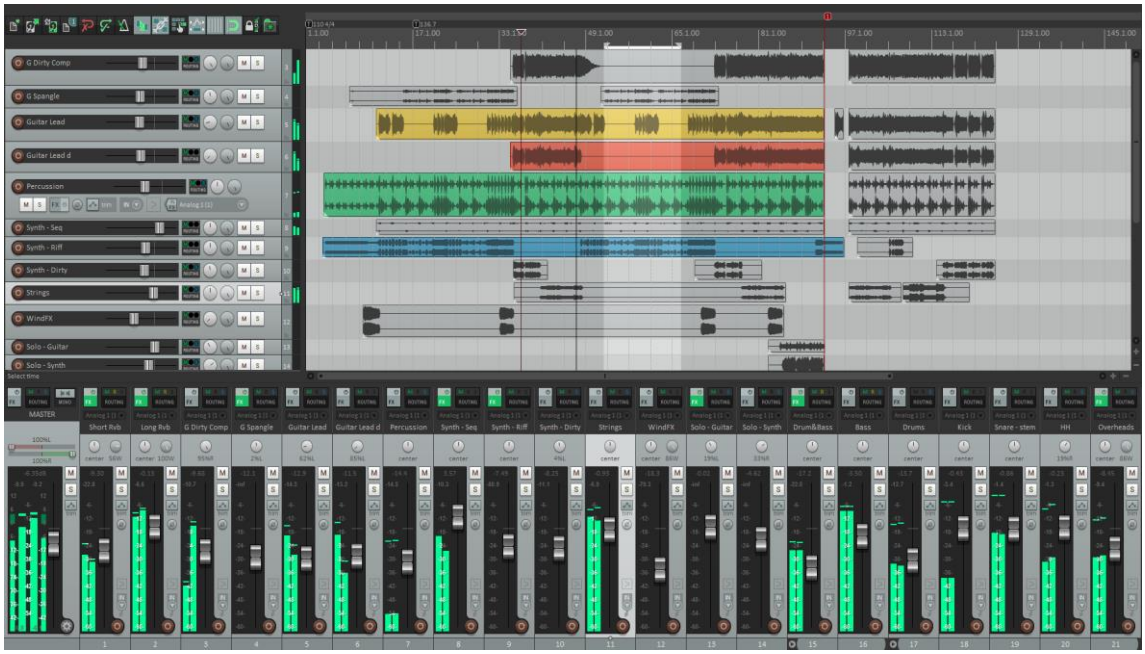

Figura 11: Interfaz del programa Reaper

Figura 12: Interfaz del programa FL Studio

En este contexto, la cualidad de "interactivo" se refiere a la existencia de una interfaz gráfica de usuario (GUI) a diferencia de los "programables". Estos, son programas que permiten arrastrar archivos MIDI e instrumentos de un lado al otro, ordenar en la línea temporal cuando aparecen ciertos elementos y cambiar la forma de sintetizar el sonido mediante una interfaz. De entre las estaciones de trabajo más famosas están *FL Studio* [25] y *Reaper* [26]. En la Figura 11 y Figura 12 se pueden ver capturas de pantalla de cada una de estas aplicaciones que muestran la interfaz gráfica. Debido al deseo de la automatización del proceso de la generación de la canción, se requería de un DAW que fuese programable y adicionalmente no tuviese un coste económico como puede ser el caso de estos dos ejemplos.

2.8.2 DAWs programables

Debido a la restricción que implica el utilizar el lenguaje Python, solamente existe una librería que implemente las acciones realizadas por las aplicaciones descritas en el apartado anterior, esta librería es *DawDreamer* [27] y contiene todas las herramientas necesarias para el proceso de postproducción requerido para este proyecto. Estas herramientas son: aplicación de un instrumento a un archivo MIDI, renderizado del producto y automatización del proceso. En específico, se requería que pudiese aplicar instrumentos del tipo *Virtual Studio Technology* (VST) a un archivo MIDI. Los VST son un plugin (plugin como

componente de software añadido) por el cual se aplica el instrumento y por el cual también se configuran distintos elementos.

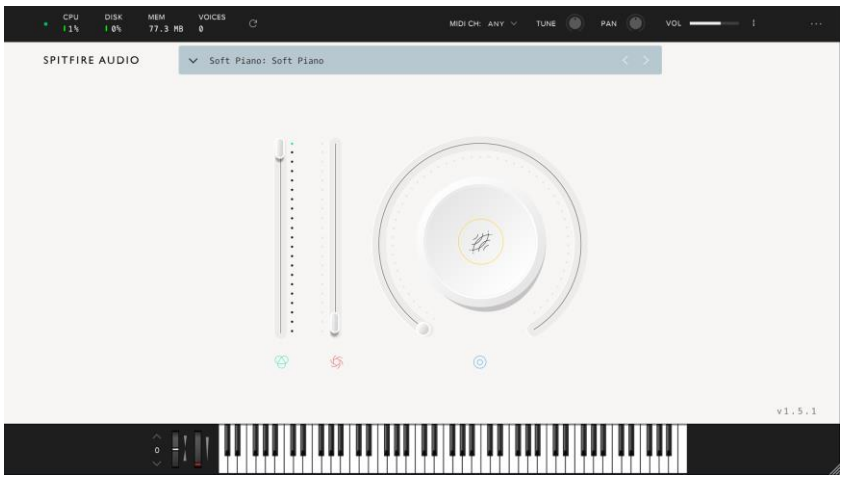

Figura 13: Interfaz del Plugin de piano

Figura 14: Interfaz del plugin de efecto de vinilo

3 Diseño del modelo

Tras ver todos los elementos que hay que tener en cuenta a la hora de realizar un modelo de IA capaz de crear melodías "*Lo-Fi*", se procede a diseccionar el diseño del programa. Primero, veremos la línea de procesado final para la generación de una canción. Segundo, se profundizará más en la explicación del desarrollo de la fase de entrenamiento que se divide en tres segmentos: preprocesado, modelo y postprocesado.

3.1 Organización de la línea de procesado final

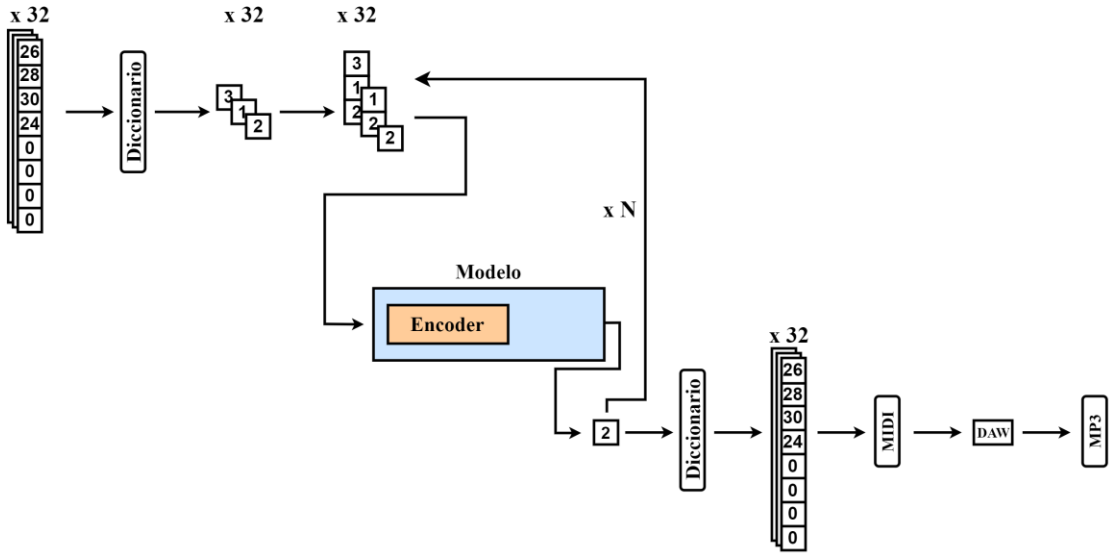

Figura 15: Diagrama de la línea de procesado de generación de una canción

En la Figura 15 se puede observar un diagrama de la línea de procesado general por la que pasa la generación de una canción.

1. Obtención de notas: De primeras se obtienen 32 notas (la longitud de secuencia escogida) que en la generación de canciones del proyecto se decidide eligiendo una posición aleatoria de todas las notas de entrenamiento y cogiendo las siguientes 32 notas.

2. Paso a diccionario: Tras esta obtención, la secuencia se pasa por un diccionario que contiene todo el vocabulario y asigna a cada nota su posición en dicho diccionario (La primera de toda la secuencia es un 1, la segunda un 2, etc.).

3. Generación de los n-gramas: Con los índices del diccionario se generan los n-gramas. Los n-gramas consisten en que cada elemento pasa a ser una secuencia desde el primero hasta el propio elemento (el primero tiene tamaño 1, el segundo tamaño 2 con el primero y segundo, etc.) y

rellenados hasta una longitud fija con el *token* de *pad* (de índice en el diccionario de 0).

4. Paso por el modelo: Después de esta primera parte de preprocesado de la información, se pasa al modelo que devuelve el valor del índice del diccionario que por búsqueda inversa nos determina la nota, este valor se añade a la secuencia anterior y se vuelve a pasar. Así hasta cumplir el número de notas que se piden.

