GRADO EN INGENIERÍA EN TECNOLOGÍAS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

GENERADOR DE MELODIAS MEDIANTE REDES NEURONALES

Autor: Claudio Esteban Quesada

Director: Miguel Ángel San Bobi

Madrid

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

Generador de melodías mediante redes neuronales

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el

curso académico 2022/2023 es de mi autoría, original e inédito y

no ha sido presentado con anterioridad a otros efectos.

El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido

tomada de otros documentos está debidamente referenciada.

Fdo.: Claudio Esteban Quesada Fecha: 04/07/2023

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Firmado por SANZ BOBI MIGUEL ANGEL - ***6599** el día 04/07/2023
con un certificado emitido por AC FNMT Usuarios

Fdo.: Miguel Ángel Sanz Bobi Fecha: 04/07/2023

GRADO EN INGENIERÍA EN TECNOLOGÍAS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

GENERADOR DE MELODIAS MEDIANTE REDES NEURONALES

Autor: Claudio Esteban Quesada

Director: Miguel Ángel San Bobi

Madrid

Agradecimientos

A mi familia, por confiar en mí, apoyarme y guiarme siempre.

A mis amigos de la universidad, por darme su apoyo y momentos increíbles durante cuatro años.

A mi tutor de este proyecto, Miguel Ángel Sanz, por su ayuda y guía durante este proyecto.

GENERADOR DE MELODIAS MEDIANTE REDES NEURONALES

Autor: Esteban Quesada, Claudio
Director: Sanz Bobi, Miguel Ángel
Entidad Colaboradora: ICAI – Universidad Pontificia Comillas

RESUMEN DEL PROYECTO

En este proyecto de Trabajo de Fin de Grado (TFG), se ha desarrollado una aplicación web basada en modelos de redes neuronales para la generación de música artificial. Se ha implementado un modelo de C-RNN-GAN, entrenado con archivos MIDI para lograr aprender patrones musicales y generar nuevas muestras originales. Se ha realizado un estudio de distintos modelos obtenidos para llegar al óptimo. Los resultados han demostrado la capacidad del modelo para crear música original y diversa. Se ha creado una interfaz de usuario mediante el desarrollo de una aplicación web que permite a los usuarios generar y reproducir las muestras generadas con acceso a distintos métodos de compresión de las muestras generadas.

Palabras clave: Música artificial, redes neuronales, generación y entrenamiento, aplicación web.

1. Introducción

Este proyecto, tiene como objetivo principal es superar las limitaciones de las plataformas de música existentes, las cuales se centran en ofrecer un catálogo finito de canciones basadas en títulos, artistas o estilos musicales predefinidos. El proyecto busca proporcionar una experiencia musical más enriquecedora y personalizada a través de la generación de música artificial basada en las preferencias del usuario. Para lograr esto, se emplean técnicas de redes neuronales y procesamiento de señales de audio. El proyecto tiene como resultado la creación de una aplicación que permite a los usuarios disfrutar de una amplia variedad de pistas musicales continuas y personalizadas, rompiendo así con las restricciones impuestas por las plataformas de música tradicionales.

2. Definición del proyecto

En este proyecto se ofrece una solución a las barreas de las aplicaciones tradicionales, mediante el desarrollo de una aplicación web que utilice técnicas de generación de música artificial basadas en las redes neurales C-RNN-GAN.

El objetivo principal de esta aplicación es ofrecer al usuario la posibilidad de generar, archivar, reproducir y descargar las muestras generadas. En lugar de depender de canciones preexistentes la aplicación generará notas de manera artificial.

Se estudian diversos modelos para obtener el óptimo, analizando el rendimiento de cada modelo mediante diferentes técnicas. Una vez obtenido el modelo óptimo, se integra en una aplicación web, conectando dicha aplicación con una base de datos y utilizando API para realizar distintas consultas, permitiendo el registro de usuarios, compresión y descarga de muestras.

3. Descripción del modelo/sistema/herramienta

El modelo desarrollado es una implementación innovadora de una red neuronal utilizando la arquitectura C-RNN-GAN (Generative Adversarial Network with Convolutional and Recurrent Layers) específicamente diseñada para la generación de secuencias de música artificial. Esta arquitectura combina capas convolucionales para capturar características espaciales en los datos de entrada, como la duración y la intensidad de las notas, con capas recurrentes que modelan la estructura temporal y la relación entre las notas.

El proceso de entrenamiento del modelo desarrollado se basa en el uso de archivos MIDI como datos de entrada, que contienen información detallada sobre las notas, los acordes y otros aspectos musicales. El modelo se ajusta a través de la optimización de hiperparámetros y parámetros clave para su funcionamiento. Estos hiperparámetros se ajustan cuidadosamente para lograr un equilibrio entre la capacidad de generalización del modelo y la precisión en la generación de música.

Además, se han implementado algunas funciones adicionales para mejorar su rendimiento y capacidad de generación de música, las cuales contribuyen a mejorar la calidad de las secuencias de música generadas y a garantizar una mayor coherencia y precisión en los resultados. Al abordar aspectos como la normalización, la eliminación de ruido, la corrección de errores y el preprocesamiento de los archivos

4. Resultados

El modelo obtenido ha logrado reducir de manera significativa el loss durante las etapas de entrenamiento, validación y prueba, lo cual indica que ha aprendido eficientemente los patrones y estructuras de las secuencias de música. Esta reducción del loss es una indicación de que el modelo está mejorando su capacidad para generar música artificial de alta calidad. Además, se ha observado que el modelo obtenido logra un equilibrio entre la precisión y el sobreentrenamiento. Aunque el modelo es capaz de generar muestras precisas y coherentes, evita caer en el sobreentrenamiento, lo cual es crucial para garantizar que las nuevas muestras generadas sean generalizables y no estén limitadas a la reproducción de las muestras de entrenamiento. Estas características destacadas de este modelo se traducen en métricas de precisión sólidas, lo que demuestra su habilidad para generar música artificial de alta calidad con fidelidad a los patrones y estructuras de las muestras originales**.**

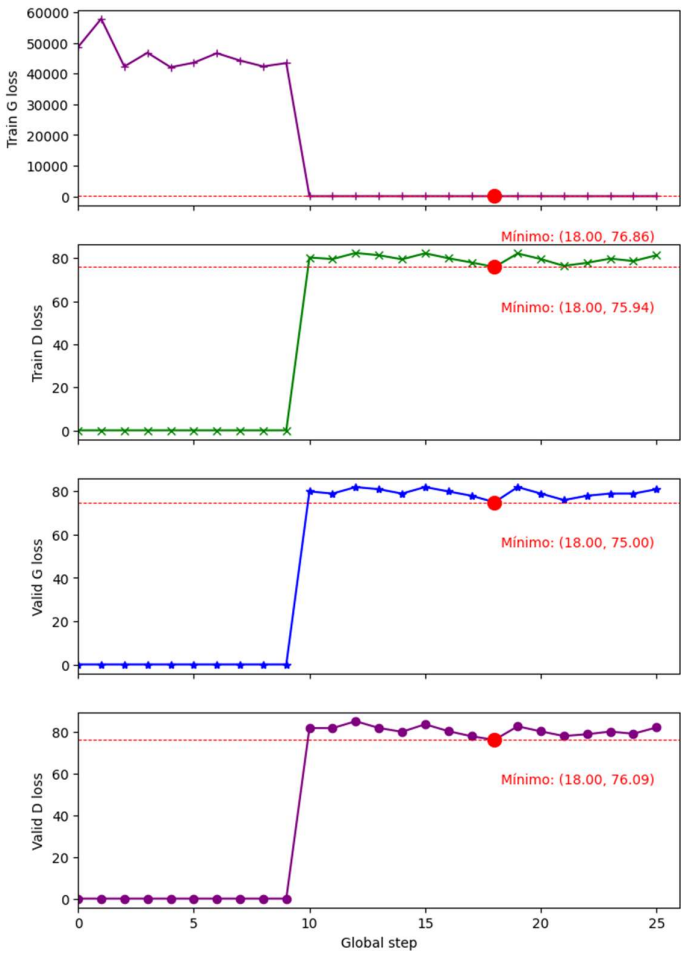

Ilustración 1 – Resultados función de pérdida modelo obtenido

Precisión	**Recall**	**F1-score**
0.793	0.733	0.762

Tabla 1 - Métricas Resultados métricas precisión

5. Conclusiones

El proyecto ha logrado desarrollar un modelo basado en la arquitectura C-RNN-GAN que es capaz de generar secuencias de música artificial de calidad. Este modelo ha demostrado ser efectivo en la reducción de la pérdida durante el entrenamiento y en la generación de música diversa y coherente. Las funciones añadidas, como la normalización de datos, la eliminación de ruido, la corrección de errores y el preprocesamiento de archivos MIDI, han contribuido significativamente a mejorar el rendimiento y la calidad del modelo.

La aplicación web desarrollada proporciona una interfaz intuitiva y amigable para que los usuarios puedan interactuar con el modelo, generar y descargar muestras de música artificial. Además, la capacidad de compartir muestras entre usuarios y la posibilidad de personalizar los hiperparámetros del modelo brindan una experiencia más enriquecedora para los usuarios.

6. Referencias

6.1. C-RNN-GAN: Continuous recurrent neural networks with adversarial training ,Olof Mogren ,Chalmers University of Technology, Sweden http://mogren.one/publications/2016/c-rnn-gan/mogren2016crnngan.pdf

6.2. Generative Adversarial Networks" (GANs) de Ian Goodfellow et al. (2014):https://arxiv.org/abs/1406.2661

6.3. Conditional Generative Adversarial Nets" de Mehdi Mirza et al. (2014) https://arxiv.org/abs/1411.1784

6.4. Long Short-Term Memory" (LSTM) de Sepp Hochreiter y Jürgen Schmidhuber (1997):https://www.bioinf.jku.at/publications/older/2604.pdf

6.5. Angular Development with TypeScript https://www.oreilly.com/library/view/angular-development-with/9781491902218/

MELODY GENERATOR THROUGH NEURAL NETWORKS

Author: Esteban Quesada, Claudio
Director: Sanz Bobi, Miguel Ángel
Collaborating Entity: ICAI – Universidad Pontificia Comillas

ABSTRACT

In this Final Degree Project (TFG), a web application based on neural network models has been developed for the generation of artificial music. A model of C-RNN-GAN has been implemented, trained with MIDI files in order to learn musical patterns and generate new original samples. A study of different models obtained to reach the optimum has been carried out. The results have demonstrated the model's ability to create original and diverse music. A user interface has been created by developing a web application that allows users to generate and play the generated samples with access to different compression methods of the generated samples.

Keywords: Artificial music, neural networks, generation and training, web application.

1. Introduction

This project's main objective is to overcome the limitations of existing music platforms, which are focused on offering a finite catalog of songs based on titles, artists or predefined musical styles. The project seeks to provide a more enriching and personalized music experience through the generation of artificial music based on user preferences. To achieve this, neural network techniques and audio signal processing are used. The project results in the creation of an application that allows users to enjoy a wide variety of continuous and personalized music tracks, thus breaking the restrictions imposed by traditional music platforms.

2. Proyect's definition

This project offers a solution to the barriers of traditional applications, through the development of a web application that uses artificial music generation techniques based on C-RNN-GAN neural networks.

The main objective of this application is to offer the user the possibility to generate, archive, reproduce and download the generated samples. Instead of relying on pre-existing songs, the app will artificially generate notes.

Various models are studied to obtain the optimum, analyzing the performance of each model using different techniques. Once the optimal model is obtained, it is integrated into a web application, connecting said application with a database and using APIs to perform different queries, allowing user registration, compression and download of samples.

3. Model's description

The developed model is an innovative implementation of a neural network using the C-RNN-GAN (Generative Adversarial Network with Convolutional and Recurrent Layers)

architecture specifically designed for the generation of artificial music sequences. This architecture combines convolutional layers to capture spatial features in the input data, such as note duration and intensity, with recursive layers that model the temporal structure and relationship between notes.

The training process of the developed model is based on the use of MIDI files as input data, which contain detailed information about notes, chords, and other musical aspects. The model is adjusted through the optimization of hyperparameters and key parameters for its operation. These hyperparameters are carefully tuned to strike a balance between the generalizability of the model and the accuracy in generating music.

In addition, some additional features have been implemented to improve its performance and music generation capabilities, which help improve the quality of the generated music streams and ensure greater consistency and accuracy in the results. By addressing things like normalization, noise removal, error correction, and file preprocessing.

4. Results

The model obtained has managed to significantly reduce the loss during the training, validation, and test stages, which indicates that it has efficiently learned the patterns and structures of the music sequences. This loss reduction is an indication that the model is improving its ability to generate high quality artificial music. In addition, it has been observed that the model obtained achieves a balance between precision and overtraining. Although the model can generate accurate and consistent samples, it avoids overtraining, which is crucial to ensure that the newly generated samples are generalizable and not limited to reproducing the training samples. These outstanding features of this model translate into robust precision metrics, demonstrating its ability to generate high-quality artificial music with fidelity to the patterns and structures of the original samples.

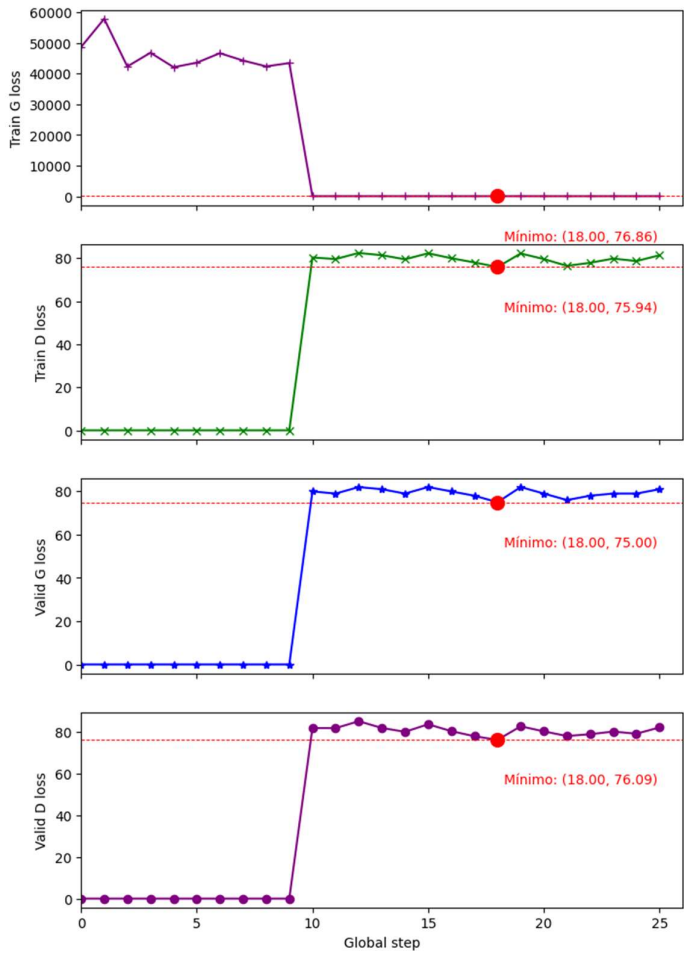

Ilustración 2 - Model loss function's results obtained.

Precision	**Recall**	**F1-score**
0.793	0.733	0.762

Tabla 2 - Metrics Precision metric results

5. Conclusions

The project has succeeded in developing a model based on the C-RNN-GAN architecture that can generate quality artificial music sequences. This model has been shown to be effective in reducing loss during training and in generating diverse and coherent music. Added features such as data normalization, noise removal, error correction, and MIDI file pre-processing have contributed significantly to improved model performance and quality.

The developed web application provides an intuitive and friendly interface for users to interact with the model, generate and download artificial music samples. Additionally, the ability to share samples between users and the ability to customize model hyperparameters provide a richer experience for users.

6. Referencias

6.1. C-RNN-GAN: Continuous recurrent neural networks with adversarial training ,Olof Mogren ,Chalmers University of Technology, Sweden http://mogren.one/publications/2016/c-rnn-gan/mogren2016crnngan.pdf

6.2. Generative Adversarial Networks" (GANs) de Ian Goodfellow et al. (2014):https://arxiv.org/abs/1406.2661

6.3. Conditional Generative Adversarial Nets" de Mehdi Mirza et al. (2014) https://arxiv.org/abs/1411.1784

6.4. Long Short-Term Memory" (LSTM) de Sepp Hochreiter y Jürgen Schmidhuber (1997):https://www.bioinf.jku.at/publications/older/2604.pdf

6.5. Angular Development with TypeScript https://www.oreilly.com/library/view/angular-development-with/9781491902218/

Índice de la memoria

Índice de Ilustraciones

Índice de tablas

Capítulo 1. INTRODUCCIÓN

El proyecto actual forma parte del Trabajo de Fin de Grado (TFG) del grado en Ingeniería en Tecnologías de Telecomunicación de la Universidad Pontificia de Comillas (ICAI). En este contexto, se aborda un desafío en el ámbito de los servicios de música.

En la actualidad, existen numerosas aplicaciones y servicios que ofrecen la posibilidad de escuchar música, ya sea a través de plataformas de streaming o sistemas de radiotransmisión. Sin embargo, la mayoría de estos servicios se basan en algoritmos que recomiendan canciones en función de títulos, artistas o estilos musicales. Esto puede resultar limitante para los usuarios, ya que su experiencia musical se ve restringida a un número finito de canciones disponibles en cada plataforma.

El objetivo principal de este proyecto es superar esta limitación y crear una aplicación innovadora que rompa con los esquemas tradicionales de reproducción musical. La idea es ofrecer a los usuarios una experiencia musical más enriquecedora permitiendo a esto explorar una biblioteca ilimitada de piezas musicales, mediante la generación de muestras de un modelo empleando redes neuronales.

La elección de este tema como objeto de estudio se debe a su relevancia en el ámbito de las señales de audio y la compresión de estas. Estos son campos de investigación continuamente explorados y desarrollados por ingenieros de telecomunicaciones. Además, el proyecto implica el uso de tecnologías avanzadas, como redes neuronales y herramientas de aprendizaje automático, para la implementación de los modelos de generación de música.

Como parte integral del proyecto, se ha considerado la creación de una interfaz de usuario intuitiva y amigable que permita a los usuarios interactuar con la aplicación de manera sencilla y accesible. Además, se exploran técnicas de compresión de audio para optimizar la transferencia y el almacenamiento de las composiciones generadas, ofreciendo así una experiencia completa y fluida para los usuarios.

En resumen, este proyecto busca ofrecer una solución innovadora en el ámbito de la reproducción musical, permitiendo a los usuarios disfrutar de una experiencia sin limitaciones. A través de la generación artificial de notas musicales y el uso de tecnologías avanzadas, se pretende romper con las limitaciones tradicionales de la reproducción musical y ofrecer a los usuarios un abanico de posibilidades mucho más amplio.

Capítulo 2. DESCRIPCIÓN DE LAS TECNOLOGÍAS

En este capítulo, se proporcionará una descripción detallada de las tecnologías, protocolos, herramientas y bibliotecas especificas utilizadas a lo largo del proyecto para facilitar su compresión y seguimiento.

2.1. DESARROLLO DE MODELOS Y GRÁFICAS

El desarrollo de los distintos modelos y su estudio de su validez mediante la visualización de los resultados, son aspectos fundamentales en este proyecto. Para esto se utiliza Python como lenguaje de programación principal además de varias bibliotecas para la implementación y entrenamiento de los modelos.

2.1.1. Python

Este lenguaje de programación se ha convertido en uno de los más utilizados en el campo del aprendizaje automático y procesamiento de datos debido a su versatilidad y la amplia gama de bibliotecas y herramientas disponibles. Entre estas bibliotecas se encuentran varias especializadas en el aprendizaje del lenguaje automático, como TensorFlow, PyTorch o Keras, que proporcionan implementaciones eficientes de algoritmos para estos procesos.

2.1.2. Tensorflow

TensorFlow es una biblioteca de código abierto desarrollada por Google, la cual ofrece una amplia gama de funcionalidades y herramientas usadas en labores de implementación y entrenamiento de modelos de aprendizaje, como son las redes neuronales.

Entre las muchas tareas que permite realizar esta librería, destaca por ofrecer alternativas para desarrollar modelos orientados a tareas de clasificación, regresión o procesamiento de lenguaje natural.

La característica principal usada por los modelos de este trabajo es conocida como TensorBoard. Esta herramienta de visualización permite comprender y analizar los modelos de TensorFlow de manera detallada, ya que permite visualizar las métricas de rendimiento como la función de pérdida a lo largo de las etapas de entrenamiento.

2.1.3. Python Midi

Esta biblioteca de Python desempeña un papel crucial en el procesamiento de datos musicales de este proyecto. Está diseñada para trabajar con archivos MIDI, que son el formato estándar usado para almacenar y representar información musical. Proporciona una amplia variedad de funciones y utilidades que permiten leer, escribir y manipular este tipo archivos.

2.1.4. Gráficos

La representación de diversas funciones y conjuntos de datos son de gran importancia en este proyecto. Para ello se utilizan librerías como Matplotlib y Seaborn para visualizar los resultados de entrenamiento de los modelos, generar gráficas de pérdida y analizar los resultados.

- Matplotlib es una de las bibliotecas de gráficos más utilizadas de Python, debido a su facilidad para crear una variedad de visualizaciones, como gráficos de líneas y diagramas de dispersión. En este proyecto se utiliza dicha biblioteca para representar las curvas de la función de pérdida de los distintos modelos a lo largo de las distintas etapas. Mediante el uso de matplotlib se pueden personalizar dichas gráficas para un análisis más claro y detallado de los datos representados.
- Seaborn es otra biblioteca de gráficos de Python que se utilizar para calcular distintos gráficos como la matriz de confusión, utilizada para el análisis de los distintos modelos. Esta biblioteca ofrece amplia gama de estilos y paletas de colores para ofrecer gráficos más fáciles de interpretar.

2.2. FRONT-END

El front-end de la apliación web se encarga de proporciar una interfaz de usuario interactiva e intuitiva para los usuarios. Para ello se ha desarrollado esta parte utilizando Angular y Ngx-Amdin.

2.2.1. Angular

Angular es un framework de desarrollo de aplicaciones web de código abierto, utilizado ampliamente para la creación de interfaces de usuario interactivas. Utiliza el lenguaje de programación TypeScript, siendo este un superconjunto de JavaScript, para aprovechar las mejores prácticas del desarrollo web.

Una de las principales ventajas, y por lo que es utilizado comúnmente, es que basa su arquitectura en componentes. Este modelo de arquitectura permite dividir la aplicación en componentes reutilizables y facilita el mantenimiento y organización del código. Cada uno de estos componentes se encarga de una parte especifica de la interfaz de usuario y es responsable de su propia lógica, estando compuesto por varios archivos que desempeñan distintos roles en la aplicación:

Ilustración 3 - Lenguajes usados en Angular

- Archivo de componente (.component.ts)

Este archivo contiene la definición del componente en TypeScript, donde se define la clase, lógica, métodos y eventos de dicho componente.

- Archivo de plantilla (.component.html)

Define el diseño del componente en HTML, estableciendo la manera en la que se visualizan los datos y se interactúa con el componente mediante la interfaz de usuario.

- Archivo de estilos (.component.scss)

Este archivo es el encargado de contener las reglas de estilo SCSS que se aplican al componente, donde se definen los estilos visuales y la apariencia de los elementos de cada componente.

- Archivo de pruebas (.component.spec.ts)

Este archivo es utilizado para realizar pruebas unitarias de cada componente en caso de que se quiera probar el correcto funcionamiento e integración de este.

2.2.2. Ngx-Admin

Ngx-Admin es una plantilla de interfaz basada en Angular que proporciona una amplia gama de componentes y características que pueden ser utilizados para el desarrollo y diseño de aplicaciones web.

La principal ventaja de su uso es la adaptación de componentes existentes, como tablas, formularios o paneles de navegación, para la elaboración de componentes propios, agilizando el proceso de diseño de la aplicación.

2.3. BACK-END Y BASE DE DATOS

El back-end de la aplicación web se encarga de gestionar la ejecución del modelo utilizado y proporcionar una interfaz para realizar peticiones HTTP, mediante una API, además de utilizar un base de datos MYSQL para el almacenamiento de información relevante.

2.3.1. APIs mediante Flask

API de Flask es una herramienta que facilita la creación de API (Application Programing Interfaces) en Python.

Las principales ventajas de utilizar Flask para desarrollar las distintas API son su sintaxis clara y concisa y su facilidad para la definición de rutas que manejen las solicitudes y respuestas HTTP. Mediante estas solicitudes se consigue comunicar el front-end de la aplicación con la base de datos de los usuarios y canciones, además de permitir la ejecución del modelo utilizado para la generación de música y la conversión de archivos de música.

Para su uso es necesario instalar la librería Flask de Python y definir las rutas que manejan las distintas peticiones. Finalmente se ha de levantar el servicio de la API mediante la creación de una instancia de aplicación, donde se puede definir el puerto en el que se levanta dicho servicio.

Las peticiones se haran mediante los metodos habituales (GET, POST, PUT, DELETE) y los resultados de las mismas se devuelven en formato JSON, como se muestra en la siguiente figura:

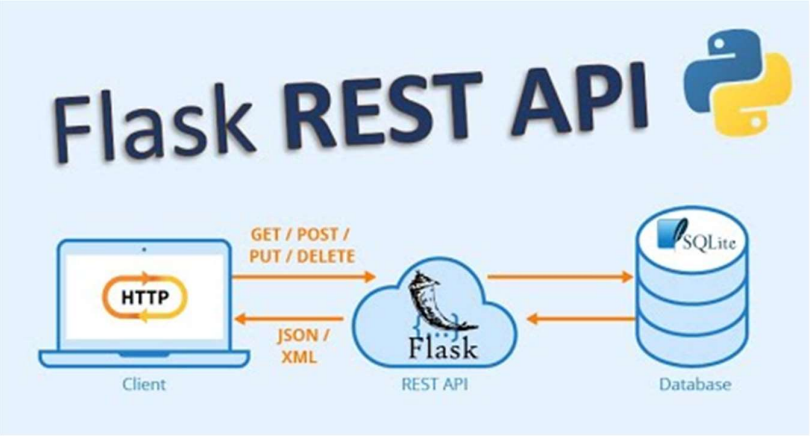

Ilustración 4 – Funcionamiento Flask API

2.3.2. Mysql

Mysql es un sistema de gestión de bases de datos relacionales, comúnmente utilizado en el desarrollo de todo tipo de software, ya que garantiza un entorno seguro para almacenar, administrar y manipular datos.

Este sistema destaca por su escalabilidad y rendimiento, al poder manejar altos volúmenes de datos y permitir realizar consultas de forma rápida y eficiente. Además, ofrece funciones de seguridad como la gestión de usuarios y privilegios que garantizan la integridad y confidencialidad de los datos almacenados.

Adicionalmente, este sistema es compatible con una gran variedad de lenguajes de programación y frameworks, como los utilizados en la elaboración de este trabajo.

Capítulo 3. ESTADO DE LA CUESTIÓN

La utilidad y el desarrollo de las redes neuronales ha supuesto la revolución de números campos en los últimos años, siendo usadas para una amplia gama de tareas desde el reconocimiento de voz hasta el procesamiento de imágenes. En particular, las Redes Generativas Antagónicas (también conocidas por GANs, debido a sus siglas en inglés) han demostrado ser de gran eficacia a la hora de generar una serie de datos originales que imitan un conjunto de datos proporcionados a la entrada. Debido a esto su uso en ámbitos relacionados con el trabajo de secuencias temporales, cono la música o el habla, presentó nuevas alternativas.

En este contexto se produjo la aparición de las Redes Generativas Antagónicas Recurrentes Convolucionales (también llamadas C-RNN-GAN). La característica principal de dichas redes es la capacidad de combinar la potencia de las redes GANs con las Redes Neuronales Convolucionales (CNNs) las cuales tienen gran utilidad a la hora del procesamiento de imágenes, debido a su estructura de convolución, lo que las hace muy eficientes para tratar datos de entrada de alta dimensionalidad y Redes Neuronales Recurrentes (RNNs), siendo estas muy eficaces a la hora de trabajar con datos secuenciales ,como la predicción de series temporales o el reconocimiento de voz, gracias a que estas cuentan con una memoria de la información que han visto previamente.

Las redes C-RNN-GAN fueron introducidas por primera vez en 2016 por Olof Mogren, autor principal del artículo "C-RNN-GAN: Continuous Recurrent Neural Networks with Adversarial Training" (Redes neuronales recurrentes continuas con entrenamiento adversarial). Este artículo centra su enfoque en la utilización de las redes GANs para generar nuevas secuencias de notas musicales, utilizando las RNNs para capturar la dependencia temporal entre los elementos de secuencia y una CNN para llevar a cabo el procesamiento de cada elemento. A continuación, se describirán cada una de estas redes más a fondo, estudiando su funcionamiento.

3.1. REDES NEURONALES

Las redes neuronales son una categoría de modelos computacionales que toman como inspiración el funcionamiento del cerebro humano a la hora de procesar información. Estos modelos suelen basar su funcionamiento en el aprendizaje profundo y automático, diseñados para aprender a realizar tareas a través de la exposición iterativa a datos.

Estas redes están compuestas por una serie de nodos de procesamientos o "neuronas" las cuales están organizadas siguiendo una estructura de capas. Cada neurona se encarga de tomar un conjunto de entradas, a las que aplica una transformación lineal, seguida de una función de actividad no lineal y produciendo una salida. Las neuronas de una capa están conectadas con neuronas de la siguiente capa mediante un conjunto de enlaces ponderados. Dichas neuronas aprender a realizar su labor de manera correcta mediante el ajuste estos pesos.

En la actualidad, existen varios tipos de redes neuronales, que presentan una estructura y características propias. Entre las más comunes se encuentran las de alimentación directa (feedforward), donde la información se propaga de manera unidireccional, es decir, desde la entrada a la salida y las redes neuronales recurrentes, que permiten la retroalimentación de las conexiones, siendo de gran utilidad para el procesamiento de datos secuenciales. También cabe destacar las redes neuronales convolucionales, caracterizadas por utilizar operaciones de convolución para procesar datos con una estructura de cuadricula, como podría ser una imagen.

El proceso de aprendizaje de una red neuronal implica el ajuste iterativo de los pesos de la red, los cuales son actualizados como respuesta a los datos de entrada. Existen numerosos métodos para guiar este proceso, siendo la función de perdida uno de los más conocidos, ya que cuantifica la discrepancia entre las salidas y los verdaderos valores objetivos.