5. Postprocesado: Ya con todas las notas generadas, se pasan a un archivo MIDI y mediante el DAW se instrumentaliza y genera un archivo MP3.

3.2 Organización de las capas

A continuación, se presenta en la Figura 16 el esquema general que tiene la estructura del modelo. Este mismo esquema se puede dividir en 3 partes:

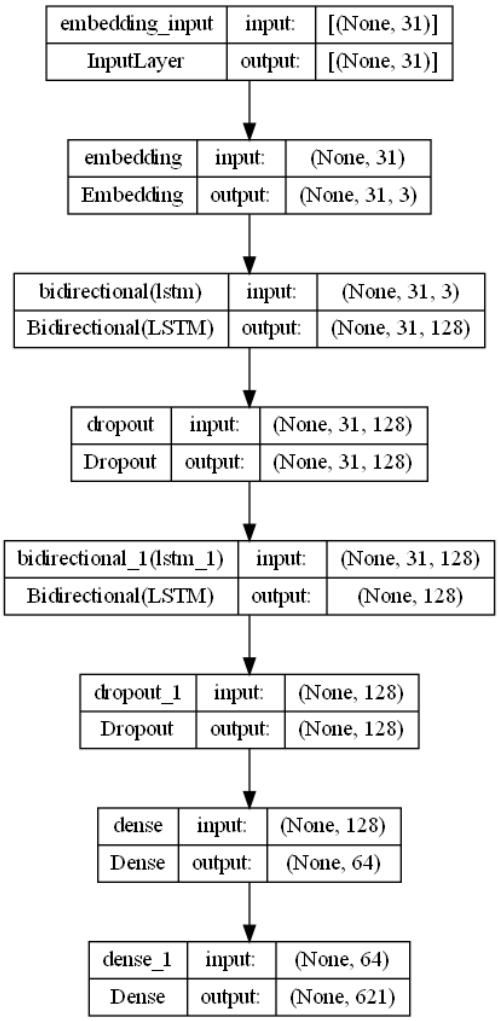

Figura 16: Diagrama de las capas del modelo

Primero, el modelo recibe 31 secuencias de n-gramas, este número es debido a que originalmente, las secuencias son de 32 n-gramas pero la última se utiliza para comparar la salida. Tras pasar esta secuencia pasan las notas por una capa de *embedding* que transforma cada nota en vectores de dimensión 3. Esta capa de *embedding* se explica en mayor profundidad en la sección 3.3.1.2.

Segundo, tras esta transformación, se pasa por dos capas LSTM Bidireccionales con una capa de *dropout* tras cada una. Estas dos capas de *dropout* son importantes para ayudar a prevenir la optimización de todas las neuronas de manera síncrona.

Tercero y finalmente, las dos capas *dense* permiten transformar la salida de la parte anterior para que tenga el tamaño del vocabulario, que como se puede ver en el diagrama es 621 para el material de entrenamiento.

Como se ha realizado un acercamiento similar al de un procesador de lenguaje natural, se puede ver esta estructura intermedia de capas LSTM y *dropout* que se encuentra en los procesadores más sencillos del lenguaje porque provee buenos resultados. Normalmente suelen contener más veces esta estructura de LSTM y *dropout,* pero debido al pequeño tamaño del vocabulario y de datos de entrenamiento, se debía tener cuidado con que fuese demasiado efectiva la red y aprendiese demasiado los datos de entrenamiento (*overfitting*).

3.3 Arquitectura elegida

En el apartado anterior se ha podido ver una imagen general del proceso por el que pasan los datos, a continuación, se describirá con más detalle qué procesos se le realiza a la información antes, durante y después de pasarla al modelo.

3.3.1 Introducción y procesamiento de datos

Debido a que un modelo de inteligencia artificial solamente conoce de números, es muy importante el segmento donde se transforma la información original a estos números, sobre todo porque si se realiza mal, se puede perder gran cantidad de información que puede resultar crítica para la tarea que se quiere realizar. De este proceso se ha decidido extraer solamente el valor del tono de las notas, perdiendo así otros atributos como la intensidad de estas.

3.3.1.1 Paso de notas a números

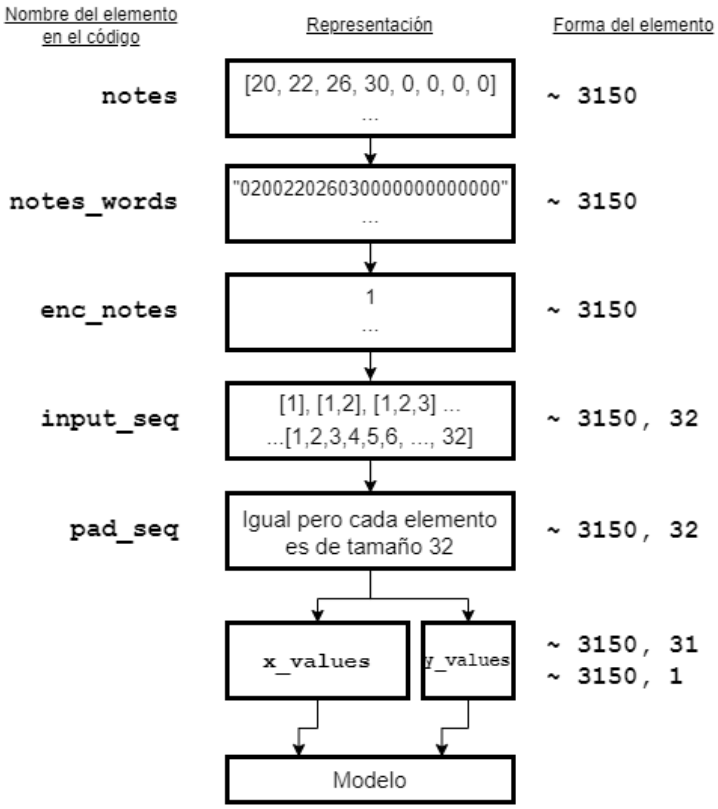

Figura 17: Esquema general del preprocesado de las notas

Para realizar el paso a números, lo primero que se decidió hacer fue convertir cada mensaje que contuviese notas a una lista de números. Como se ha explicado anteriormente, un archivo MIDI consiste en mensajes de eventos musicales (determinando que nota suena en qué momento), estos se pueden recorrer de manera secuencial y cada uno puede ser de tres tipos: acorde, nota o pausa. Para este proyecto, solamente se ha tenido en cuenta dos de estos tres tipos, acordes y notas. Cada mensaje de estos dos tipos contiene notas (o solo una nota) que pueden ser pasados al valor de su tono, para el set de entrenamiento el máximo obtenido es 119.

Al poder convertirlo a los valores de los tonos (del 0 al 119) nos deja la posibilidad de pasarlos a una lista de números. Durante todo el material de entrenamiento no hay ningún acorde que pasase de contener más de 7 notas así que se decidió poner un tamaño máximo de 8 por precaución. Teniendo ahora listas de tamaño 8 que representan a cada nota o acorde tocados durante la canción, estas tienen que transformarse a objetos de la clase *String* para pasarlas al diccionario. Este proceso podría saltarse y meter la información al diccionario directamente, pero como el acercamiento escogido se basa en el procesado del lenguaje natural, resulta más fácil de entender pasarlo a palabras.

Dichas palabras se introducen en el diccionario que contiene todo el vocabulario, la primera palabra de este, con posición 0, está reservada para el *token* de *pad* que se utiliza en la parte siguiente de generación de los n-gramas. Así se va completando el vocabulario una a una y tras esto se pasa cada nota al valor de su posición en el diccionario.

Para la fase de entrenamiento se generan las secuencias (de tamaño 32 para los ejemplos realizados) y estas se convierten en n-gramas que consisten en la conversión de cada elemento a una lista de elementos en orden ascendente: el elemento en primera posición es de tamaño 1, el de segunda posición contiene los dos primeros, el tercero los tres primeros y así hasta los 32. Luego se convierten todos a tamaño fijo (32 en este caso) con el token de *pad* anteriormente mencionado. Para finalizar, se divide en un grupo de 31 elementos para la entrada y otro de 1 para la comparación en la salida. En la Figura 17 se muestra un esquema de este proceso que puede resultar más esclarecedor.

3.3.1.2 Desarrollo de un codificador

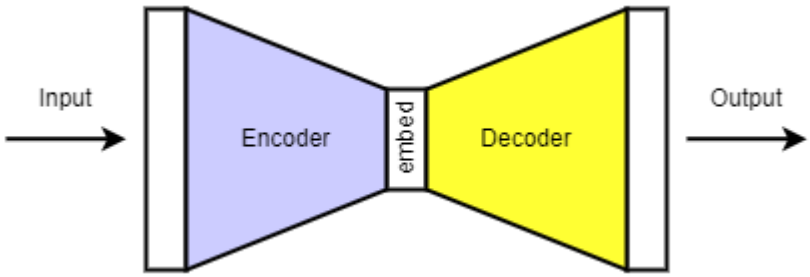

Figura 18: Diagrama simplificado de la estructura de un *autoencoder*

Para dotar de más contexto al modelo, una de las optimizaciones que se aplican a los modelos de lenguaje natural, es el uso de un *autoencoder* o codificador, este es otra red neuronal que se entrena con la finalidad de producir a la salida lo mismo que la entrada pasando por una capa de un tamaño más reducido como se puede ver en la Figura 18. Este proceso obliga a sintetizar los distintos elementos a unas pocas variables, cosa que permite poder colocar según la cercanía los distintos elementos que se le pasan y convertir los acordes (o notas) en vectores de *embeddings*. Como este campo ha sido estudiado con anterioridad, existen muchos tipos de codificadores que pueden cumplir con esta tarea, para el desarrollo del proyecto se ha decidido usar el modelo *word2vec* que ha conseguido demostrar buenos resultados.