Pese a ser notablemente flexibles y potentes, siendo capaces de aprender representaciones complejas, pueden presentar desafíos a la hora de entrenar de manera efectiva, debido a problemas como los mínimos locales, el sobreajuste y la dimensionalidad. Sin embargo,

estos inconvenientes suelen ser resueltos a través de técnicas de regularización, inicialización de pesos y algoritmos de optimización.

3.2. ETAPAS REDES NEURONALES

El proceso de puesta en marcha de una red neuronal requiere una serie de pasos esenciales, siendo los principales la fase de preentrenamiento, entrenamiento y testing o prueba.

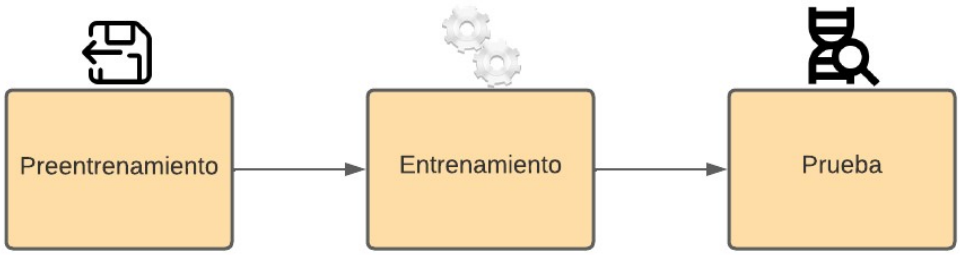

Ilustración 5 - Etapas redes nuronales

Preentrenamiento:

Esta etapa se considera de importancia crítica en el desarrollo de diversas redes neuronales al tener un impacto significativo en el rendimiento del modelo final desarrollado. Esta fase involucra varias actividades preliminares necesarias antes de que la red pueda ser entrenada de manera eficiente.

Algunas tareas que se tienen lugar durante esta etapa es la preparación y procesamiento de los datos de entrada, los cuales deben ser organizados y transformados o normalizados para ser adecuados para el funcionamiento de la red.

Otro componente de gran importancia de este proceso es la inicialización de los parámetros de la red, debido a que las redes neuronales aprenden ajustando iterativamente sus parámetros y el punto de partida puede influir significativamente en el aprendizaje final.

Entrenamiento:

Una vez se ha realizado la fase de preentrenamiento, la red comenzará el proceso de entrenamiento. Durante esta fase, la red neuronal aprende a realizar la tarea deseada ajustando de manera iterativa sus parámetros en respuesta a los datos del entrenamiento.

Esto se realiza mediante el uso de un algoritmo de optimización, que busca minimizar una función de perdida que cuantifica la discrepancia entre las predicciones de la red y los verdaderes valores objetivo.

En este proceso también se busca evitar el sobreajuste, el cual se produce cuando la red aprende a replicar los datos de entrenamiento de tan cerca que se comporta de manera errónea y poco fiable con valores no vistos anteriormente.

Testing o Prueba:

Este proceso se lleva a cabo una vez la red ha sido entrenada en la que se prueba su rendimiento en datos no vistos durante el entrenamiento. Esto proporciona una medida de cómo se espera que la red se comporte en situaciones del mundo real donde los datos de entrada son distintos a los datos con los que se ha llevado a cabo la fase de entrenamiento. Los resultados de esta fase pueden informar sobre los ajustes necesarios y mejoras que se pueden aplicar a la red.

3.3. GANS (REDES GENERATIVAS ANTAGÓNICAS)

Las Redes Generativas Antagónicas o GANS (Generative Adversial Network), se presentaron por primera vez en el trabajo “Generative Adversial Network” por Ian Goodfellow, Yoshua Bengio y Aaron Courville en el año 2014, donde se introducían este nuevo tipo de redes, las cuales serían punteras por su capacidad de generar nuevos datos de entrada imitando un conjunto de datos de entrada.

Las GANs basan su funcionamiento en una estructura de dos componentes principales, un generador y un discriminador, que trabajan en conjunto en un modo competitivo, debido a que ambos son entrados simultáneamente con objetivos opuestos.

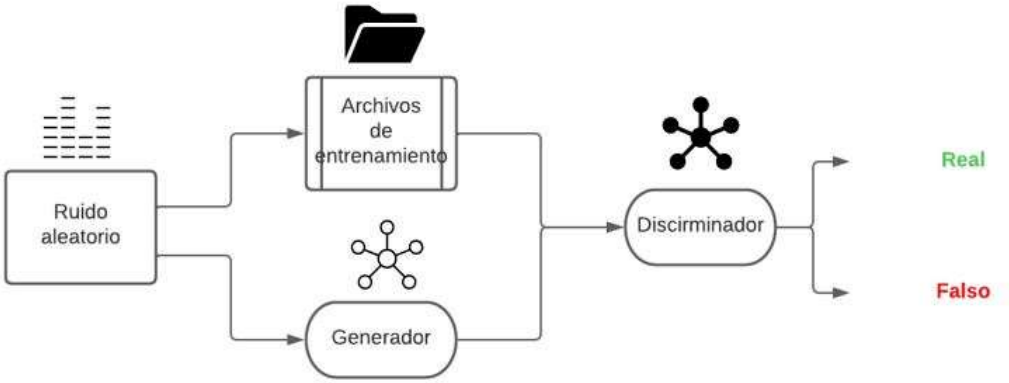

Ilustración 6. Diagrama de Arquitectura de una red GAN

El generador es una red neuronal que tiene como objetivo producir nuevos datos que imiten los datos reales que toma como entrada la red GAN. Su objetivo principal es que dichos datos generados o falsos sean lo más parejos posible a los datos reales que se toman como entrada de la red. En la mayoría de los casos, el generador toma como entrada un conjunto de vectores de ruido aleatorio, transformando estos en datos que intentan imitar la distribución de los datos reales. Dicho proceso de transformación se realiza a través de múltiples capas de la red, aplicando diferentes transformaciones no lineales en cada una de ellas.

El discriminador es considerado como un "crítico" del trabajo de la red GAN. Su objetivo es distinguir entre los datos reales y los datos generados. Este componente suele tratarse de otra red neuronal profunda que toma como entrada un dato (real o generado) y produce a una salida que representa la probabilidad de que dicho dato sea real.

Durante el periodo de entrenamiento, tanto el generador como el discriminador se optimizan de manera alternativa. En una iteración del proceso de entrenamiento, primero se actualizan los parámetros del discriminador, mejorando su funcionamiento para distinguir entre datos reales y generados. Posteriormente, se procede a la actualización de los parámetros del generador con el fin de potenciar su capacidad para engañar al discriminador.

Este proceso se repite hasta que alcanzar un equilibrio donde el generador es capaz de generar datos que resultan indistinguibles de los datos reales y el discriminador no tiene la capacidad para determinar la autenticidad de un dato.

En el marco del entrenamiento adversarial a las GANs, que incorporar una dinámica de competencia bidireccional, ha demostrado su eficacia en la generación de datos sintéticos con una gran verosimilitud. Su utilización se ha extendido a través de una amplia gama de campos, incluyendo, la generación de imágenes, la transferencia de estilos artísticos y la generación de texto y sonido.

No obstante, la implementación de las GANs no está exenta de desafíos. Uno de los problemas es el "colapso de modos", en el cual el generador produce una muestra limitada de salidas, sin tener en cuenta la diversidad de los datos de entrada. Este problema presenta que los datos generados pueden tener un alto grado de similitud entre sí, sin reflejar la variedad existente en los datos reales.

Además las GANs pueden llegar a ser inestables y requieren un de gran cuidado en la selección de hiperparemetros, por lo que para garantizar la estabilidad en el entrenamiento, se han propuesto una seri de técnicas de normalización por lotes (estandarizando los datos de entrada, calculando una media y desviación estándar para aplicar una transformación logrando que los datos tengan una media cercana a cero) , la penalización del gradiente (evitando el sobreajuste del entrenamiento, agregando términos adicionales a la función de pérdida para penalizar la complejidad del modelo) y los métodos de optimización adaptativa (algoritmos utilizados para ajustar los parámetros de un modelo de manera más eficiente, adaptando la tasa de aprendizaje).

Por otra parte, la evaluación de la calidad de los datos generados por las GANs es aún un área activa de investigación. Dado que el objetivo principal del generador es el de confundir al discriminador, la función de pérdida del discriminador por sí sola no es siempre un indicador de calidad de los datos generados.

3.4 RNNs (REDES NEURONALES RECURRENTES)

Las RNNs representan una clase de redes neuronales diseñadas para el manejo de forma eficiente de una serie de datos sequenciales. En contextos donde los datos no son entidades aisladas, sino parte de una secuencia, como pueden ser las series temporales, discursos de lenguaje natural, imágenes y videos, estas redes son de gran utilidad.

La arquitectura de las RNNs se basa en la ide a de un bucle de retroalimentación interno, el cual permite mantener una "memoria" de la información relevante de las iteraciones pasadas. Esta característica es la principal causa por la cual las RNNs puedan incorporar el contexto temporal en su contexto de aprendizaje, de manera que la salida de un instante determinado dependerá del dato de entrada en ese instante y de los datos de entrada previos.

La principal limitación de las RNNs es la capacidad de recordar información de pasos tiempos de mayor antigüedad debido a un problema conocido como el desvanecimiento del gradiente. Este problema ocurre debido a que en el proceso de retroprogación de la red, los gradientes calculados para actualizar los pesos de la red pueden tomar transformase de manera exponencial en valores sumamente pequeños a medida que se propagan por antigüedad en el tiempo, lo que resulta en la aparición de problemas de aprendizaje a largo plazo.

Para abordar este problema se introdujo un nuevo tipo de RNN llamado LSTM (Long Short-Term Memory), las cuales presnetan una mejora significativa sobre las originales debido a la solución del problema mencionando anteriormente.

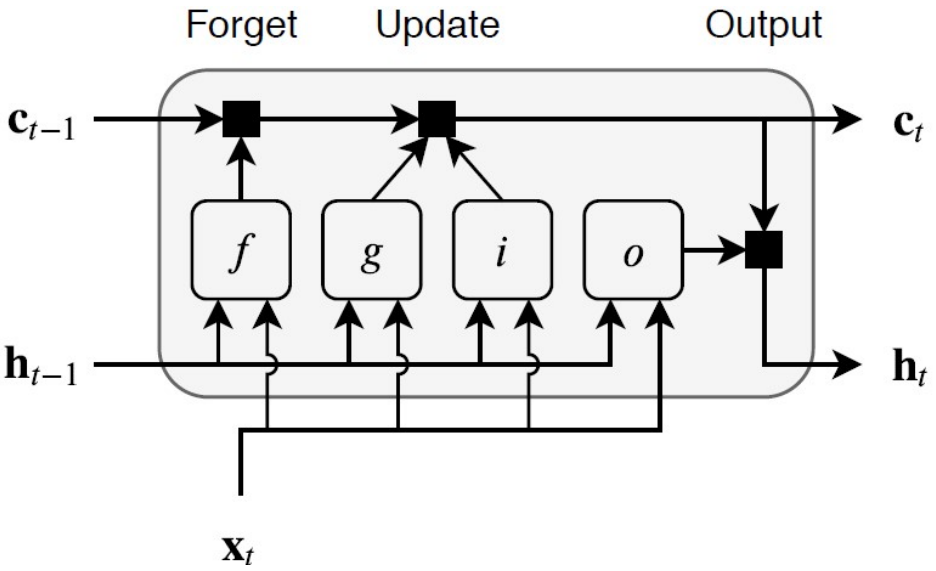

Ilustración 7 - Esquema célula LSTM

Las LSTM logran esto a través de una estructura de celda de memoria compleja que incorpora tres puertas de entrada (input gate), olvido (forget gate) y salida (output gate).

La celda de memoria actúa como núcleo de la célula LSTM. Esta contiene la información de los pasos de tiempo anteriores y puede transportarla a lo largo de la secuencia de datos para influir en etapas posteriores, manteniendo un registro de la información a largo plazo.

La puerta de entrada es la encargada de decidir la cantidad de la nueva información que debe ser almacenada en la celda de memoria, ya sea de la entrada anterior o del estado anterior. Esta tarea se realiza mediante una función sigmoide, que transforma la información de entrada devolviendo valores entre 0 y 1, para cada elemento, asignando el valor 0 si la información no es relevante, por lo que no pasará y el valor 1 a la información que si es relevante.

Ecuación 1- Sigmoide $\sigma(x) = 1 / (1 + e\hat{} - x)$

La puerta de olvido es la encargada de decidir la cantidad de información del estado de la celda actual que debe ser descartada. Este proceso utiliza nuevamente la función sigmoide, asignando el valor 1 a la información que desea mantener y 0 la información que debe ser descartada por ser irrelevante.

Finalmente, la puerta de salida determina la cantidad de información de la celda estado que deber ser utilizada para calcular la salida actual de dicha célula LSTM. Una vez más, se utiliza la función sigmoide a los valores de la entrada y el estado anterior, además de una función tangente hiperbólica, la cual tiene un rango de salida similar a la función sigmoide, siendo este de -1 a 1, al estado actual. Los resultados de ambas funciones son multiplicados para obtener la salida definitiva.

$$tanh(x) = (e\hat{}x - e\hat{} - x) / (e\hat{}x + e\hat{} - x)$$

Las LSTM han sido aplicadas a una amplia gama de problemas que implican datos secuenciales, como la traducción automática de texto y la clasificación de video y audio. A pesar de tener una mayor complejidad añadida en comparación con las RNN originales, las LSTM logran un mejor rendimiento en tareas que requieren un aprendizaje a largo plazo.

Recientemente, han surgido nuevas variantes y extensiones a las LSTM, como las GRU (Gated Recurrent Units), que simplifican las estructuras de memoria y las LSTM convolucionales, las cuales combinan las LSTM con las redes neuronales convolucionales para manejar datos secuenciales con una estructura espacial.

3.5. C-RNN-GAN

Las C-RNN-GAN representan una síntesis de las técnicas discutidas anteriormente, llevando las capacidades de generación de datos de las GANs y el manejo de datos secuenciales de las RNNs a un nuevo nivel de potencial y complejidad de aplicación.

Estas redes se originaron mediante el trabajo de Olof Mogren (2016) y fueron diseñadas con el objetivo de generar secuencias de datos complejas y realistas, siendo el caso del trabajo original, la generación de música, aunque sus principios pueden ser aplicados a cualquier tipo de datos secuenciales.

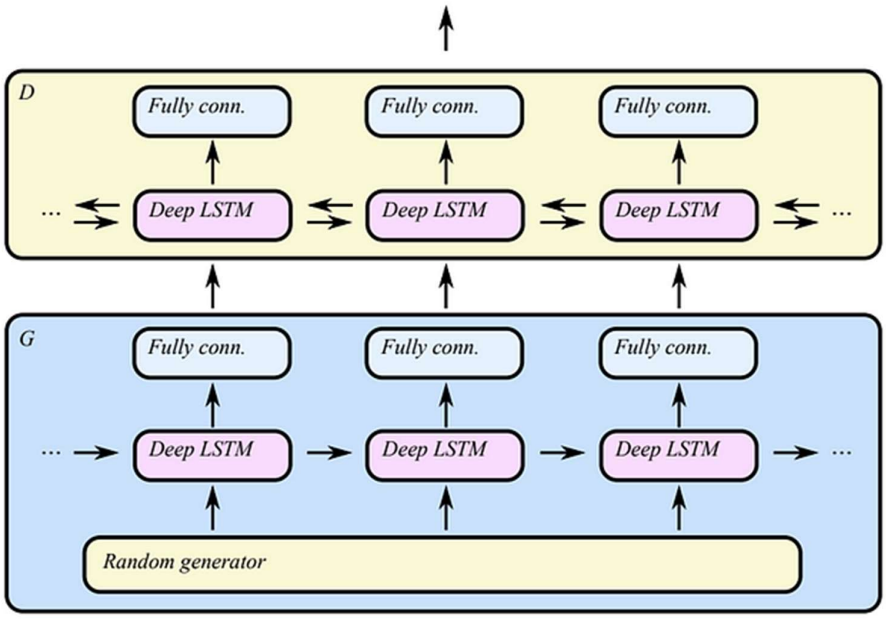

Ilustración 8 - Esquema red C-RNN-GAN

La arquitectura de las C-RNN-GAN consta de dos componentes principales: un generador y un discriminador, como en el caso de las GANs, ambos construidos mediante capas de redes neuronales recurrentes y convolucionales.

Al igual que en las GANs, el generador de una C-RNN-GAN está diseñado para producir secuencias de datos, que idealmente, deberían ser lo mas parejas posibles a las secuencias reales. Para ello, el generador de estas redes incorpora una RNN con células LSTM para modelar las dependencias temporales de los datos. Esta RNN es la encargada de tomar secuencias de valores con ruido como entrada y producir una secuencia de vectores de características que posteriormente es transformada en la secuencia de datos generada por la capa convolucional. Esta capa convolucional es la que permite al generador aprender patrones a nivel e estructura de los datos, al mismo tiempo que la RNN aprende sobre la evolución temporal de dichos patrones.

El discriminador de las C-RNN-GAN, es una combinación de redes neuronales convolucionales y recurrentes, teniendo el mismo objetivo que en las GAN, es decir, diferenciar entre datos generados y reales. Al igual que una GAN tradicional, el discriminador es entrenado para maximizar su capacidad de distinguir entre los datos reales y generados, mientras que el generador es entrenado para maximizar la probabilidad de que el discriminador clasifique sus muestras generadas como reales.

De igual manera que las GANs y las RNNs, las C-RNN-GAN presentan desafíos propios. El entrenamiento de estas redes es delicado y requiere de un correcto ajuste de los hiperparámetros del modelo, con el objetivo de evitar la inestabilidad del modelo desarrollado.

3.6. FUNCIÓN DE PÉRDIDA EN C-RNN-GAN

La función de pérdida, o función objetivo, en una red generativa adversaria como se trata las C-RNN-GAN, es una medida cuantitativa de la efectividad del modelo al realizar su tarea. Al tratarse de redes que incorporan a las GANs, las C-RNN-GAN tienen dos funciones de

pérdida, asignado una al generador y la restante al discriminador. Ambos componentes tienen como objetivo el minimizar el resultado de dicha función.

El genenerador (G) tratará de producir datos indistinguibles de los reales, mientras que el discriminador (D) tratará de distinguir entre los datos reales y los generados. De esta manera, el generador (G) trata de maximizar la posibilidad que el discriminador (D) clasifique sus muestras como reales al mismo tiempo que el discriminador (D) intenta maximizar su precisión en la clasificación correcta de las muestras.

En el caso de las C-RNN-GAN, las funciones de pérdida para el generador (LG) y el discriminador (LD) se calculan de la siguiente manera:

Función pérdida del generador (LG)

Esta función representa la esperanza matemática del logaritmo de uno menos la probabilidad de que el discriminador clasifique las muestras generadas como reales, calculada sobre las muestras generadas, con la siguiente expresión matemática:

$$LG = 1/m \sum[i = 1, m] \, log(1 - D(G(z\hat{}(i))))$$

siendo m el número de muestras generadas, G(z^(i)) la i-ésima muestra que ha sido generada y D(G(z^(i))) la probabilidad de que el discriminador clasifique dicha i-ésima muestra como real.

Función de pérdida del discriminador (LD):

Esta fórmula representa la esperanza matemática de la suma del logaritmo de la probabilidad de que tanto las muestras reales, como las generadas, sean clasificadas correctamente como reales y generadas, respectivamente y el logaritmo de uno menos la probabilidad de que las muestras generadas sean clasificadas como reales. Esta función se calcula sobre todas las muestras, tanto reales como generadas, siendo su expresión matemática:

$$LD = 1/m \sum[i = 1, m] \, (-log \, D(x\hat{}(i)) - log(1 - D(G(z\hat{}(i)))))$$

Siendo x^(i) la i-ésima muestra real.

Ambas funciones de pérdida son utilizadas por el modelo para ajustar los pesos de la red mediante backpropagation y un algoritmo de optimización como el descenso del gradiente, ajustando dichos pesos con el objetivo de minimizar las pérdidas.

Estas funciones de perdida presentan un impacto directo en la calidad de los datos generados. Un generador con una baja función de pérdida será capaz de generar muestras parejas a las muestras reales. Del mismo modo, un discriminador con una baja función de pérdida ajustará el modelo de manera correcta para obtener unos resultados más cercanos a los objetivos.

3.7. C-RNN-GAN POR OLOF MOGREN

Este estudio propone un modelo generativo que trabaja con datos secuenciales continuos aplicándolo al entrenamiento de una colección de música clásica en archivos MIDI de libre disponibilidad para obtener muestras generadas totalmente originales.

Se propone una arquitectura de red neuronal recurrente, C-RNN-GAN, que se entrena mediante un proceso adversarial para modelar la probabilidad conjunta de una secuencia y generar secuencias de datos originales.

En este estudio se modela la señal con cuatro escarales en cada punto de datos: longitud del tono, frecuencia, intensidad y tiempo transcurrido desde el anterior tono. De esta manera, se logra que la red modelice los datos permitiendo que sean representados en acordes polifónicos (con cero tiempos entre dos tonos).

Los datos del entrenamiento se recopilaron en formato MIDI, siendo de gran utilidad al reconocer eventos de tipo “nota encendida”, cuando suena dicha nota, guardado en el archivo MIDI junto a su duración, tono, intensidad (velocidad) y tiempo desde el comienzo del anterior tono.

Como conclusiones, dicho estudio demostró una creciente complejidad en la música generada por la red C-RNN-GAN a medida que avanza el proceso de entrenamiento. Además, se observó que el número de tonos únicos utilizados tiene una tendencia creciente, mientas que la consistencia de la escala se estabiliza en torno a las diez o quince épocas de entrenamiento.

Capítulo 4. DEFINICIÓN DEL TRABAJO

En esta sección se describe el trabajo desarrollado en varios apartados:

- Justificación del desarrollo del trabajo
- Objetivos del trabajo
- Metodología para lograr los objetivos
- Planificación seguida para el trabajo

4.1. JUSTIFICACIÓN

El continuo avance en el campo de la inteligencia artificial en los últimos años ha proporcionado un amplio espectro de herramientas y técnicas que haciendo posible la aparición de nuevas vertientes para la generación y creación del contenido musical. Entre la gran variedad de redes existentes, las C-RNN-GAN han demostrado ser particularmente prometedoras, al ser capaces de generar muestras de secuencias musicales originales y creativas. Sin embargo, existen aún oportunidades significativas para mejorar y optimizar el rendimiento de dichas redes.

La necesidad de implementar dichas mejoras tiene un interés académico y técnico, pero, además, refleja la demanda del mercado. En la actualidad hay una creciente cantidad de aplicaciones y servicios que proporcionan música al usuario, siendo la gran mayoría de estos servicios de pago y con una biblioteca limitada de canciones. También existen aplicaciones que utilizan la inteligencia artificial para la generación de música, pero a menudo estas aplicaciones carecen de funcionalidades clave que permitan al usuario experimentar de manera fácil e intuitiva las capacidades plenas de una red como las C-RNN-GAN.

Por lo tanto, en este proyecto se busca mejorar el rendimiento y la eficiencia de la red C-RNN-GAN desarrollada por Olof Mogren a través de la obtención de variaciones del modelo usado, mediante la variación de los hiperparámetros, introducción de nuevas funciones y análisis de los resultados de cada modelo. Además, se desarrollará una aplicación web que

permita a los usuarios interactuar de forma intuitiva con el modelo y obtener canciones generadas originales.

Debido a esto, este proyecto tiene el potencial de llenar un vacío existente en el sector de las redes neuronales, al mismo tiempo que empuja los límites de la generación de lo que es posible en la generación de música mediante el uso de este tipo de redes.

4.2. OBJETIVOS

Este trabajo tomo como inspiración el proyecto sobre las C-RNN-GAN llevado a cabo por Olof Mogren. Utilizando como base dicho proyecto los objetivos de este trabajo de fin de grado han sido los detallados a continuación.

4.2.1. Necesidad de optimizar la C-RNN-GAN a través de la variación de hiperparámetros

En particular, la variación de hiperparámetros de una red neuronal, como la C-RNN-GAN, puede ser una manera clave de mejorar la calidad de las muestras generadas por el modelo. Dichos hiperparametros, desempeñan un papel crucial en el entrenamiento y la eficacia de los modelos. Entre estos hiperparametros se pueden encontrar la ratio de aprendizaje, la profundidad de la red, el número de unidad en cada capa, entre otros, teniendo todos ellos un impacto significativo sobre la capacidad del modelo para aprender y generar muestras a partir de los datos de entrada.

El ajuste de dicho hiperparametros es un proceso complicado y de larga duración ya que requiere una exploración cuidadosea y sistemática de cada uno de ellos. Este proyecto tiene como cometido la obtención de un modelo mejorado, en comparación con el original, mediante la realización de experimentos para encontrar las configuraciones óptimas.

4.2.2. Potencial para mejorar la calidad de la música generada con nuevas funciones

Aunque la C-RNN-GAN ya desarrollada es capaza de generar música que es convincente para el oído humano, existen oportunidades para mejorar la calidad de la música generada y la velocidad de entrenamiento mediante la introducción de nuevas funciones en la red y el modelo, mejorando la generación de la red y coherencia y musicalidad de las muestras producidas.

Se han estudiado por lo tanto los aspectos de posibles mejoras del modelo, desarrollando dichas funciones para logra dichos objetivos.

4.2.3. Desarrollo de una interfaz de usuario intuitiva

Se define como objetivo el desarrollo de una interfaz de usuario intuitiva mediante la elaboración de una aplicación web, para permitir a los usuarios interactuar con el modelo de manera sencilla. El objetivo de este desarrollo es que el usuario, teniendo conocimiento o no sobre las C-RNN-GNN, pueda generar música mediante el uso del modelo desarrollado. Para ello se desarrollará la interfaz visual usando Angular y se utilizará una aplicación flask para simular el funcionamiento de un servidor, facilitando la formulación de request en modo de API.

4.2.4. Creación de una biblioteca para las muestras generadas

Con el fin de permitir a los usuarios guardar y revisitar las muestras de música generadas, se fija como objetivo la creación de una biblioteca dentro de la aplicación web. Esta biblioteca permite a los usuarios registrados almacenar las muestras que han generado usando el modelo y revisitarlas cuando lo deseen.

4.2.5. Implementación de funcionalidades de compresión y descarga de muestras de música

Para que los usuarios puedan disfrutar de la música generada más allá de la aplicación web, se fija la implementación de funcionalidades de compresión y descarga de las muestras de

música generadas. El modelo actual produce muestras en formato midi, por lo que se ha realizado el estudio de formas de transformar dichas muestras en otros formatos (.WAV, .MP3, .FLAC), facilitando al usuario elegir el formato deseado en función a sus necesidades.

4.3. METODOLOGÍA

El proceso de desarrollo y trabajo de este proyecto se dividirá en tres bloques principales, en los que se realizaran múltiples tareas en cada uno de ellos. Se desarrollarán distintas tareas simultáneamente, pese a pertenecer a distintos bloques, debido a la relación de dichas tareas.

Las tareas principales serán las siguientes:

1. **Estado del arte**

Se lleva a cabo un análisis exhaustivo del estado del arte en el campo de la generación de música mediante Redes Generativas Antagonistas Convolucionales Recurrentes (C-RNN-GAN). Este estudio permite entender mejor las técnicas y métodos utilizados, identificar las áreas de mejora y establecer un marco de referencia para el desarrollo posterior.

2. **Instalación del entorno de trabajo**

Se instala y configura el entorno de trabajo necesario para el proyecto, que incluye las herramientas de desarrollo y las librerías requeridas.

3. **Funcionamiento de aplicación existente**

Durante esta etapa se estudia el funcionamiento de la aplicación actual, la forma en la que se asignan los hiperparámetros de la red, bloques del proyecto y limitaciones y la posibilidad de introducción de mejoras.

4. **Ajuste de hiperparametros e introducción de nuevas funciones**

Se lleva a cabo la fase de implementación de mejoras. Esta fase se dedica a aplicar las mejoras identificadas en la fase anterior a la C-RNN-GAN. Esto implica la experimentación

con la variación de hiperparámetros, la introducción de nuevas funciones y la medición y evaluación del impacto de estas modificaciones en el rendimiento de la red.

5. **Creación de la interfaz gráfica**

Se diseña y crea una interfaz de usuario para el sistema. Esta interfaz permite a los usuarios interactuar con la C-RNN-GAN de una manera intuitiva y accesible, además de generar y escuchar los resultados obtenidos.

Durante esta fase se desarrolla por completo la aplicación web junto al desarrollo del servidor flask y sus interacciones.

6. **Integración del proyecto en la interfaz**

Se integra la C-RNN-GAN y las mejoras implementadas con la interfaz de usuario. Esta fase implica combinar el backend y el frontend del sistema y realizar pruebas exhaustivas para garantizar el correcto funcionamiento del sistema como un todo. Se realizan ajustes y correcciones según sea necesario para garantizar una experiencia de usuario fluida y sin problemas.

4.4. PLANIFICACIÓN

A continuación, se muestra un esquema, de la manera de organización y el tiempo que ha seguido cada una de las tareas principales detalladas anteriormente.