3.3.2 Generación de información

Con las notas convertidas ya a un formato apto para la introducción al modelo, se pasa a la fase de entrenamiento. Para el desarrollo de los ejemplos se entrenó el *word2vec* durante 100 *epochs* y la red principal durante 120. Es importante tener en cuenta que el modelo principal rápidamente hacía *overfitting* así que no se pudo entrenar durante mucho más tiempo. Después de esta fase, se pasa por la línea de procesado presentada en el apartado 3.1 y se consigue generar combinaciones de notas no vistas en los datos de entrenamiento.

3.4 Postprocesado de la melodía generada

Adicionalmente para poder hacer algo que pueda ser considerado como una canción, se ha realizado una parte de postprocesado de la señal MIDI generada del modelo. Para esta fase se implementó un DAW programable que pudiese interpretar el archivo junto con otros elementos y poder conseguir un resultado final más cercano a una canción del género "*Lo-Fi*".

3.4.1 Procesamiento mediante el DAW

A la hora de procesar el archivo MIDI de la fase anterior, este ha de pasar por la estación de trabajo de audio digital. Para convertir las instrucciones del archivo a sonido, se tiene que introducir un instrumento que sonará siguiendo dichas instrucciones. En este caso de creación de una canción "Lo-Fi", se ha elegido el instrumento *LABS Soft Piano* [28] (visto en la Figura 13) que provee una serie de matices más suaves a la interpretación de las notas para dar una sensación de tranquilidad cómo la buscada por canciones del género. Además, también se pasa otro instrumento que produce un efecto de ruido de vinilo para provocar esa sensación de baja fidelidad. Para completar el procesado de la canción se le añade un sonido de lluvia comúnmente relacionado con canciones del género y una base rítmica. También, durante esta fase se decide el *beats per minute* (BPM) que dictamina la velocidad general de la canción, como la base rítmica tiene un BPM especifico (en este caso 88) una de las limitaciones de las canciones generadas es que deben tener también este BPM para poder estar acorde.

4 Desarrollo del modelo

El modelo final del proyecto ha sido el resultado de varias iteraciones, intentos y pruebas de implementación de los acercamientos que se podrían realizar. En el siguiente apartado se va a ver las distintas iteraciones por las que ha pasado el desarrollo del programa. Cabe destacar que el desarrollo del modelo final ha sido de manera progresiva, pero se van a señalar tres puntos clave en los que el modelo principal ha cambiado considerablemente.

4.1 Primera Iteración

En una fase previa a la primera iteración, se consideró el acercamiento mediante el paso de archivos MIDI a imágenes para meterlo posteriormente en la red neuronal. Esta idea previa, además, buscaba implementar un *Transformer* para ello, pero debido a la complejidad (a la hora de entender el funcionamiento) junto con la capa del paso a imágenes, se descartó por un modelo más sencillo. Tras esto la primera iteración resultó en implementar la red propuesta en [29] y el aprendizaje del funcionamiento de esta.

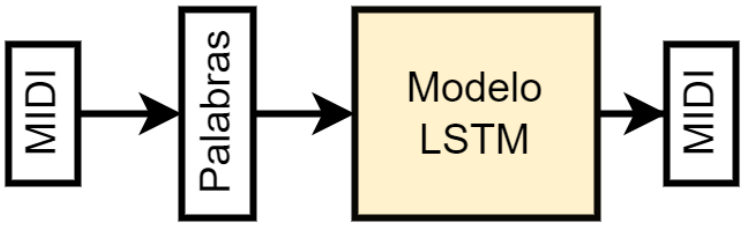

Figura 19: Esquema general del comportamiento de la primera iteración

En la Figura 19 se puede ver mejor el comportamiento del procesado de la primera iteración, nótese la falta de un postprocesado o un codificador que se explicará también a continuación. Resultados de esta iteración se pueden ver en el Anexo 1 y Anexo 2, hay que tener en cuenta que los archivos producidos son MIDI y puede que haga falta descargarlos para escucharlos.

4.1.1 Elementos base

Con respecto al preprocesado de las notas de esta iteración, es un procesado bastante directo, similar al acercamiento realizado en modelos de procesado del lenguaje natural. Los eventos musicales de notas marcados en los archivos MIDI tienen un atributo de nombre con las que se pueden clasificar mediante la nota y la octava en la que suenan (por ejemplo: B4, E5, C4, etc.). Los acordes se representan como concatenaciones de los valores del tono (por ejemplo: "26.28.32.") y así se consigue crear de manera similar a la iteración final un vocabulario con el cual podemos entrenarlo.

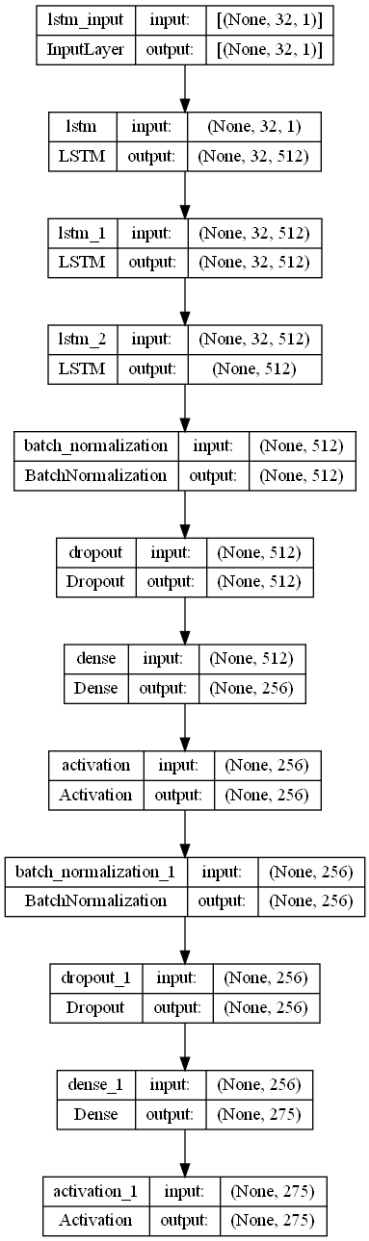

Figura 20: Diagrama de la red neuronal propuesto en [29]

En la parte del modelo de la red neuronal, se puede ver una organización sencilla de una LSTM, en la Figura 20 se muestra dicha arquitectura de la red neuronal propuesta en [29]. Esta estructura puede dividirse en una primera fase que contiene la estructura con capas LSTM y luego una segunda fase con capas de *dropout*, normalización y activación.

Finalmente, el segmento de postprocesado es nulo. La salida del sistema presenta un archivo MIDI que se aleja de la idea final del proyecto de generar una canción.

4.1.2 Fallos y partes a mejorar para la siguiente iteración

Uno de los fallos del modelo es a nivel estructural de la red neuronal, la organización de tres capas LSTM con solo una de *dropout* después puede causar más *overfitting* del esperado. También al tener una falta de un codificador al principio, las notas pueden carecer de contexto. Para la siguiente iteración se buscó mejorar la arquitectura general para poder ofrecer mejores resultados, adicionalmente se tuvo en cuenta el desarrollo de un postprocesado.

4.2 Segunda Iteración

En esta segunda iteración se trabajó sobre el contenido de la anterior para poder ofrecer mejoras, se implementó un codificador simple desde cero que no conseguía codificar correctamente las entradas. Además, también se modificó ligeramente la estructura de la red y se creó un postprocesado simple sin introducir ningún instrumento.

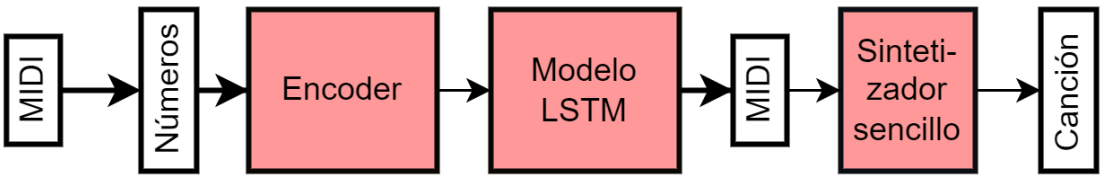

Figura 21: Esquema general del comportamiento de la segunda iteración

En la Figura 21 se puede ver la arquitectura de la segunda iteración, en comparación con la anterior, hay elementos como el codificador o un postprocesado (el sintetizador sencillo). Importante darse cuenta de la conversión a números que se hace a la entrada. El color rojo de los elementos es para representar el mal funcionamiento de estos. Se va a explicar a con más detalle a continuación. Resultados de esta iteración se pueden ver en el Anexo 3 y Anexo 4.