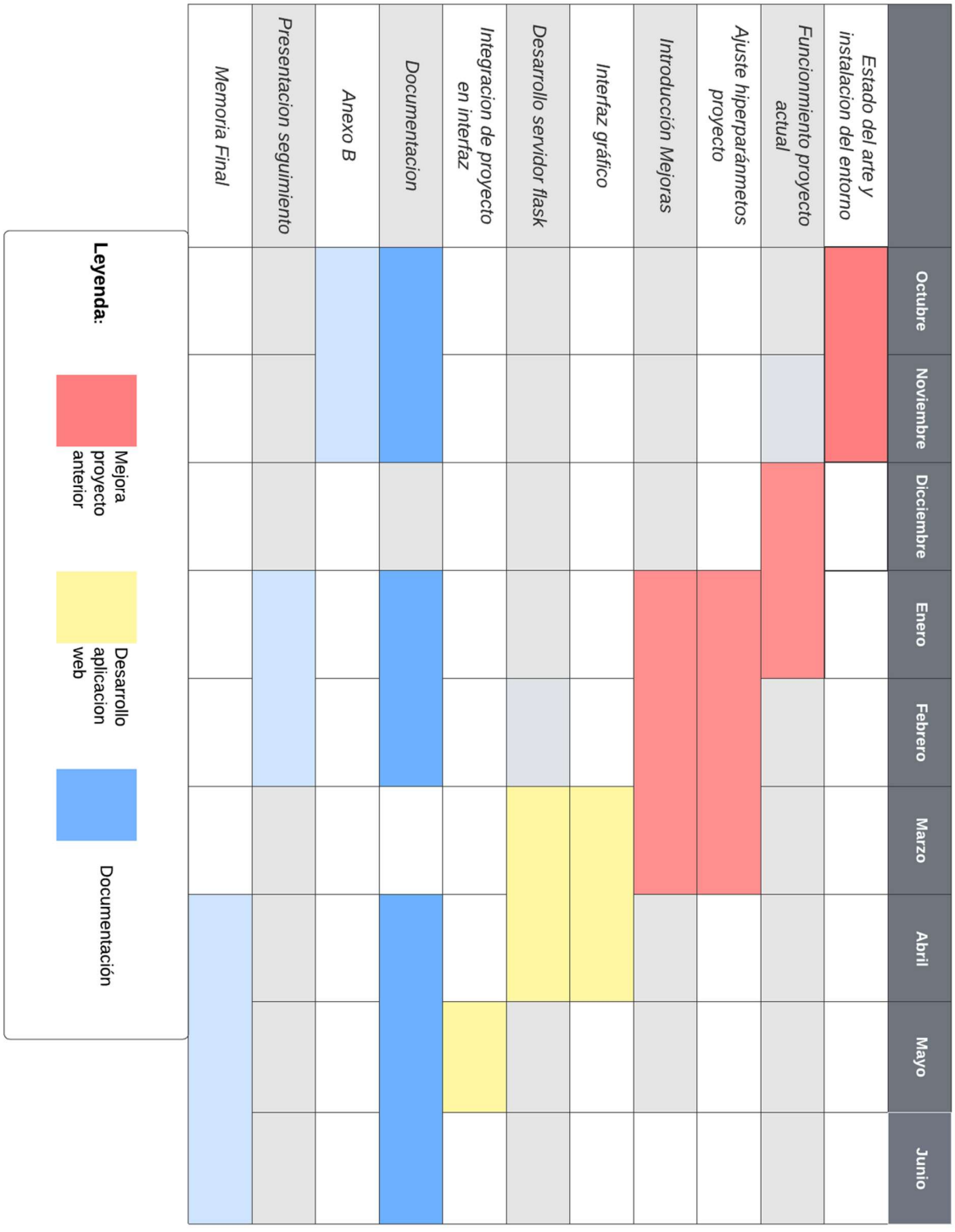

Ilustración 9 - Diagrama de Gantt del proyecto

Capítulo 5. MODELO DESARROLLADO

Este capítulo se centra en el análisis y desarrollo del modelo de C-RNN-GAN utilizado en la generación de muestras musicales. Para ello, se presentan las diferentes etapas y decisiones tomadas durante el proceso de adaptación del modelo, así como los resultados obtenidos y las mejoras implementadas.

El objetivo principal de este capítulo es proporcionar una descripción detallada del modelo actual y su rendimiento, así como explorar y comprobar los distintos modelos obtenidos a partir del estudio y variación de hiperparámetros y funciones implementadas. Se lleva a cabo un análisis de las gráficas de la función de pérdida, la matriz de confusión y la precisión de los modelos con el fin de seleccionar el modelo final óptimo.

El capítulo se estructura en varias secciones que abordan aspectos de los modelos obtenidos. En primer lugar, se realiza un análisis del modelo existente describiendo su funcionamiento y los hiperparámetros utilizados.

A continuación, se comparan los distintos modelos desarrollados, propios de este TFG, mediante los metódos mencionados anteriormente, analizando los resultados en términos de calidad y eficiencia temporal a la hora de ejecutar dichos modelos.

Finalmente se presentará el modelo final seleccionado como el más optimo, justificando esta elección en base a los análisis previos.

5.1. ANÁLISIS DEL MODELO EXISTENTE

En este apartado se describe el modelo desarrollado basado en el enfoque de C-RNN-GAN propuesto por Olof Mogren, explicando su funcionamiento e hiperparámetros utilizados, así como otros parámetros relevantes usados.

El primer paso fue comprender el correcto funcionamiento de las redes C-RNN-GAN,según el esquema de trabajo establecidos en el capítulo 4.2. Para ello se analizó el estudio realizado y se prosiguió con la instalación del entorno de trabajo.

5.1.1. Etapas de funcionamiento de la red C-RNN-GAN

En la figura siguiente se puede observar el funcionamiento de dicho modelo:

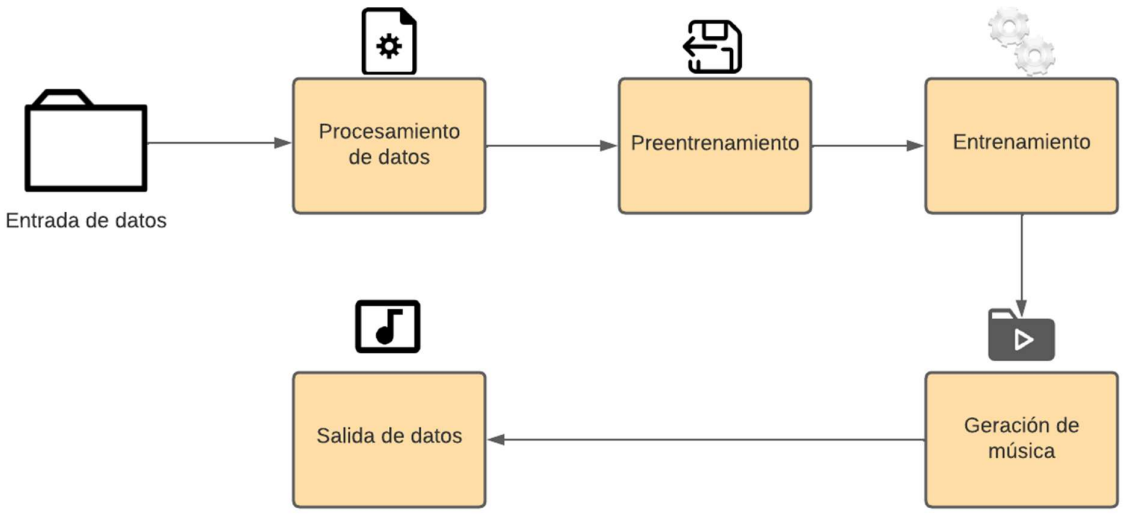

Ilustración 10 - Etapas funcionamiento C-RNN-GAN

1. **Entrada de datos**

La entrada de datos para este modelo son archivos de tipo MIDI (Interfaz Digital de Instrumentos Digitales). Este tipo de archivo es un protocolo de comunicación estándar para la música y producción de audio. Su característica principal es que no contienen sonido en sí mismo. En su lugar, contiene una serie de "eventos" que describen la música, es decir, instrucciones, que describen las notas que se deben tocar. En la siguiente tabla se muestran algunos de los principales eventos de dichos archivos:

Evento	Función
Note_on	Comienzo de una nota en un canal
Note_off	Detención de una nota en un canal
Control_change	Mensaje para ajustar parámetros MIDI
Pitch_bend	Control de la flexión de tono
Polyphonic_aftertouch	Información de presión aplicada tras activación de una nota

Tabla 3 - Eventos midi

Este proyecto utiliza este tipo de archivos debido a su representación estructurada de la música, puesto que se trabaja con eventos discretos como notas en lugar de tratar con formas de audio crudo, que pueden ser variables y ruidosas, además de ser fácilmente manipulables, permitiendo una amplia gama de variaciones a partir de una única pieza de música.

En total se utilizan 2870 archivos de tipo MIDI, todos ellos correspondientes a música clásica con piano.

2. Preprocesamiento

Los archivos MIDI de entrada se procesan utilizando la biblioteca 'python-midi' para convertirlos en un formato entendible por la red neuronal, transformando estos archivos en un formato numérico. Este proceso se lleva a cabo mediante la siguiente secuencia:

1) Lectura del archivo MIDI

Donde se evalúan los eventos contenidos por dicho archivo mediante las funciones aportadas por la librería 'python-midi'.

2) Transformación de eventos MIDI

La red C-RNN-GAN no puede comprender los eventos MIDI como tal, por lo que se transforman en una codificación numérica, usando el método de one-hot encoding. El

funcionamiento de esta codificación se basa en convertir datos simbólicos en identificadores numéricos. Posteriormente, cada evento se representa como un vector de números donde todos son ceros menos el correspondiente al evento, con valor 1. De esta manera, si hay 25 teclas en un teclado, la nota C podría ser representada mediante un vector de 25 números, donde todos los valores son 0, a excepción del primero.

3) Secuencia de entrada para la RNN

Acorde a la explicación del capítulo 3.4 , las RNNs toman como entrada una secuencia de números. Por lo tanto, una vez se ha realizado la transformación de eventos MIDI, se lleva a cabo la agrupación de dichos vectores en secuencias, cada una correspondiente a una porción de música, que se alimenta a la RNN durante el proceso de entrenamiento.

3. Preentrenamiento

Como se detalla en capítulo 3.2, el preentrenamiento es una técnica utilizada en el aprendizaje profundo donde un modelo se entrena primero en una tarea auxiliar antes de ser entrenado en la tarea principal, para garantizar la obtención de mejores resultados.

Para la C-RNN-GAN, el preentrenamiento implica entrenar la red RNN como generador para producir muestras que se parezcan a las reales con la función de pérdida de reconstrucción.

Una vez completado esta etapa, el generador ya tiene una idea de cómo producir secuencias de música realistas, con una mejor posición de partida para empezar el proceso de entrenamiento.

4. Entrenamiento del modelo

El modelo actual se entrena con el conjunto de datos de archivos MIDI. Durante este proceso, el generador intenta generar música realista, en comparación con el conjunto de datos de entrenamiento. Por su parte el discriminador intenta distinguir entre muestras generadas de manera artificial y reales. Ambos componentes se entrenan juntos mediante un proceso adversario, como el detallado en el capítulo 3.3.

5. **Generación de música**

Una vez el modelo ha sido entrenado, se generan música artificial que se parece a la aprendida durante el entrenamiento, alimentando al generador con un vector de ruido, siendo, en este caso, un conjunto de números aleatorios. La red genera una secuencia de eventos MIDI, que es convertida en un archivo MIDI mediante la biblioteca 'pyhton-midi'.

6. **Salida de datos**

Los archivos MIDI generados por la red, son guardados como salida del proyecto, los cuales pueden ser reproducidos con cualquier software que soporte este tipo de archivos.

5.1.2. Instalación de entorno de trabajo

El primer paso fue clonar el repositorio de github del modelo anterior para comprobar su funcionamiento actual (referencia github). Este modelo se desarrolla utilizando Python 3 en su totalidad y se utilizan las siguientes librerías:

- sys
- os
- midi
- math
- string
- time
- tensorflow
- python-midi

Todas ellas juegan un papel principal a la hora de ejecutar el modelo, destacando la importancia de tensorflow y python-midi.

Tensorflow es una biblioteca de código abierto para aprendizaje automático y profundo que proporciona un conjunto de módulos que permiten diseñar modelos de aprendizaje

automático. En 2018 Tensorflow lanzó su versión 2.X, aunque este proyecto funciona con la versión Tensorflow 1.15.2, por lo que fue necesario instalar una versión anterior.

Python-midi es una biblioteca usada para trabajar con archivos MIDI permitiendo varias operaciones como leer y escribir, manipulación de eventos MIDI y manejar su compatibiliadad. Esta biblioteca es de gran importancia para manejar los archivos de entrada y salida del modelo.

Para trabajar con el modelo se ha utilizado el entorno Visual Studio Code, por la facilidad de trabajar con proyectos organizados en carpetas y su compatibilidad con lenguajes de programación.

5.1.3. Descripción de hiperparámetros

La definición de los hiperparametros de la red C-RNN-GAN es un aspecto principal a la hora de obtener resultados válidos y coherentes. Para ello se realizó el estudio de dichos hiperparámetros y su función.

1. **Tasa de aprendizaje (learning_rate)**

Este hiperparametro determina cuanto deben actualizarse los pesos del modelo en cada paso de optimización. Un valor elevado de este parámetro puede conseguir que el modelo se actualice rápidamente, corriendo el riesgo de saltar mínimos locales de valores óptimos. Un valor bajo de este parámetro puede hacer que le modelo converja lentamente hacia la solución óptima.

2. **Valor máximo de la norma del gradiente (max_grad_norm)**

Establece el valor máximo permitido para dicha norma, lo que es esencial para la explosión del gradiente durante la etapa de entrenamiento, evitando así que los valores se vuelvan inestables y dificulten la convergencia del modelo.

3. **Numero de capas (num_layers)**

Es el encargado de definir la profundidad del modelo y su capacidad para capturar representaciones complejas. Elevar el número de capas, puede aumentar la capacidad de aprendizaje de dicho modelo, al mismo tiempo que aumenta el riesgo del sobreajuste si no es controlado de manera correcta.

4. **Longitud de las muestras generadas (song_length)**

Como su nombre indica, este hiperparametro establece la longitud de las secuencias de música generadas por el modelo. Esto determinan cuantos pasos de tiempo se consideran en cada etapa de generación, afectando a la estructura y duración de las muestras generadas.

5. **Tamaño de la capa oculta (hidden_size)**

Este valor representa la cantidad de estados ocultos presentes en las celdas LSTM. La dimensión de esta capa define la cantidad de unidades de memoria o estados ocultos presentes en estas celdas. Cada estado oculto se encarga de almacenar la información pasada, ayudando a la modelación de las dependencias a largo plazo. Un valor demasiado pequeño puede limitar la capacidad del modelo para capturar la complejidad de las secuencias musicales, dando lugar a valores de menor veracidad. Por otro lado, un valor demasiado grande de este parámetro puede llevar a la aparición del sobreajuste y dificultar el proceso de entrenamiento.

6. **Número de épocas antes de disminución de la tasa de aprendizaje (epochs_before_decay)**

Este hiperparametro indica el número de épocas antes de que se aplique un decaimiento a la tasa de aprendizaje, permitiendo controlar la velocidad de cambio de aprendizaje durante el entrenamiento. Esto puede ser útil a la hora de mejorar la convergencia y estabilidad de este.

7. **Número máximo de épocas (max_epoch)**

Su valor define el límite de iteraciones en las que el modelo actualiza los pesos con los datos de entrenamiento, evitando llegar al sobre ajuste, producido por el sobre entrenamiento.

8. **Probabilidad de mantener una neurona activa (keep_prop)**

El valor de este hiperparametro representa la probabilidad de mantener una neurona activa durante el entrenamiento. Esto se logra mediante la técnica de dropout, que ayuda a prevenir el sobreajuste mediante a la desactivación aleatoriamente de un porcentaje del número de neuronas usadas en cada paso del entrenamiento. De esta manera se logra la generalización del modelo.

9. **Factor de decaimiento (lr_decay)**

Dicho factor es aplicado a la tasa de aprendizaje después de cada conjunto de épocas especificado en el hiperparametro "epochs_before_decay". Mediante esta técnica se consigue controlar la reducción gradual de la tasa de aprendizaje a medida que avanza el entrenamiento, con la necesidad de requerir más recursos computacionales.

10. **Tamaño del lote de muestras (batch_size)**

Este valor indica cuantos ejemplos se procesan simultáneamente antes de actualizar los pesos del modelo. Aumentar el valor del lote puede acelerar el modelo, requiriendo más recursos computacionales.