4.2.1 Diferencias con respecto a la anterior iteración

Con respecto a la anterior iteración, las mejoras fueron a nivel estructural en comparación con el nivel de resultados. Primero se consiguió introducir las notas directamente como listas de números en vez de como palabras, se consiguió introducir el tipo de notas de pausa y que la salida produjese la siguiente nota (en vez del índice en el diccionario). Todas estas inclusiones conllevaron a un modelo que junto con las modificaciones de la red y el postprocesado no conseguía crear melodías que fuesen generalmente armónicas.

4.2.2 Fallos y partes a mejorar para la siguiente iteración

A nivel de estructura todo tenía sentido y se podría esperar mejoras a la hora de la generación, pero en el momento de la práctica los resultados mostraban

una salida distinta. Es por esto que se buscó cambiar drásticamente el esquema general y revertir algunos de los cambios para poder conseguir buenos resultados. En el segmento del postprocesado se avanzó consiguiendo crear un instrumento digital con una onda sinusoidal que producía un sonido decente.

4.3 Tercera Iteración

Esta tercera y final iteración construye sobre el conocimiento adquirido del desarrollo de las dos anteriores. Fijándose mejor en el enfoque del procesado de lenguaje natural, se profundizó en las mejoras presentes en esa rama de la inteligencia artificial e intentó reformular el progreso de la anterior para que se ajustase a este nuevo cambio de paradigma.

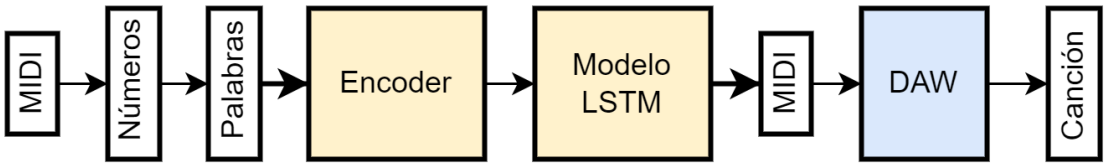

Figura 22: Esquema general del comportamiento de la tercera iteración

En la Figura 22 se puede ver el esquema general de la tercera iteración. Como se puede ver mantiene la estructura de la segunda iteración, pero se añade el elemento de la primera del paso a palabras. La distinción de colores es para diferenciar de la fase de generación de notas del postprocesado (DAW). A continuación, se explica más en profundidad. Resultados de esta tercera iteración se pueden ver en el Anexo 5 y Anexo 6. Existe también un repositorio de *GitHub* con el código en [1].

4.3.1 Diferencias con respecto a la anterior iteración

De la primera iteración se mantuvo el paso de notas a números, y se añadió adicionalmente el paso a palabras para luego crear un vocabulario. Como la estructura anterior no estaba dando resultados se buscó un modelo que funcionase bien en el ámbito del procesado de lenguaje. Se mantuvo y cambió la idea de usar un codificador que acabó resultando en la implementación del *word2vec*. También se adaptó la arquitectura del modelo basada en distintas ideas que se pueden ver en esta rama. También en el segmento de postprocesado se consiguió implementar el DAW programable que permitía aplicar los instrumentos explicados previamente. Todas estas mejoras consiguieron que se pudiese generar mejores melodías en comparación con las dos anteriores iteraciones.

[1] Repositorio en: https://github.com/MR-KARPIN/lofi-transoformer

4.3.2 Fallos y mejoras propuestas para futuras iteraciones

Los problemas identificados y posibles mejoras en el modelo actual se dividen en los tres segmentos principales:

- En el preprocesado: surgen fallos relacionados con la pérdida de información al convertir notas a números, como la falta de pausas y la pérdida de datos adicionales del archivo MIDI original. También la introducción secuencial de canciones sin separadores puede generar cambios bruscos en la secuencia. Esta perdida de información se puede arreglar para futuras iteraciones y añadir una separación en canciones para eliminar los cambios bruscos.

- En la red neuronal: la cantidad de datos de entrenamiento utilizados se resulta insuficiente para obtener información relevante sin caer en el sobreajuste. En cuanto a la generación de notas hay una dificultad para "aprender" ciertas armonías no presentes en el set de entrenamiento debido a que el modelo se limita a generar notas o acordes que ya haya visto antes, cosa que restringe su respuesta. Se podría continuar viendo si alguna otra arquitectura resulta mas óptima para el problema y buscar más datos de entrenamiento para reducir el sobreajuste.

- En el postprocesado: se destaca la falta de variedad musical debido a la presencia de un solo instrumento melódico, una base rítmica y un efecto de sonido de lluvia. Esta limitación hace que haya solo un BPM (por la base rítmica) que puede producir una sensación repetitiva entre las canciones. Sería cuestión de añadir más variedad musical y/o una interfaz para poder personalizar qué elementos extra suenan.

5 Resultados

En el siguiente apartado se mostrarán los dos experimentos realizados para evaluar el modelo propuesto. Estos dos modelos buscan analizar la calidad del resultado a nivel musical (evaluación "subjetiva") y los recursos computacionales que requiere este (evaluación "objetiva"). Se refiere de manera "subjetiva" y "objetiva" a ambos debido a la medición no representable numéricamente de la interacción entre un humano y su información generada del primero y la capacidad de trasladar las mediciones a números del segundo.

5.1 Evaluación del aspecto subjetivo del resultado final

La evaluación del aspecto "subjetivo" de los elementos generados, ha sido realizada con un test de Turing [30]. Este ha consistido en 15 preguntas divididas en tres segmentos de 5 grupos de 2 canciones en la que cada segmento contenía 10 fragmentos musicales de 10, 15 y 20 segundos respectivamente. A continuación, se mostrarán el planteamiento y resultados de este.

5.1.1 Planteamiento del Test de Turing

Para recoger los datos del test de Turing se ha utilizado la herramienta de *Google Forms*. Consiste en la comparación de dos canciones en la que una de ellas ha sido generada con el modelo y la otra obtenida del conjunto de entrenamiento. Debido a la poca disponibilidad de archivos MIDI del género "*Lo-Fi*" se tomó la decisión de utilizar la misma información con la que se ha entrenado el modelo. Una vez obtenidos los 30 archivos MIDI (15 humanos y 15 generados) todos se pasan por la línea de postprocesado del proyecto para que estén en igualdad de condiciones. Adicionalmente a las 15 preguntas, al final de cada grupo de 5, se pregunta si han tenido que repetir alguna de las canciones o si han sido capaces de contestar antes de terminar de escuchar alguna de las canciones. Estas dos preguntas ayudan a poder determinar si la diferencia entre una melodía generada y una humana es más sutil o se pueden diferenciar directamente. Al final de este cuestionario también se pregunta si han sido capaces de poder detectar algún patrón entre las canciones generadas y las humanas, elemento que también puede ayudar a explicar en qué partes falla o hace bien el modelo (en comparación con la generación humana).

5.1.2 Resultados del Test de Turing

Tras el planteamiento del test, este se puso en marcha consiguiendo 27 respuestas en total. Nueve de las cuales no contienen las preguntas adicionales a las 15 principales debido a la introducción de estas preguntas mas tarde del lanzamiento inicial de la encuesta. Esto nos deja con 19 respuestas con contenido adicional para ver que sutilezas hay en la diferencia entre las canciones generadas y las realizadas por humanos.

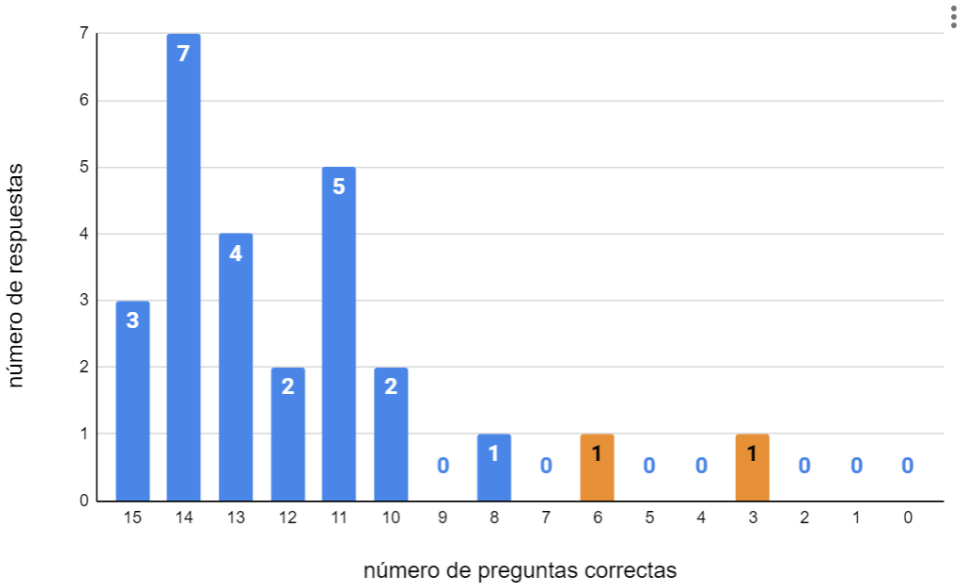

Figura 23: Número de respuestas según número de preguntas correctas

Como se puede ver en la Figura 23, la distribución de las respuestas resulta conglomerada el sector superior (rango del 8 al 15) encontrando su moda en el 14 con 7 respuestas. De media se ha obtenido un 11,048 sobre 15, teniendo en cuenta que el caso óptimo sería que de media resultase 7,5 (mostrando que los participantes responden sin encontrar la diferencia entre humano y generado) se ha puesto de distinto color los valores menores a 7,5.