11. **Escala inicial para inicial los pesos del modelo (init_scale)**

Es el encargado de establecer la escala inicial de los pesos del modelo. Este valor influye en el valor inicial de los pesos del modelo afectando a la estabilidad y convergencia del modelo durante el entrenamiento.

Los valores de los hiperparametros utilizados en el modelo original se muestran en la siguiente tabla:

Hiperparametro	Valor
learning_rate	0.1
max_grad_norm	5.0
num_layers	2
songlength	100
hidden_size	100
epochs_before_decay	60
max_epoch	500
keep_prob	0.5
lr_decay	1.0
batch_size	20
init_scale	0.05
learning_rate	0.1

Tabla 4 - Hiperparametros inciales

5.1.4. Otros parámetros usados por el modelo

El modelo utiliza otros parámetros que influyen en su comportamiento y rendimiento de este. Estos incluyen:

- datadir y traindir:

Rutas de los directorios utilizados para almacenar los datos de entrenamiento y los resultados obtenidos.

- epochs_per_checkpoint:

Este parámetro especifica el número de épocas entre cada punto de control donde se guarda el modelo. Estos puntos de control son los momentos en el proceso de entrenamiento donde se almacena el estado actual del modelo para su posterior evaluación o reanudación.

- log_device_placement:

Este indicador controla si se muestra información de colación de dispositivos durante la ejecución del modelo, mostrando información sobre que dispositivo (CPU o GPU) se utiliza para cada operación de la red neuronal.

- call_after y exit_after:

Permiten configurar el tiempo en minutos después del cual se ejecuta o finaliza la ejecución del modelo.

- select_validation_percentage y select_test_percentage:

Estos parámetros son los encargados de identificar el porcentaje de datos de entrada que se utilizan para el proceso de validación y de prueba del modelo, separándolo de los de entrenamiento.

- sample:

Mediante este parámetro se controla si el modelo genera muestras de música durante el proceso de entrenamiento.

- works_per_composer:

Permite controlar la distribución de las obras de distintos compositores en el conjunto de entrenamiento.

- disable_feed_previous:

Mediante este indicador, se controla la desactivación de la retroalimentación anterior en la generación de muestras.

- d_lr_factor:

Este factor de ajuste se aplica a la tasa de aprendizaje del discriminador de la red GAN, controlando la velocidad de aprendizaje del discriminador en relación con el generador.

- meta_layer_size:

Este parámetro especifica el tamaño de la capa meta utilizada en el modelo, siendo esta una capa adicional que se utiliza para la captura de información de alto nivel, ayudando en la generación de muestras coherentes.

- hidden_size_g y hidden_size_d:

Ambos parámetros estan relacionados con el hiperparametro "hidden_size" y permiten establecer distintos tamaños de las capas ocultas en el generador y discriminador.

- biscale_slow_layer_ticks:

Especifica el número de ticks o pasos de tiempo se asignan para cada capa lenta de la arquitectura biscale. Esta arquitectura es una variante del modelo C-RNN-GAN que utiliza múltiples escalas de tiempo para capturar distintas características de las muestras generadas, como patrones a largo plazo. La variación de este parámetro dependerá del tipo de música que se desea generar y sus características musicales específicas.

- multiscale:

Este indicador controla si se utiliza la arquitectura multiscale en el modelo. Esta arquitectura se basa en una técnica que emplea múltiples escalas de tiempo para la captura de características tanto locales como globales durante la generación de muestras.

- pretraining_epochs:

Indica el número de etapas de preentrenamiento utilizadas en el proceso de entrenamiento del modelo.

- pretraining_d:

Este indicador establece si se realiza etapas de preentrenamiento para el discriminador en específico de la red GAN.

- initialize_d:

Mediante este indicador se controla si se inicializan los pesos del discriminador antes del entrenamiento. Esta inicialización puede ayudar a acelerar el proceso de entrenamiento y la estabilidad del modelo.

- ignore_saved_args

Controla si se ignoran los argumentos previamente guardados del modelo, proporcionando flexibilidad mediante la omisión de ciertos parámetros guardados y la utilización de nuevos valores.

- pace_events:

Este parámetro controla el ritmo establecido para los eventos generados durante la generación de muestras, estableciendo una cadencia específica para lograr un flujo y estructura adecuados.

- minibatch_d

Este parámetro es utilizado para controlar el uso del entrenamiento del discriminador por lotes, lo que procesa múltiples ejemplos al mismo tiempo, lo que puede mejorar la eficiencia del modelo.

- unidirectional_d:

Mediante este indicador se controla si el discriminador es unidireccional, considerando solo la información pasada en la secuencia de música. Si por el contrario es bidireccional, se toma información de muestras generadas.

- profiling:

Este indicador controla si se realiza el perfilado del modelo durante el entrenamiento. Dicha técnica se utiliza para medir y analizar el rendimiento del modelo.

- float16

Este indicador controla si se utilizan datos de tipo float16 en lugar de float32 durante el proceso de ejecución del modelo. Los datos de tipo float16 pueden acelerar el entrenamiento del modelo, al ocupar menos memoria, aunque también derivará en una menor precisión numérica.

- adam:

Este parámetro controla si se utiliza el optimizador Adam en lugar del SGD (descenso del gradiente estocástico) en el entrenamiento del modelo. La principal ventaja del optimizador Adam es su capacidad de adaptar automáticamente la tasa de aprendizaje durante el entrenamiento. Para ello utiliza una estimación adaptativa del primer momento (media) y el segundo momento (varianza) de los gradientes para ajustar dicha tasa de aprendizaje.

El sistema SGD utiliza una tasa fija de aprendizaje durante todo el proceso de entrenamiento, por lo que Adam puede proporcionar una mejora en términos de velocidad de convergencia y precisión del modelo, aunque no tiene por qué derivar en una solución óptima.

- feature_matching:

Este indicador controla si la técnica de “Feature matching “es utilizada en la arquitectura GAN. Esta técnica busca minimizar la diferencia entre las características extraídas de las muestras reales y las generadas por el generador de la GAN.

- disable_l2_regularizer

Este parámetro se encarga del control de la desactivación de la regulación L2 del modelo, la cual es responsable de introducir un término de penalización en la función de pérdida para evitar el sobreajuste de este.

- reg_scale:

Este parámetro especifica la escala utilizada para la regularización en caso de que no se desactive por el parámetro "disable_l2_regularizer", controlando la fuerza de dicha regularización y su impacto en el modelo.

- synthetic_chords:

Controla el uso de acordes sintéticos en lugar de datos reales en el entrenamiento del modelo. Estos acordes pueden ser generados de forma aleatoria o mediante el uso de algoritmos y pueden ser de gran utilidad para ampliar la variedad y diversidad de las muestras generadas.

- tones_per_cell:

Este parámetro determina el número de tonos que se permiten en cada celda durante el proceso de generación, afectando a la complejidad y densidad de las composiciones generadas.

- composer:

Este parámetro permite enfocar el entrenamiento a un compositor en específico adaptando las muestras generadas a su estilo y características musicales.

- generate_meta:

Este indicador controla la generación de información meta durante el proceso de entrenamiento. Dicha información puede contener datos adicionales sobre las muestras generadas, como el tempo, la métrica o las marcas de tiempo.

- random_input_scale:

Este parámetro especifica la escala utilizada para la entrada aleatoria en el proceso de generación de música, la cual se utiliza para introducir variabilidad y creatividad en las muestras generadas.

- end_classification:

Este indicador controla si se realiza una clasificación de al final del modelo de las muestras generadas. Esto pude ser útil al ahora de clasificar muestras por diferentes clases o muestras musicales.

A continuación, se muestran los valores de los anteriores parámetros utilizados en el modelo original:

Parámetro	**Valor**
datadir	None
traindir	None
epochs_per_checkpoint	2
log_device_placement	False
call_after	None
exit_after	1440
select_validation_percentage	None
select_test_percentage	None
sample	False
works_per_composer	None
disable_feed_previous	False
d_lr_factor	0.5
meta_layer_size	200

hidden_size_g	100
hidden_size_d	100
biscale_slow_layer_ticks	8
multiscale	False
pretraining_epochs	6
pretraining_d	False
initialize_d	False
ignore_saved_args	False
pace_events	False
minibatch_d	False
unidirectional_d	False
profiling	False
float16	False
adam	False
feature_matching	False
disable_l2_regularizer	False
reg_scale	1.0
synthetic_chords	False
tones_per_cell	1
composer	None
generate_meta	False

Tabla 5- Tabla de parámetros iniciales

5.2. COMPARACIÓN DE MODELOS E INTRODUCCIÓN DE NUEVAS FUNCIONES

En este apartado, se lleva a cabo la descripción de varios modelos desarrollados de la red C-RNN-GAN. Se han explorado numerosas variaciones de hiperparámetros y parámetros además de la introducción de nuevas funciones con el objetivo de mejorar el rendimiento y calidad de las muestras generadas. A continuación, se presentan los modelos más significativos que han surgido de este proceso de experimentación.

Con el fin de facilitar el seguimiento y comprensión de los modelos desarrollados durante este trabajo, se ha adoptado un sistema de denominación se basa en el número de etapas de preentrenamiento llevadas a cabo en cada modelo. De esta manera, el modelo original, se denomina como "Modelo 6", ya que ha sido preentrenado durante 6 etapas específicas.

5.2.1. Modelo 6 (modelo original)

El primer paso consistió en examinar con detalle el modelo original con los valores de los hiperparámetros y parámetros descritos en la sección anterior 5.1.

Este modelo ha sido preentrenado durante 6 etapas específicamente para adaptarse a los datos de entrenamiento, aprendiendo patrones y estructuras musicales presentes en el conjunto de datos, aunque el número de etapas puede aumentar, como se verá en los siguientes apartados, para lograr resultados más optimizados sin llegar al sobreajuste del modelo.

El objetivo del análisis de este modelo es establecer una base comparativa que sirva de referencia para el resto de los modelos explorados durante este trabajo. Este modelo proporciona una referencia valiosa para comprender las fortalezas y debilidades de la C-RNN-GAN y establece un punto de partida sólidos sobre el que enfocar el trabajo. Al evaluar y comparar los resultados obtenidos, se pueden determinar si las variaciones introducidas en los hiperparámetros y parámetros además de la introducción de las nuevas funciones han

logrado mejoras significativas en el rendimiento y calidad de las muestras generadas por el modelo.

5.2.2. Modelo 8

El modelo 8 es uno de los modelos que refleja mejoras significativas en la generación de muestras. En este modelo, se ha llevado a cabo un incremento en el número de etapas de preentrenamiento a 8, con el objetivo de obtener una representación más sólida de los datos de música utilizados durante este trabajo.

Además, se han realizado diversas variaciones en algunos de los hiperparámetros y parámetros, mejorando la calidad de las muestras generadas, mostrados en la siguiente tabla:

Hiperparámetros / parámetros	**Nuevo valor**
Learning_rate	0.3
Hidden_size, / Hidden_size_g / Hidden_size_d	120
Tones_per_cell	3
Batch_size	30
Keep_prob	0.6
Otros hiperparámetros y parámetros	Valores por defecto

Tabla 6- Tabla Modelo 8

1. Learning_rate

Se ha establecido un valor de 0.3 para este parámetro, aumentando en 0.2 respecto al original con la intención de encontrar una convergencia rápida y estable de manera efectiva durante el proceso de entrenamiento del modelo.

2. Hidden_size, / Hidden_size_g / Hidden_size_d

Se ha incrementado el valor de estos parámetros hasta 120. Mediante este ajuste, se ha logrado una representación más exacta de los datos y mejoras en la captura de las características más complejas en las muestras generadas.

3. Tones_per_cell

El nuevo valor de este parámetro, 3, ha conseguido asegurar una representación adecuada de los tonos musicales en cada celda, manteniendo una coherencia melódica en la generación de muestras

4. Batch_size

El incremento de este parámetro a 30 ha facilitado una mejor estimación del gradiente y permite una actualización más precisa de los pesos del modelo y una mejora en los tiempos de entrenamiento.

5. Kee_prob

Se ha aumentado la capacidad de la red mediante la técnica de dropout, lo que ayuda a prevenir el sobreajuste y mejora la generalización del modelo.

El resto de hiperparámetros y parámetros han mantenido sus valores originales para mantener la coherencia con la implementación del Modelo 6 y permitir una comparación adecuada.

5.2.3. Modelo 10

Este modelo es una continuación del análisis y mejora del modelo realizado en el Modelo 8. En este modelo, se ha llevado a cabo un incremento adicional en el número de etapas de preentrenamiento a 10, con el objetivo de mejorar aún más la capacidad de representación y generación del modelo. Adicionalmente, se han llevado a cabo ajustes adicionales en los hiperparámetros e introducción de nuevas funciones, para mejorar el rendimiento, que serán descritas más adelante.

Los nuevos valores de los hiperparámetros y parámetros se muestran en la siguiente tabla:

Hiperparámetros / parámetros	**Nuevo Valor**
Learning_rate	0.2
Hidden_size / Hidden_size_g / Hidden_size_d	150
Tones_per_cell	4
Batch_size	30
Keep_prob	0.7
Init_scale	0.03
initialize_d	True
Adam	True
Otros hiperparámetros y parámetros	Valores por defecto

Tabla 7 – Tabla Modelo 10

1. Learning_rate

Se ha establecido el nuevo valor en 0.2 para ajustar la tasa de aprendizaje del modelo, con el objetivo de equilibrar la rapidez de convergencia y la exploración efectiva del espacio de soluciones.

2. Hidden_size / Hidden_size_g / Hidden_size_d

Se aumenta el valor de estos parámetros a 150, incrementando la capacidad de representación de la red. De esta manera, el modelo tiene una mayor capacidad para capturar características más complejas.

3. Tones_per_cell

Se ha aumentado el valor a 4 para permitir una representación más detallada de los tonos musicales en cada celda y así la capacidad de capturar mayor complejidad por parte del modelo.

4. Batch_size

Se ha mantenido en 30 respecto al Modelo 8, para mantener el equilibrio entre tiempo de entrenamiento y exploración del espacio de entrenamiento.

5. Keep_prob

Este parámetro se ha incrementado a el valor 0.7, comprobando el aumento de la capacidad de la red sin llegar al sobre ajuste.

6. Init_scale

Se ha establecido en 0.03 para logra una inicialización más precisa de los pesos de la red, contribuyendo a un entrenamiento más estable.

7. Initialize_d & Adam

Se han activado estos dos parámetros para ajustar de mejor manera el modelo, consiguiendo una mejora en los tiempos de ejecución de este.

El resto de los valores han sido mantenidos para lograr una coherancia con la comparación del resto de modelos. Además, se han incluido varias funciones adicionales, mejorando el sistema de procesado y tiempos de ejecución del modelo.

5.2.4. ***Modelo 12***

En el proceso de búsqueda de los valores óptimos de los hiperparámetros y parámetros del modelo, se han explorado diversas configuraciones que pueden resultar en modelos en los que se produce sobreajuste o pérdida de estabilidad en la generación de muestras. Uno de estos modelos es el Modelo 12, que se utilizará para analizar los efectos del sobreajuste y comparar con el resto de los modelos.

Para este modelo se han usado 12 etapas de preentrenamiento y las siguientes variaciones:

Hiperparámetros / parámetros	**Nuevo valor**
Learning rate	0.01
Keep prob	0.3
Batch size	10
Init_scale	0.001
Tone per cell	2
Adam	true
Initialize_d	true

Tabla 8 - Tabla Modelo 12

1. Learning_rate

Se utiliza una tasa de aprendizaje más baja que los modelos anteriores, lo que puede ralentizar la convergencia y reducir la genaralización.