Siendo 0 el 7,5 y 10 el 15, de media se obtiene un 4,731, factor que nos indica que por muy poco (0,3) es más probable que sea indistinguible que distinguible. Debido a que no es una diferencia muy notable y que hay casos como la respuesta con 3 preguntas correctas que pueda implicar un fallo de entendimiento del enunciado, no se puede llegar a la conclusión de que los resultados producidos por el sistema pasan el test de Turing.

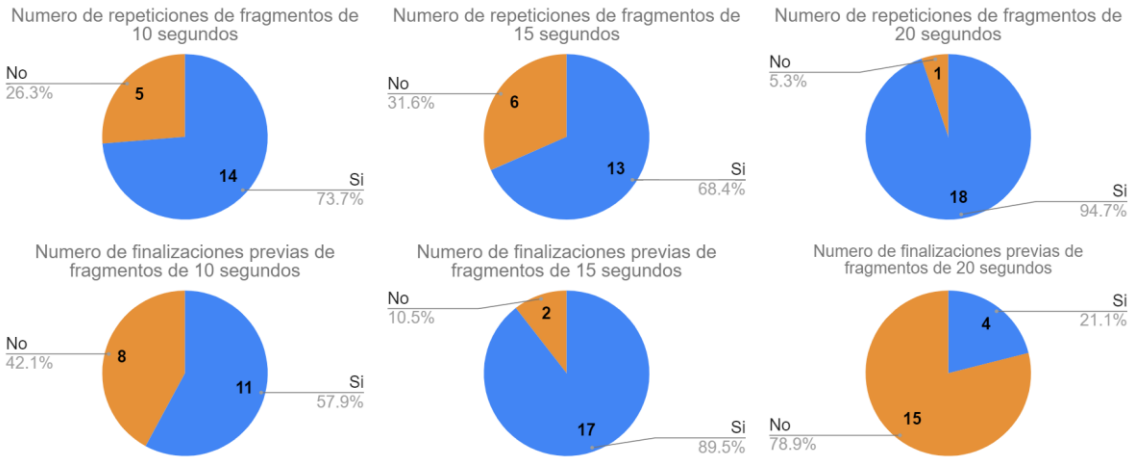

Figura 24: diagramas sectoriales del número de respuestas a las preguntas "¿Has necesitado repetir algún fragmento?" y "¿Has sido

capaz de contestar antes de terminar algún fragmento?" dividido según los tres segmentos

Para aportar de más contexto a los resultados, se van a analizar las preguntas adicionales realizadas. En la Figura 24, se puede ver la distribución de las respuestas a lo largo de los tres segmentos y muestra información relevante sobre cada segmento:

- En los fragmentos de 20 segundos se repiten mucho las pistas de audio, pero se saltan en gran parte, esto revela que la diferencia es más sutil (por eso se repite) pero en zonas más específicas (por eso no se termina de escuchar el fragmento al completo).

- En el segmento de 15 segundos se escuchan hasta el final la gran mayoría de fragmentos musicales y se repiten en gran parte, mostrando una posible mayor dificultad para discernir en este segmento.

- En menor medida, pasa lo mismo en los fragmentos de 10 segundos que presenta las mismas cualidades.

Finalmente, donde también se puede conseguir información, es en la última pregunta: "¿Has identificado algún patrón que creas que te haya permitido identificar cual era el auténtico?". Ante esta, se han pronunciado diferentes respuestas, pero se pueden ver dos distintos patrones detectados por los encuestados:

1. Diferencias armoniosas: Los fragmentos humanos mostraban un sentido armonioso mientras que los fragmentos generados podrían mostrar una falta de relación entre las notas. Este fallo podría ser cuestión de un mal rendimiento a la hora de entrenar.

2. Diferencias rítmicas: Como en el modelo no se consideran las pausas, todas las canciones generadas suponen una secuencia de notas una detrás de la otra. Por el contrario, los fragmentos humanos contenían pausas para marcar un nivel de ritmo más complejo.

5.2 Evaluación del aspecto objetivo del proceso

Con respecto a la evaluación "objetiva" del proyecto, se busca analizar el rendimiento de este a nivel de ejecución. Como parte de la finalidad de este trabajo es el análisis de modelos sencillos para la generación de música "Lo-Fi", este apartado gana importancia puesto que nos permite medir cuanto de sencilla es nuestra red neuronal. Adicionalmente también entra dentro de la medición el tiempo de preprocesado y postprocesado.

5.2.1 Medición temporal

La primera métrica que se va a considerar es la medición del tiempo de ejecución sobre un procesador Intel Core i5-9600K de 3.7GHz. Esta medición ha sido realizada gracias a la biblioteca *timeit* de Python que permite registrar el tiempo que se tarda en la llamada a un método, para esta medición se ha registrado el método general que llama a todo el *pipeline*. Debido a que se pueden realizar distintas longitudes de canciones, este factor cambia el tiempo de ejecución. Por ello, se han tenido en cuenta cuatro casos para realizar la medición.

- La generación de 2 notas y el procesado de 1 segundo de renderizado: Para intentar acotar cuanto se tarda en ejecutar toda la parte externa a la generación de notas y renderizado, se ha intentado medir una pequeña cantidad de notas y segundos con tal fin. De media se tardan alrededor de 5,41 segundos en realizar este procesado.

- La generación de 20 notas y el procesado de 15 segundo de renderizado: Otra longitud a tener en cuenta es la creación de un fragmento corto de 15 segundos debido a la naturaleza de repetir una melodía corta inherente al género "*Lo-Fi*" y a que es la longitud de los ejemplos de prueba para la evaluación del apartado 5.1. La media de procesado resulta en 11,74 segundos. Unos 6,33 segundos por encima de la base que se puede inferir de la medición anterior, son tomados por el procesado de las notas y renderizado.

- La generación de 120 notas y el procesado de 100 segundo de renderizado: Esta longitud trata de representar la duración general que puede tener una canción. Nótese que normalmente se generan más notas que segundos, esto es debido a que la relación notas/segundo dado el BPM y el *offset* entre notas ronda por el rango de 1,05 a 1,1. La media de procesado resulta en 41,22 segundos, con un supuesto tiempo de procesado solo de notas y sintetización de 35,81 segundos. Tardando 5,65 veces más en comparación con el anterior cuando el número de notas y tiempo resulta ser mayor que 6 veces más. Esto permite plantear la hipótesis de que a mayor tiempo de ejecución, menos tarda por nota y segundo.

- La generación de 1.000 notas y el procesado de 1.000 segundos de renderizado: Para poder probar la hipótesis planteada en la medición anterior se ha buscado intentar llegar a números que exceden el uso normal que se haría del sistema. Como el BPM y *offset* no parecen tener un gran impacto en el procesado, se ha igualado a 1.000 el tiempo y el número de notas para simplificar el proceso. La media de procesado resulta en 535,05 segundos. Eliminando el tiempo que no es de procesado queda 529,64. En comparación con el segundo experimento es 50 veces más información, pero tarda 83,67 veces más en procesar. En comparación con el tercer experimento es unas 9 veces mayor, pero tarda 14,79 veces más. Demostrando que la hipótesis planteada durante la anterior medición es falsa.

Para intentar dar una explicación a la diferencia entre el tercer y cuarto experimento, se ha dividido el tiempo de procesado entre la generación de notas

y el renderizado. Para 120 notas se tarda de media unos 13,59 segundos mientras que para el renderizado unos 31,72 segundos. Se puede ver que el tiempo de renderizado ocupa un 70% del procesado. A diferencia del experimento cuatro que el tiempo para la generación de notas tiene de media 219,13 segundos y el de renderizado 302,38 segundos. El porcentaje del renderizado frente al tiempo total supone un 58% indicando que el tiempo de procesado a niveles más bajos resulta más costoso en comparación con números mucho más grandes.

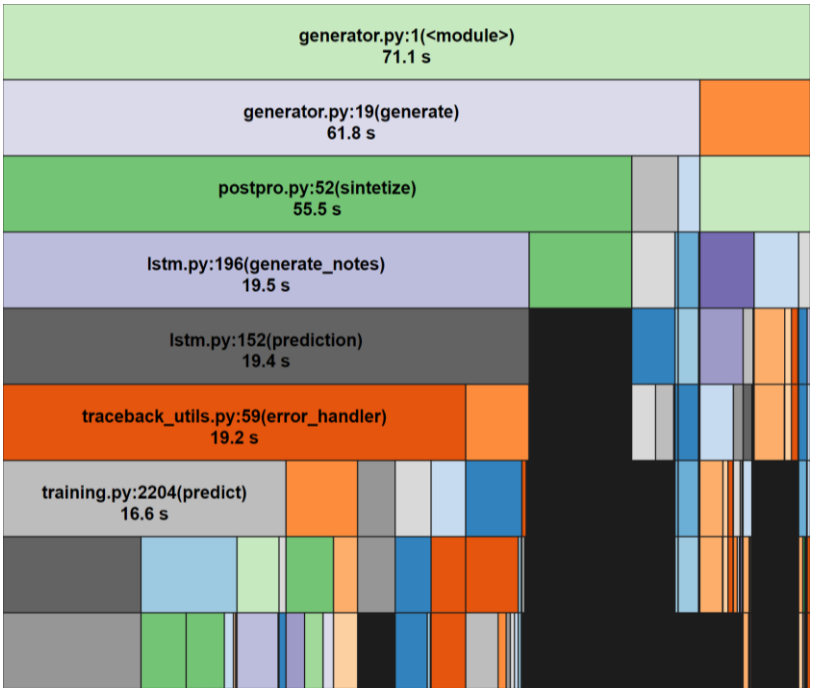

Figura 25: Diagrama de la distribución de tiempo de ejecución presentado con la herramienta *SnakeViz* para el procesado de 20 notas y renderizado de 15 segundos

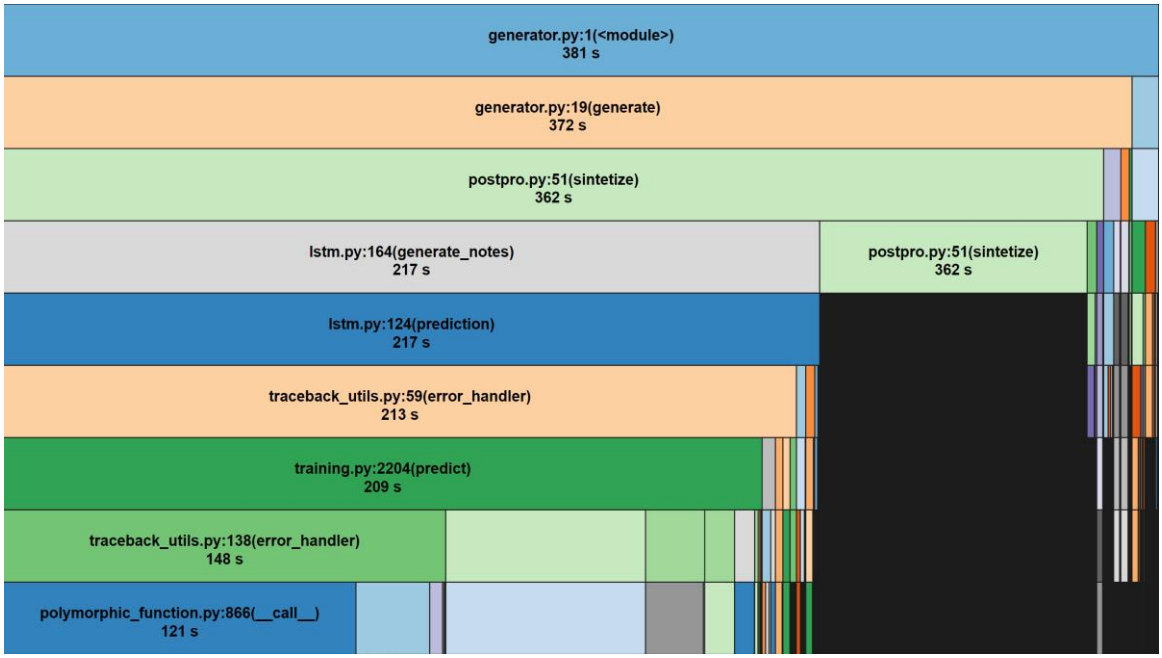

Figura 26: Diagrama de la distribución de tiempo de ejecución presentado con la herramienta *SnakeViz* para el procesado de 1.000 notas y segundos

De manera adicional para ver la división de tardanza en el tiempo, se ha utilizado el módulo *cProfile* para la medición y *snakeViz* para la visualización del resultado. A continuación, se puede ver en la Figura 25, la visualización de la distribución temporal en la generación de 20 notas y procesado de 15 segundos de duración. También, se ha medido la distribución durante la generación de 1.000 notas y renderizado de 1.000 segundos mostrado en la Figura 26 el eje X (vertical) representa los distintos procesos mientras que el eje Y (horizontal) muestra el tiempo que han tardado. En este segundo ejemplo se puede ver mejor la diferencia temporal que tiene la generación y el renderizado; como se puede ver en la cuarta fila (que contiene *lstm.py* y *postpro.py*) gran parte del tiempo lo ocupa el método *generate_notes* que tiene como función generar la cantidad especificada de notas y guardarla en un archivo, el resto del tiempo se lo lleva la parte de renderizado (el resto de la función *sintetize*).

5.2.2 Uso de Memoria

Para la medición de memoria se ha utilizado la biblioteca *tracemalloc*. Esta nos permite ver el uso de memoria pico durante la ejecución del programa. Los resultados muestran 314.596.000 *Kilobytes* (o unos 314 MB) para el procesado de 20 notas y 15 segundos, 429.490.967 *KB* para el procesado de 120 notas y 100 segundos y 1.336.886.847 *KB* (1,336 GB)para el procesado de 1000 notas y segundos.

Esta progresión muestra que a niveles normales (siguiendo la suposición de que una canción dura unos pocos minutos) se ocupa menos de unos 400 MB pero que a niveles más elevados puede escalar bastante alto llegando a 1,3GB. Aun así viendo la relación entre la segunda medición (120 notas y 100 segundos) y la tercera medición (1000 notas y segundos) vemos que aunque sea 10 veces más en proporción, ocupa solamente 3 veces más de memoria.

6 Conclusiones

En este capítulo se verán las conclusiones obtenidas del desarrollo y evaluación del sistema, donde veremos el cumplimiento de los objetivos junto con los fallos actuales del modelo.

6.1 Análisis de cumplimiento de objetivos

A continuación, en este apartado se va a proceder a explicar el cumplimiento de los objetivos que se introdujeron en el apartado 1.2:

1. Aprendizaje de los sistemas de generación de audio para el óptimo diseño de modelos aplicando técnicas utilizadas por el estado del arte: Como se ha podido ver en el estado del arte, el estudio de los distintos acercamientos que se han tenido al problema de generación de música han sido considerados, llevando a la decisión de crear un modelo capaz de generar melodías musicales en formato MIDI.

2. Diseño estructural del modelo aplicando los conocimientos adquiridos del estudio de la literatura: El estudio de ésta, además, ha profundizado en las distintas ramas de la inteligencia artificial que podían servir de utilidad para resolver el problema presentado (generar una canción "Lo-Fi"). Finalmente se ha decidido por seguir un diseño guiado por las mejoras en el campo del procesado del lenguaje natural.

3. Implementación y entrenamiento del modelo para su evaluación en comparación con otros acercamientos al problema: Durante la fase de diseño y resultados se ha podido ver la implementación y ejecución del modelo entrenado que presenta las características analizadas en los dos apartados que preceden a este. La comparación con respecto a otros acercamientos se tiene en cuenta con el análisis "subjetivo" del modelo. El test de Turing realizado compara la generación mediante inteligencia artificial con la humana y pone a prueba cómo de competente puede resultar.

6.2 Fallos del modelo actual

Durante el desarrollo de esta memoria se han podido atisbar algunos indicios de los fallos del modelo actual. Igual que se ha hecho con el diseño, los fallos pueden dividirse en tres principales segmentos:

- Preprocesado: En lo que respecta a la introducción de notas al modelo, existen varios fallos que se pueden encontrar. Todos los fallos vienen de la pérdida de información que ocurre en la fase de codificación de notas a números. El caso más importante es la pérdida de un tipo de nota, las pausas. Adicionalmente también se pierde información sobre el archivo MIDI original como puede ser el *tempo* o información extra sobre las notas

como puede ser la intensidad. También la manera de procesado general de introducción de las canciones es de manera secuencial (una detrás de la otra sin ningún tipo de separador), haciendo así también al modelo entender que puede haber cambios bruscos en mitad de la secuencia (porque coge el segmento entre dos canciones). Otro problema resulta la cantidad de datos de entrenamiento a disposición, para el entrenamiento del modelo actual se han utilizado 103 fragmentos de canciones "Lo-Fi" obtenidos de distintas fuentes de internet. En términos de entrenamiento de inteligencia artificial, esta cantidad resulta escasa para poder extraer información relevante sin caer en el *overfitting*.

- Generación de notas: En este segundo segmento de la línea de procesado general, los fallos principalmente pueden provenir del diseño de la red neuronal. El problema más importante que puede tener es el *overfitting* anteriormente mencionado y algún posible problema a la hora de "aprender" ciertas armonías del set de entrenamiento. Otro aspecto también a tener en cuenta, son las limitaciones provenientes del acercamiento similar al del procesado del lenguaje. Debido a que se genera un vocabulario con todas las notas vistas en la fase de entrenamiento, el modelo en sí no puede crear notas (o acordes) que no haya visto antes, limitando así su respuesta. También existe el problema heredado del segmento anterior que es la falta de pausas dentro de su vocabulario, produciendo así notas en secuencia una después de la otra sin ninguna pausa entre medias.

- Postprocesado: Los fallos del modelo presentes en este tercer segmento vienen a raíz de la falta que puede haber de variedad musical. Actualmente solo existe un instrumento para la melodía, una base rítmica y un elemento adicional que es el efecto de sonido de lluvia. Debido a que solo se tiene una base rítmica implica en la existencia de un único un BPM posible para la canción, produciendo así una sensación no deseada de repetitividad entre canciones.

7 Discusión

En este último capítulo de la memoria, se llevará a cabo una ampliación de varios elementos con el objetivo de fomentar una conversación en torno al proyecto. Se presentarán diversas cualidades y fallos identificados durante el desarrollo, así como los posibles sesgos que podrían estar presentes en él. El propósito de esta ampliación es lograr una visión más completa y objetiva del proyecto, promoviendo un análisis crítico y constructivo.

7.1 Puntos fuertes y débiles

En esta primera parte, que sirve como una extensión de los resultados y conclusiones previamente presentados, se llevará a cabo una recopilación exhaustiva de los factores positivos y negativos asociados al sistema.

7.1.1 Puntos fuertes

A continuación, se presentan los diversos factores positivos que se encuentran presentes en el sistema:

1. Alta modularidad en el código: El desarrollo del proyecto ha llevado a la creación de un código altamente modular, lo que facilita la reutilización y división de distintos segmentos. Esta modularidad no solo permite un desarrollo futuro más sencillo y facilidad para experimentar nuevas arquitecturas de una forma ágil, sino que también promueve la eficiencia y la escalabilidad del sistema.

2. Velocidad de procesamiento: La evaluación "objetiva" del modelo ha demostrado que el sistema propuesto logra una velocidad de procesamiento adecuada. Esto implica que el sistema puede generar resultados en tiempos razonables, lo que incluso podría permitir el procesamiento continuo de canciones sin interrupciones significativas.

3. Diseño simple y encapsulado: Tanto la cantidad de líneas de código como los elementos involucrados en el sistema son fácilmente encapsulados. Esta simplicidad en el diseño facilita la comprensión del funcionamiento de cada parte del sistema y su interacción con las demás, lo que contribuye a un desarrollo más eficiente y una depuración más sencilla de posibles errores.

4. Potencial de mejora y desarrollo continuo: Aunque el sistema actual permite generar canciones con un solo clic, existe un amplio potencial que se puede explorar y desarrollar aún más. Como se menciona en las contribuciones futuras, hay numerosas tareas y mejoras que se pueden implementar para perfeccionar y ampliar la propuesta existente. Esto brinda oportunidades emocionantes para seguir avanzando y refinando el sistema.

7.1.2 Puntos débiles

A continuación, se procede a enlistar los distintos factores negativos identificados en el sistema:

1. Limitada variedad musical: El sistema actual se caracteriza por la presencia de un solo instrumento, una base rítmica y un elemento añadido. Esta configuración limitada restringe la creatividad musical, lo que resulta en una producción musical con una variedad limitada en términos de composición y estructura.

2. Escasez de datos de entrenamiento: La cantidad de datos utilizados para entrenar el sistema ha sido limitada. Esto impide que el sistema pueda demostrar su verdadero potencial. Además, debido a la falta de datos, el sistema solo puede generar notas o acordes previamente vistos, lo que dificulta la generación de nuevas ideas musicales y limita la exploración de nuevos patrones.

3. Valoración de la arquitectura para melodías más complejos: La arquitectura elegida para el sistema puede no ser la más adecuada para abordar problemas más complejos y con conjuntos de datos más extensos. La falta de pruebas en escenarios más desafiantes y con una mayor cantidad de datos genera incertidumbre sobre la eficacia del sistema a mayor escala.

4. Posible sobreajuste de la red neuronal: Existe la posibilidad de que la red neuronal haya copiado patrones existentes en lugar de aprender las relaciones subyacentes entre ellos. Este sobreajuste puede ser resultado de la escasez de datos de entrenamiento, una capacidad de procesamiento excesiva, un tiempo prolongado de entrenamiento o una combinación de estos factores. Como consecuencia, el sistema puede presentar dificultades para generalizar y producir resultados originales y creativos.

7.2 Sesgos del trabajo realizado y oportunidades

Tras analizar los aspectos favorables y desfavorables del sistema, resulta pertinente explorar los errores sistemáticos que podrían haber surgido durante su desarrollo, así como las diversas oportunidades que se derivan de todo lo expuesto. Es importante destacar que estos errores son inherentes a la propia estructura y funcionamiento del sistema, y se presentan de manera sistemática.

7.2.1 Sesgos

Dentro de los resultados vistos del sistema, se pueden ver ciertos factores que pueden afectar a la salida de estos y se escapan por estar presentes de manera sistemática. A continuación, vamos a analizar con más detalle estos sesgos que se presentan durante el desarrollo del sistema:

1. Sesgos humanos: Uno de los sesgos más prominentes es la influencia del conocimiento previo sobre el género específico del "*Lo-Fi Hip-Hop*" en la manera en que se ha determinado qué es considerado "Lo-Fi" y qué no lo es. Esta influencia puede afectar tanto la salida del programa como otros resultados porque existe una opinión sesgada de lo que es “Lo-Fi”.

2. Sesgos de los datos de entrenamiento: Además, al realizar el proceso en su totalidad, se encontró que solo se disponía de 103 ejemplos de archivos MIDI de música "Lo-Fi", lo que significa que la definición de este género se basa únicamente en esa pequeña cantidad de información. Este limitado conjunto de datos también puede contribuir a ciertos sesgos en los resultados obtenidos.

7.2.2 Oportunidades

Después de identificar los diversos sesgos presentes en el sistema, se abren varias oportunidades para abordarlos de manera efectiva. A continuación, se presentan estas oportunidades:

1. Realizar un análisis exhaustivo del género "Lo-Fi": Es fundamental realizar una investigación más profunda sobre el género "Lo-Fi" que permita comprender y abordar todos los subgéneros y estilos presentes dentro de él. Esto ayudará a tener una visión más completa y precisa del género, lo que permitirá generar resultados más acordes con cada subgénero específico.

2. Aumentar la cantidad y variedad de datos de entrada: Para superar la homogeneidad y los sesgos presentes, es necesario ampliar tanto la cantidad como la diversidad de los datos utilizados durante el entrenamiento del sistema. Esto implica incluir fuentes de distintos artistas musicales del género "Lo-Fi", lo que proporcionará una perspectiva más amplia y representativa del estilo musical.

8 Aportaciones Futuras

Viendo los fallos presentes en el proyecto actual , en el futuro, se podría intentar resolver estos e intentar mejorar otros fragmentos del sistema para aumentar su eficiencia. Al igual que el apartado anterior, se puede dividir en tres segmentos:

- Preprocesado: La principal mejora que se puede realizar al sistema es la reducción de perdida de información presente actualmente. En el futuro se podría realizar un análisis en profundidad de la información transmitida por un archivo MIDI para poder saber que elementos tener en cuenta. El primer paso para esto sería realizar una división en canciones de entrenamiento para que no entienda que la frontera entre una y otra es una progresión musical valida. Una vez realizada la división entre canciones, se podría añadir la nota de pausa para que la red entienda en que momentos parar. Todas estas mejoras solo pueden llegar a ser tan buenas como el conjunto de entrenamiento, por lo que se debería hacer con igual o superior esfuerzo, una búsqueda de más material de entrenamiento para poder aportar variedad y mayor cantidad de información.

- Generación de notas: Viendo las mejoras realizadas en el segmento del preprocesado, se podría amoldar una arquitectura de la red de manera óptima para el caso, si se termina obteniendo más información más variada, se puede progresar a modelos de procesado de lenguaje más complejos o específicos. Otra opción es la reconstrucción total del modelo a uno que intente predecir que notas suenan correctamente en vez de que palabra puede ser la siguiente. Este modelo aumentaría mucho en complejidad y requeriría de más tiempo de desarrollo.

- Postprocesado: En este tercer segmento el progreso que se puede realizar es abordando el fallo principal de falta de opciones. Se podría realizar una selección de bases rítmicas junto con una interfaz que permita elegir si se quieren añadir o eliminar efectos como el ruido de vinilo o la lluvia de fondo. Adicionalmente, también se pueden añadir otros efectos o instrumentos.

8.1 Propuesta alternativa

Si se fuese a realizar el mismo proyecto de distinta manera, se podría optar por utilizar otros elementos presentados en el estado del arte. Sobre todo, sería cambiando dos decisiones fundamentales: el acercamiento a la generación de música y/o la arquitectura. A continuación, se muestran las dos opciones:

- Modelo de procesamiento directo del archivo: Como se ha visto en el estado del arte, podemos hacerlo procesando la canción entera en vez de archivos de notación (MIDI), este acercamiento, aunque más complejo, podría conseguir resultados más cercanos a las canciones “lo-Fi” creadas por humanos.

- Aplicación de un Transformer: para poder proveer al modelo de más contexto a nivel temporal se podría añadir una arquitectura basada en Transformers. Esta arquitectura podría ser entrenada previamente sobre canciones generales de piano para que "aprenda" conceptos como la musicalidad y luego entrenar con los datos de *"Lo-Fi"* para que especifique más en el género.

9 Análisis de Impacto

En este capítulo se realizará un análisis del impacto potencial de los resultados obtenidos durante la realización del proyecto en diferentes contextos, el contexto social y económico se recogen dentro del cultural y empresarial respectivamente:

- Personal: A nivel personal, este proyecto me permitirá demostrar el asentamiento de mis conocimientos en el campo de la inteligencia artificial, mostrando también mis capacidades para poder realizar el desarrollo e investigación de una propuesta de proyecto del mismo calibre.

- Cultural: La zona donde puede tener más impacto este proyecto es a nivel cultural. Con el creciente crecimiento de aplicaciones desarrolladas con inteligencia artificial en la vida cotidiana, las implicaciones de estas se han puesto en duda [31]. Con el recibimiento de herramientas como *ChatGPT* [21] o *DALL-E* [22] hemos podido ver qué impacto puede tener un proyecto de este estilo. El caso que más se puede similar al objeto presentado en esta tesis es el de *DALL-E* por cuestión de generación de elementos artísticos (pintura en este caso y música en el del proyecto). Así, podemos ver que si se llega a desarrollar un modelo más poderoso que el presentado pueden surgir movimientos en contra de este como la polémica surgida por la introducción de arte generado por inteligencia artificial a la red de arte *ArtStation* [32]. Aunque pueda parecer todo efectos perjudiciales, también existen beneficios de la creación de un modelo como este, principalmente puede servir para la generación de canciones sin derechos de autor que son buscadas entre creadores de contenido, utilización de estas canciones generadas para tener de fondo mientras se estudia o relaja al igual que se utilizan las canciones del género realizadas por humanos y finalmente como herramienta de los artistas que desean crear una canción del estilo y necesitan una base creativa desde donde empezar.

- Empresarial: El uso de una instancia de este proyecto a nivel empresarial puede servir para lo mismo que los beneficios a nivel cultural. Si en algún momento un sector como puede ser el de *marketing* necesita generar un contenido en forma de video o audio, puede servir de apoyo para generar una melodía de fondo sin tener que realizar el gasto adicional de pagar derechos de autoría. A nivel de autónomo o empresas relacionadas con la industria musical, puede servir como herramienta creativa para la producción musical como se ha mencionado antes.

- Medioambiental: El impacto medioambiental producido de poner en práctica este proyecto es el mismo impacto ambiental que produzca la generación de electricidad necesaria para ello. El desarrollo de fuentes de energías sostenibles reducirá el impacto que tiene la generación de electricidad y por ende el de este. En lo que respecta al gasto energético depende de varios factores del *hardware* como el procesador y la fuente de alimentación.

10 Bibliografía

[1] A. Birruin, «La app de consumo con el crecimiento más rápido de la historia de Internet es una IA,» Febrero 2023. [En línea]. Available: https://www.larazon.es/tecnologia/app-consumo-crecimiento-usuarios-mas-rapido-historia_2023020263dbf6c3bf44120001aa96ee.html.

[2] R. A, «Bill Gates vaticina que la inteligencia artificial arrasará a los dos mayores imperios de internet,» Mayo 2023. [En línea]. Available: https://www.abc.es/tecnologia/informatica/soluciones/bill-gates-vaticina-inteligencia-artificial-arrasara-dos-20230523010902-nt.html.

[3] A. Technologies. [En línea]. Available: https://aiva.ai/.

[4] P. Dhariwal, H. Jun, C. Payne, J. W. Kim, A. Radford y I. Sutskever, *Jukebox: A generative model for music,* 2020.

[5] L. Girl, «lofi hip hop radio - beats to relax/study to,» 2017. [En línea]. Available: https://www.youtube.com/watch?v=jfKfPfyJRdk.

[6] «Lo-fi (música),» Wikipedia, 2023. [En línea]. Available: https://es.wikipedia.org/wiki/Lo-fi_(música).

[7] A. Harper, «Lo-Fi Aesthetics in Popular Music Discourse,» Oxford University, UK, 2014.

[8] L. Girl, «Blissful Dreams [sleep lofi],» 2023. [En línea]. Available: https://www.youtube.com/watch?v=ldDtjQkLsss.

[9] Knowsum, «The Hommage,» 2023. [En línea]. Available: https://open.spotify.com/track/6zNwFCceDEZGFEv6s1Itsh?si=f9723281d092419f.

[10] OpenAI, «MuseNet,» Abril 2019. [En línea]. Available: https://openai.com/blog/musenet.

[11] C.-Z. A. Huang, I. Simon y M. Dinculescu, «Music Transformer: Generating Music with Long-Term Structure,» Google AI, 2018. [En línea]. Available: https://magenta.tensorflow.org/music-transformer.

[12] Larnii, [En línea]. Available: https://larnii.com/home/.

[13] P. Barreau, D. Shtefan, V. Barreau, O. Hecho, A. Zakharyan, B. Frey, C.-H. Liu, S. garcia, H. Erdogan, T. Anders, N. Kiefer y A. Sigman, «AIVA, the Artificial Intelligence composing emotional soundtrack music,» AIVA Technologies, 2016. [En línea]. Available: https://www.java.com/es/.

[14] R. Ihaka y R. Gentleman, «The R Project for Statistical Computing,» R Core Team, 1993. [En línea]. Available: https://www.r-project.org/.

[15] G. v. Rossum, «Python,» Python Software Foundation, 1991. [En línea]. Available: https://www.python.org/.

[16] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, Kopf, reas, E. Yang, Z. DeVito, M. Raison, A. Tejani y Chila, «PyTorch: An Imperative

Style, High-Performance Deep Learning Library,» de *Advances in Neural Information Processing Systems*, Curran Associates, Inc., 2019.

[17] [En línea]. Available: https://se.ewi.tudelft.nl/desosa2019/chapters/pytorch/#fnref:3.

[18] F. Chollet, «Keras,» 2015. [En línea]. Available: https://keras.io/.

[19] [En línea]. Available: https://se.ewi.tudelft.nl/desosa2019/chapters/keras/.

[20] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser y I. Polosukhin, «Attention is all you need,» 2017.

[21] J. Schulman, B. Zoph, C. Kim, J. Hilton, J. Menick, J. Weng, J. F. C. Uribe, L. Fedus, L. Metz, M. Pokorny, R. G. Lopes, S. Zhao, A. Vijayvergiya, E. Sigler, A. Perelman, C. Voss y M. Heato, «Introducing ChatGPT,» OpenAI, 2022. [En línea]. Available: https://openai.com/blog/chatgpt.

[22] J. Belgum, D. Cummings, J. Gordon, C. Hallacy, S. Jain, J. Jang, F. Kelton, V. Kuo, J. Lehman, R. Lim, B. Martin, E. Morikawa, R. Nayak, G. Powell, K. Rijshouwer, D. Schnurr, M. Simens y K. Stan, OpenAI, 2015. [En línea]. Available: https://openai.com/product/dall-e-2.

[23] Runway, CompVis y S. AI, «Stable Diffusion,» Stability AI, 2022. [En línea]. Available: https://stablediffusionweb.com/.

[24] «Github Copilot,» Github, 2023. [En línea]. Available: https://github.com/features/copilot.

[25] D. Dambrin, «FL Studio,» Image-Line Software, 1998. [En línea]. Available: https://www.image-line.com/.

[26] «Reaper Digital Audio Workstation,» Cockos, 2006. [En línea]. Available: https://www.reaper.fm/.

[27] D. Braun. [En línea]. Available: https://github.com/DBraun/DawDreamer.

[28] «LABS Soft Piano,» Spitfire Audio, 2022. [En línea]. Available: https://labs.spitfireaudio.com/soft-piano.

[29] Zachary, Noviembre 2020. [En línea]. Available: https://ai.plainenglish.io/building-a-lo-fi-hip-hop-generator-e24a005d0144.

[30] A. Turing, «Prueba de Turing,» 1950. [En línea]. Available: https://es.wikipedia.org/wiki/Prueba_de_Turing.

[31] «A European approach to artificial intelligence,» Comisión Europea, Enero 2023. [En línea]. Available: https://digital-strategy.ec.europa.eu/es/policies/european-approach-artificial-intelligence.

[32] R. Díaz, «Miles de artistas protestan en ArtStation contra las imágenes generadas por inteligencia artificial,» El Mundo, Diciembre 2022. [En línea]. Available:

https://www.elmundo.es/tecnologia/creadores/2022/12/16/639c4306fdddfff37f8b4595.html.

11 Anexos

(Anexo 1)

https://drive.google.com/file/d/1u7aNggyKq76gMDe2tQsYILHvaA2vlx8b/view?usp=sharing

(Anexo 2)

https://drive.google.com/file/d/1ONa7cOaMcNvaEkFCIo_wfGTQUOb1IKSP/view?usp=sharing

(Anexo 3)

https://drive.google.com/file/d/1o-50fVWo6c3UMZHIZ6lIkbnzjxrqDEy8/view?usp=sharing

(Anexo 4)

https://drive.google.com/file/d/1zNmEQ8dNQNcuxIrh7HpGOewUNlarAFaj/view?usp=sharing

(Anexo 5)

https://drive.google.com/file/d/1BKwzfWF72Xq6A6EHuUbjO1xoWjyZSqxc/view?usp=sharing

(Anexo 6)

https://drive.google.com/file/d/1XcxnkZwYbCffbiSbuOO7ZXanc3x29z7m/view?usp=sharing

Este documento esta firmado por

	**Firmante**	CN=tfgm.fi.upm.es, OU=CCFI, O=ETS Ingenieros Informaticos - UPM, C=ES
	Fecha/Hora	Tue May 30 22:52:24 CEST 2023
	Emisor del Certificado	EMAILADDRESS=camanager@etsiinf.upm.es, CN=CA ETS Ingenieros Informaticos, O=ETS Ingenieros Informaticos - UPM, C=ES
	Numero de Serie	561
	Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)