2. Keep_prob

Se establece una mayor propabilidad de mantener neuronas activas durante el entrenamiento, aumentando el riesgo de sobre ajuste.

3. Bacth size

Se reduce el tamaño de cada lote a 10, lo que puede llevar al aumento de la posibilidad del sobreajuste de la red durante el proceso de entrenamiento, al mismo tiempo que aumenta el tiempo de procesado de cada etapa.

4. Init_scale

Asignando un valor menor a este parámetro, se reduce el tamaño de los pesos iniciales del modelo por lo que pueden resultar más difícil de entrenar, contribuyendo a un rendimiento subóptimo.

5. Tone per cell

Se reduce el número de tonos por celda a 2, lo que puede limitar la capacidad del modelo para capturar datos de mayor complejidad.

6. Initialize_d and Adam

Este modelo se desarrolló a partir del modelo 10, por lo que se han mantenido la activación de estos dos parámetros, comprobando que no son óptimos de utilizar si el resto de hiperparámetros y parámetros son estables.

Este modelo también incluye las funciones añadidas en el modelo 10.

5.2.5. Funciones añadidas

Durante el desarrollo de los distintos modelos se ha realizado un análisis de las diferentes etapas del proceso de generación de muestras, incluyendo la manipulación y preparación de los datos MIDI.

En esta sección se presentan y analizan cuatro funciones introducidas con el objetivo de mejorar los tiempos de ejecución del modelo y reducir la pérdida (loss).

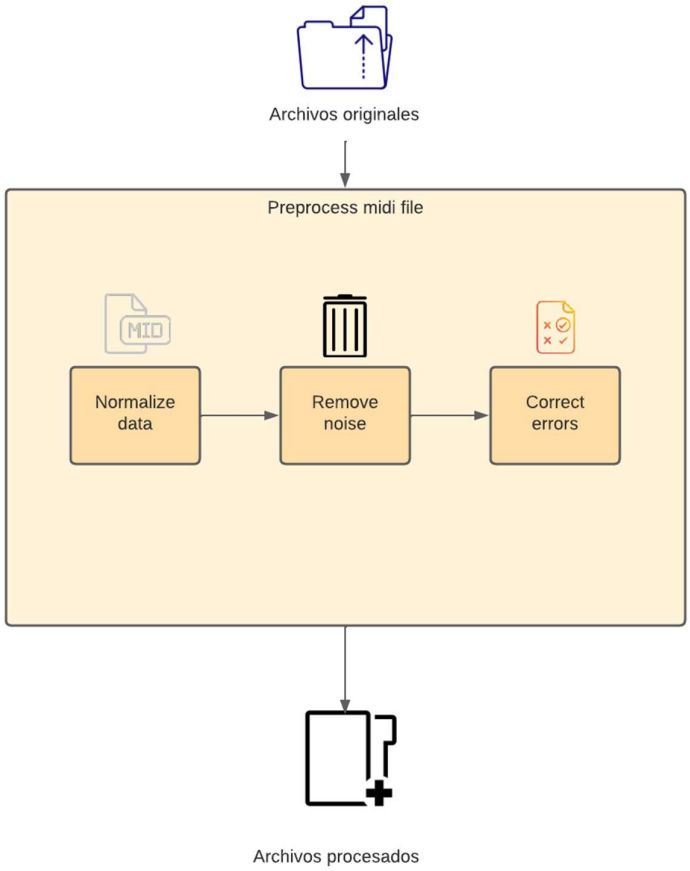

Ilustración 11 - Funciones añadidas

1. Normalize_data

Esta función se ha implementado para abordar los valores de velocidad, presentes en los eventos MIDI. Mediante esta función se normalizan estos valores entre 0 y 1, permitiendo que el modelo tenga una mejor comprensión de notas asegurando una comprensión más coherente. Con la normalización se evita la predominancia de notas demasiado fuertes o débiles, lo que puede afectar negativamente el equilibrio y calidad de las muestras.

2. Remove_noise

La labor de esta función es la de lidiar con las notas de baja intensidad o ruido en los datos MIDI. Estas notas pueden generar datos no deseados durante la generación de muestras afectando a la calidad y coherencia de estas. Al eliminar las notas por debajo de una velocidad establecido, se reduce el impacto de este tipo de notas mejorando la coherencia de las muestras generadas.

3. Correct_errors

El cometido de esta función es corregir cualquier error en las alturas de las notas (pitch) que se encuentren fuera del rango válido. En los archivos MIDI es común encontrar datos que exceden o se encuentran por debajo de los límites aceptados, afectando al rendimiento del modelo y la calidad de las muestras generadas. Esta función mejora la capacidad del modelo para obtener muestras más coherentes.

4. Preprocess_midi_file

Esta función se ha creado para compactar todas las transformaciones necesarias de los archivos midi, agrupando las anteriores, facilitando y agilizando el tratamiento de los datos antes de su entrada al modelo.

Mediante estas funciones se garantiza coherencia y calidad del conjunto de datos, lo que se traduce en muestras más precisas.

La implementación de dichas funciones se muestra a continuación:

```
def normalize_data(midi_data):

    for track in midi_data:

        for event in track:

            if isinstance(event, midi.NoteOnEvent) or isinstance(event,
midi.NoteOffEvent):

                event.velocity /= 127.0
```

```
def remove_noise(midi_data, velocity_threshold=0.1):
    for track in midi_data:
        notes_to_remove = []
        for i, event in enumerate(track):
            if isinstance(event, midi.NoteOnEvent):
                if event.velocity < velocity_threshold:
                    notes_to_remove.append(i)
        for i in reversed(notes_to_remove):
            del track[i]


def correct_errors(midi_data):
    note_range = range(21, 109)
    for track in midi_data:
        for event in track:
            if isinstance(event, midi.NoteOnEvent) or isinstance(event,
midi.NoteOffEvent):
                if event.pitch not in note_range:
                    if event.pitch < note_range.start:
                        event.pitch = note_range.start
                    elif event.pitch >= note_range.stop:
                        event.pitch = note_range.stop - 1


def preprocess_midi_file(filename):
    midi_data = midi.read_midifile(filename)
    normalize_data(midi_data)
    remove_noise(midi_data)
    correct_errors(midi_data)
```

```
return midi_data
```

5.3. ANÁLISIS DE GRÁFICAS DE PÉRDIDA Y TIEMPOS DE EJECUCIÓN

En este apartado se analizarán las gráficas de la función de pérdida de los modelos descritos en el anterior apartado, con el fin de evaluar su desempeño y comparar sus resultados. Estas gráficas son una representación visual del cambio en la función de pérdida a lo largo de la ejecución del modelo.

Se evalúan tres tipos de funciones de error correspondientes a entrenamiento (training), validación (valid) y prueba (testing). Se examina la pérdida de tanto el generador (G) como del discriminador (D). Estos conceptos se han descrito en el capítulo 3.6.

El análisis de las gráficas se realizará en varios niveles para obtener una mejor compresión del rendimiento del modelo.

5.3.1. Pérdida Entrenamiento y validación

En este apartado se muestra una gráfica que combina las curvas de pérdida tanto para el conjunto de entrenamiento y validación a lo largo de las 25 primeras épocas de entrenamiento, comparando la relación directa entre ambas tareas.

Si las curvas de entrenamiento y validación siguen una tendencia similar y disminuyen a lo largo del entrenamiento, esto indica que el modelo está aprendiendo de manera efectiva y si llegar a producirse sobre ajuste.

Es importante mencionar que durante el proceso de entrenamiento los valores de la función de pérdida del generador durante el entrenamiento, toma valores muy elevados, mientras el resto de las curvas son cercanas a cero.

Durante las etapas de preentrenamiento los pesos del generador se inicializan de manera aleatoria, puesto que la red no ha aprendido a generar música de calidad. Como resultado, el generador produce salidas que difieren significativamente de los datos reales, siendo descartadas fácilmente por el discriminador.

A continuación, se muestran dichas gráficas para los modelos 6, 8, 10 y 12:

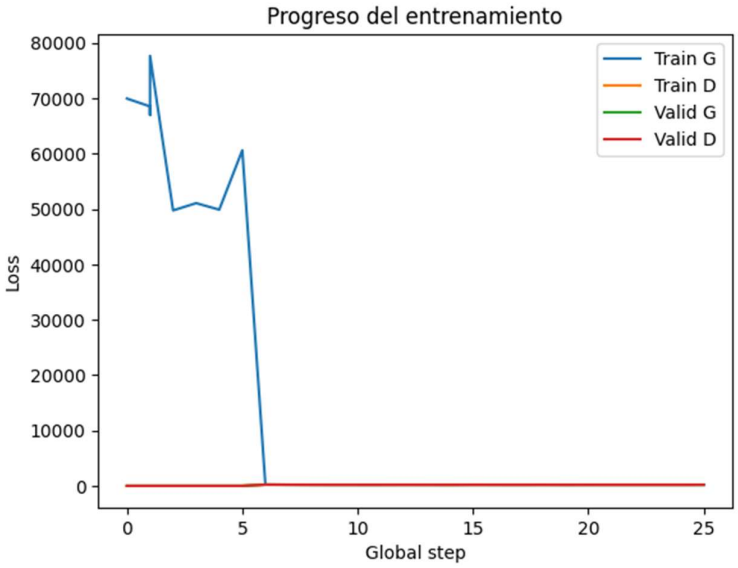

Ilustración 12 – Curvas loss entrenamiento Modelo 6

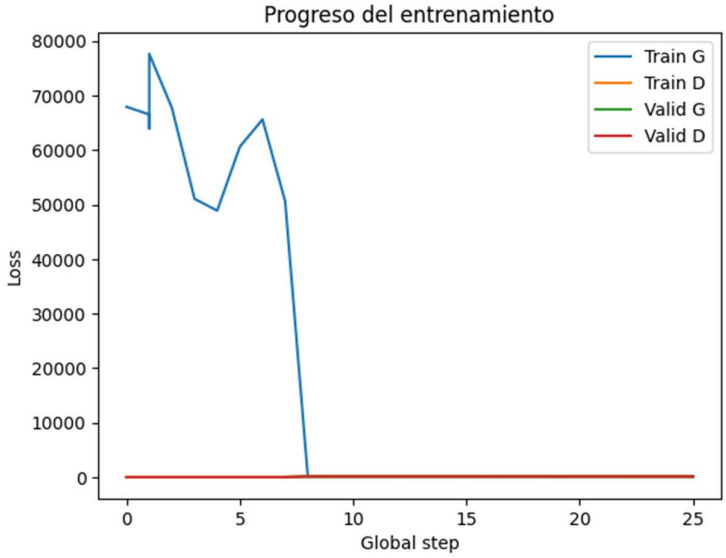

Ilustración 13 - Curvas loss entrenamiento Modelo 8

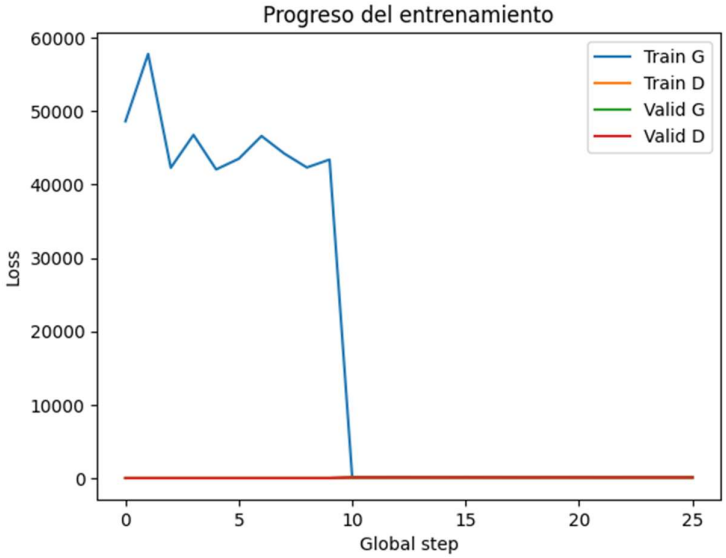

Ilustración 14 - Curvas loss entrenamiento Modelo 10

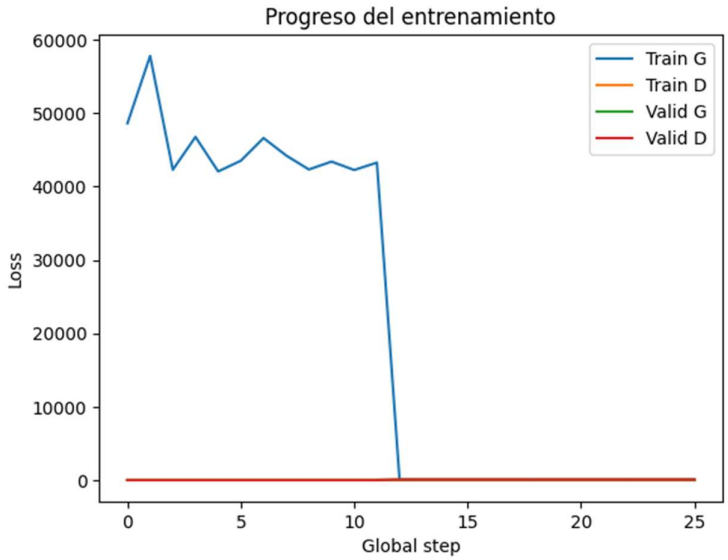

Ilustración 15 - Curvas loss entrenamiento Modelo 10

Al ser los valores iniciales de la función de pérdida del generador durante el preentrenamiento, es necesario ampliar dichas figuras para obtener unos resultados más claros, debido al solapamiento de las gráficas. Además, se separan cada una de las curvas para observarlas de manera asilada:

1. Modelo 6

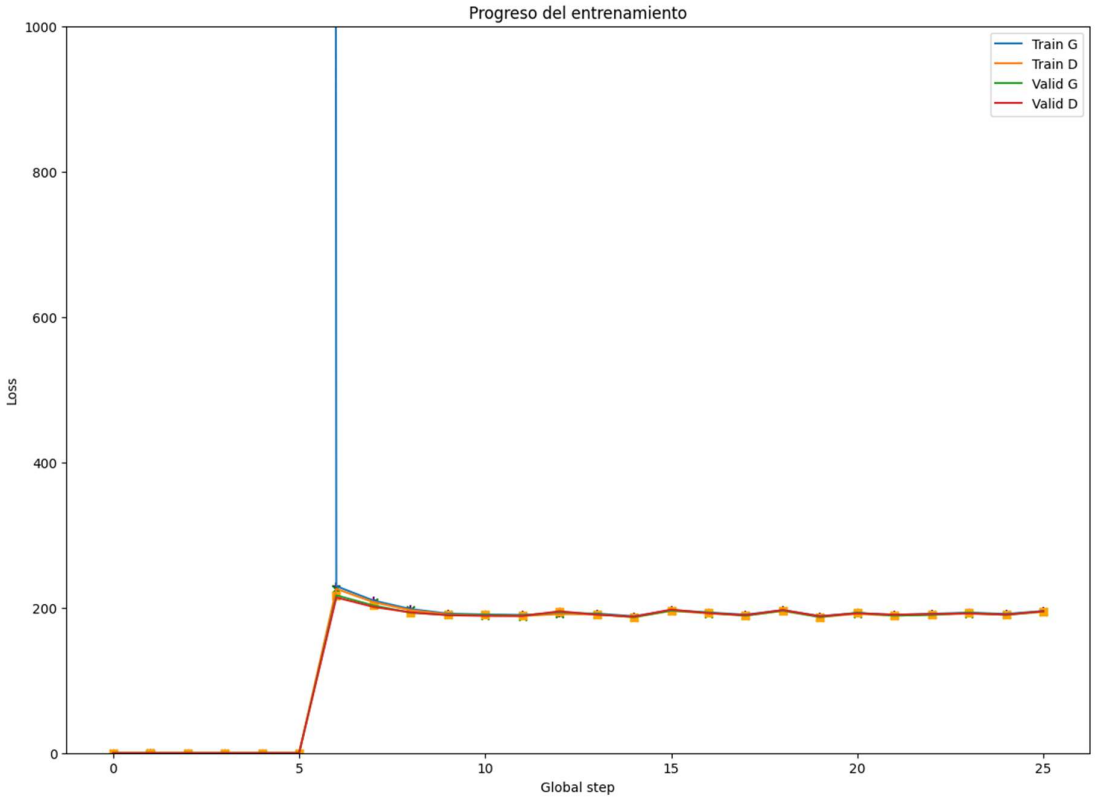

Ilustración 16 – Ampliación curvas loss entrenamiento Modelo 6

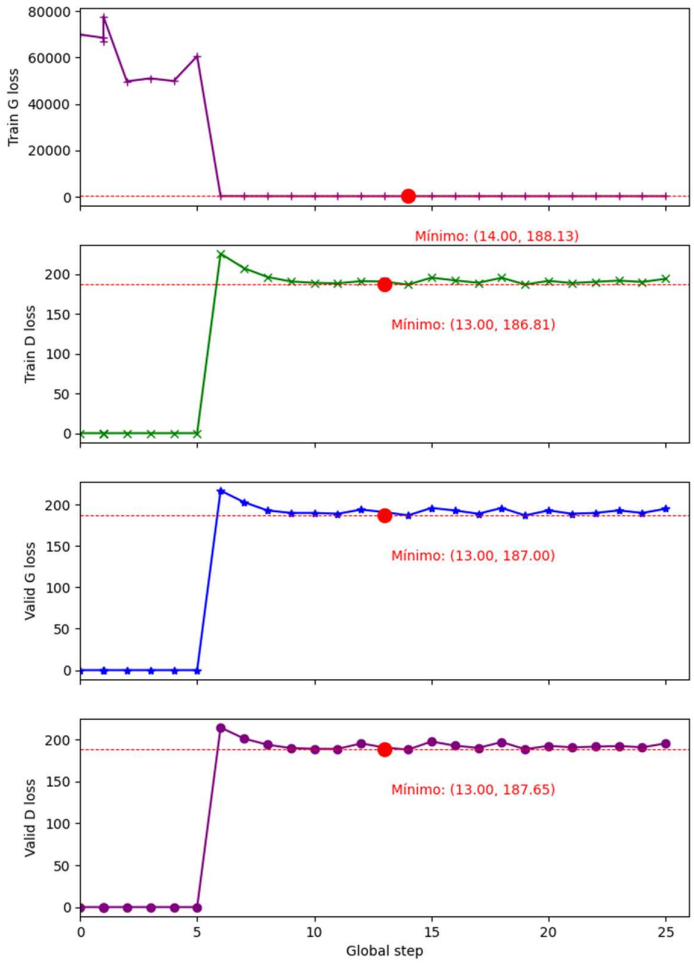

Ilustración 17 – Separación curvas loss entrenamiento Modelo 6

Observando las gráficas, podemos observar que durante el entrenamiento y validación del modelo 6, se observa que las pérdidas tanto para el generador como para el discriminador se encuentran alrededor de 200.

Los mínimos de dichas gráficas se muestran en rojo y se encuentran entrono a 187, llegando a estos valores en la etapa número 13 del entrenamiento, a excepción de la curva de la pérdida del generador en el entrenamiento. En este caso se llega al mínimo en la etapa 14, obteniendo valores muy parecidos en la etapa 13.

2. Modelo 8

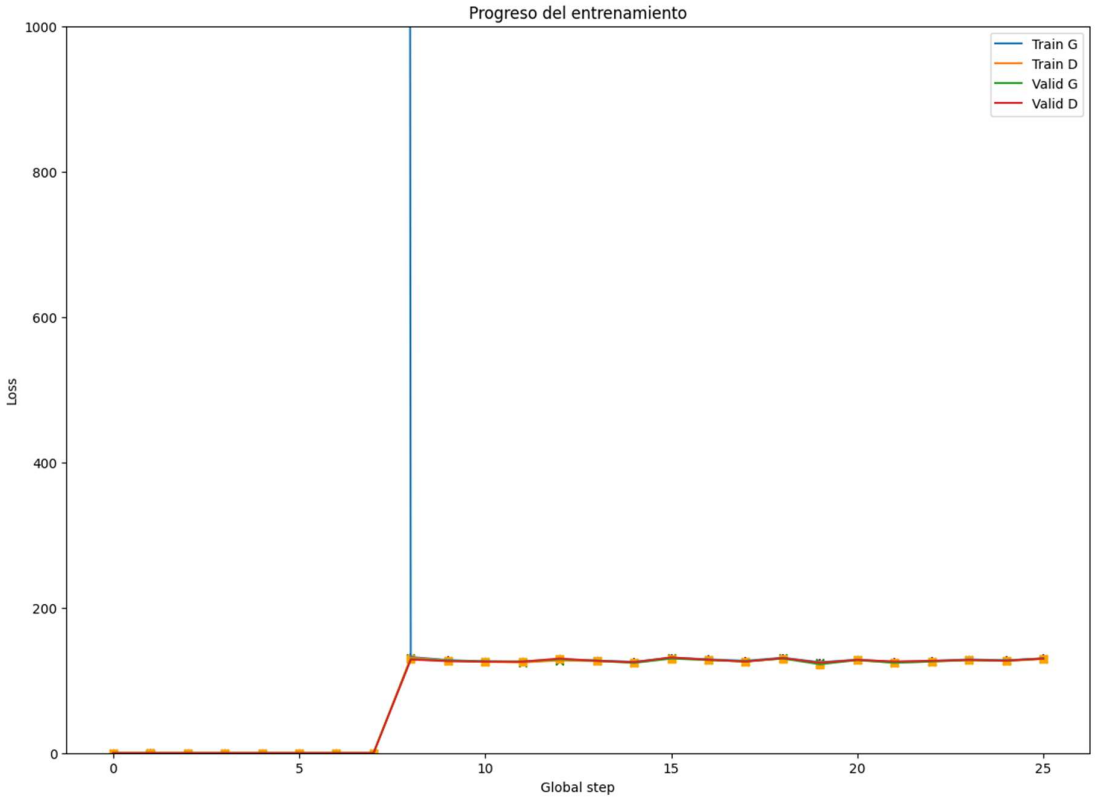

Ilustración 18 - Ampliación curvas loss entrenamiento Modelo 8

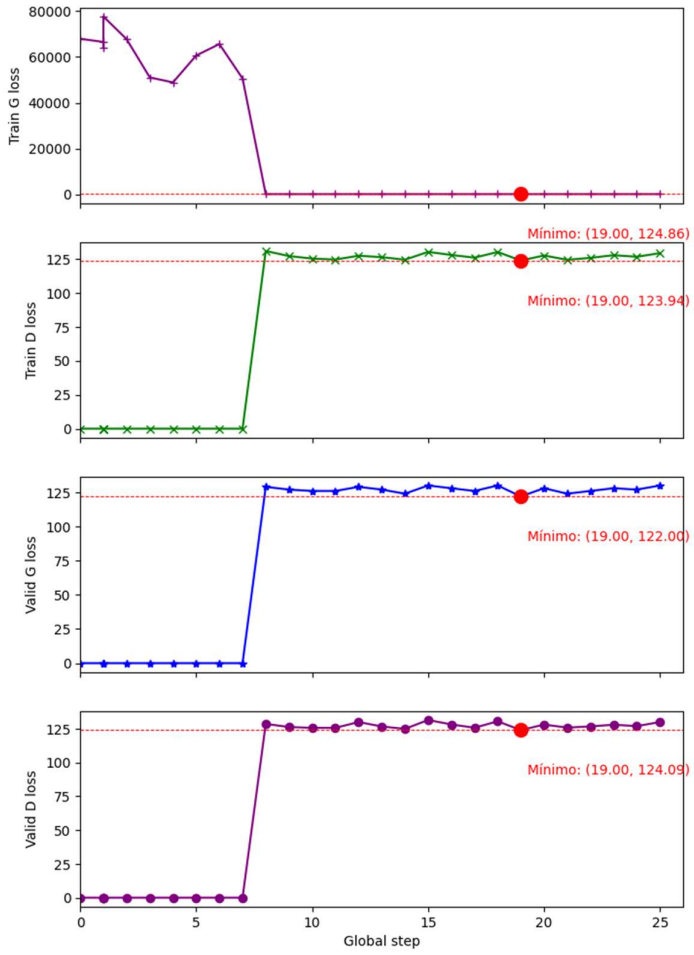

Ilustración 19 – Separación curvas loss entrenamiento Modelo 8

Durante el proceso de entrenamiento y validación del modelo 8, se puede observar que los valores de las curvas de pérdida para el generador y discriminador se encuentran en el rango de 120, con mínimos que alcanzan el valor de 123 en la etapa 19 del entrenamiento.

Esta evolución de los valores muestra una mejora significativa en comparación del modelo 6, ya que se ha logrado una reducción aproximada de 1/3 en la pérdida total. Esta disminución muestra que el modelo 8 ha mejorado su capacidad de generar secuencias más precisas y coherente que el modelo anterior.

3. Modelo 10

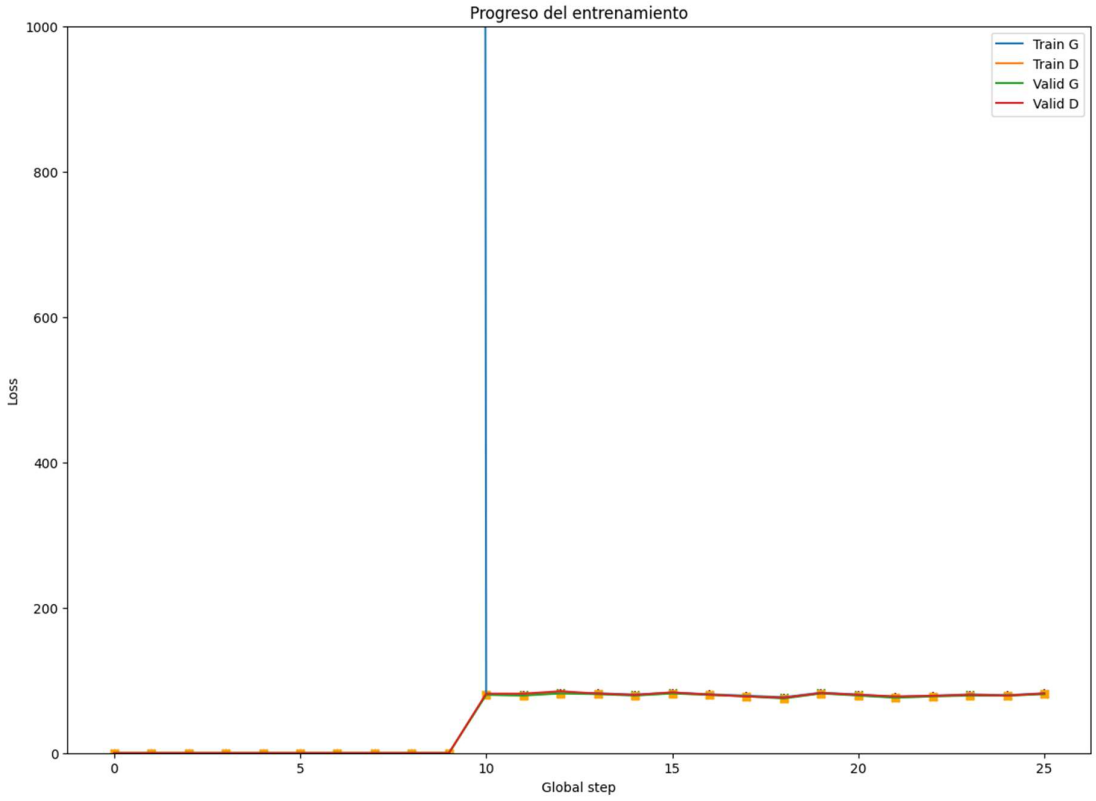

Ilustración 20 - Ampliación curvas loss entrenamiento Modelo 10

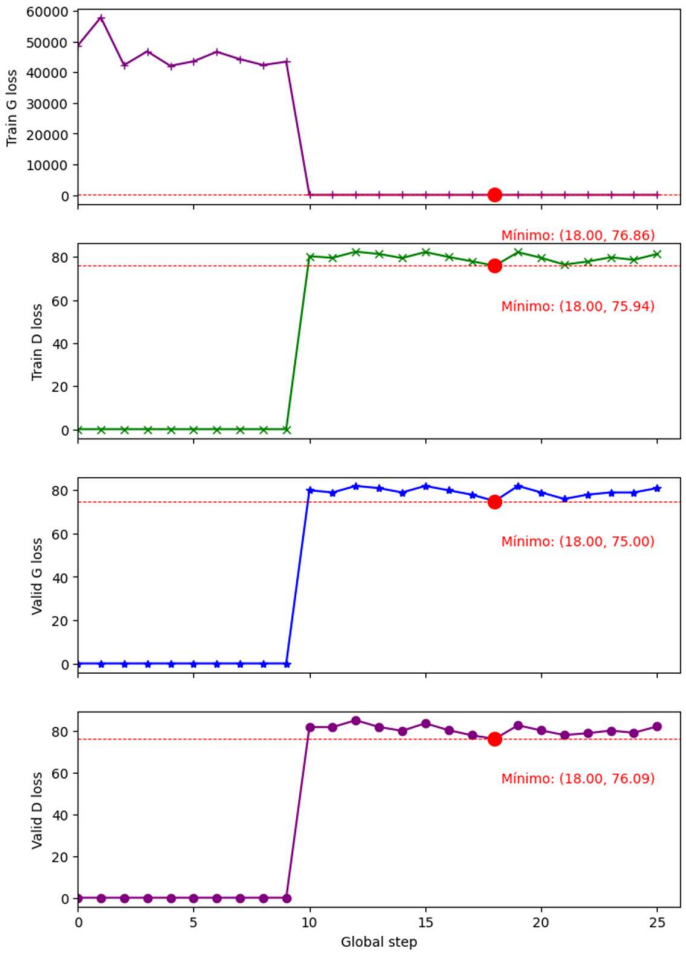

Ilustración 21 – Separación curvas loss entrenamiento Modelo 10

En las gráficas se refleja como los valores de la función de pérdida se encuentran en torno a 80, con mínimo cercanos a 76, en la etapa 18.

Esta observación revela una mejora en comparación al modelo 8, donde se ha logrado una disminución de un 40 % de la pérdida total. La reducción de esta pérdida se debe a la correcta asignación de los hiperparámetros y las nuevas funciones implementadas.

4. Modelo 12

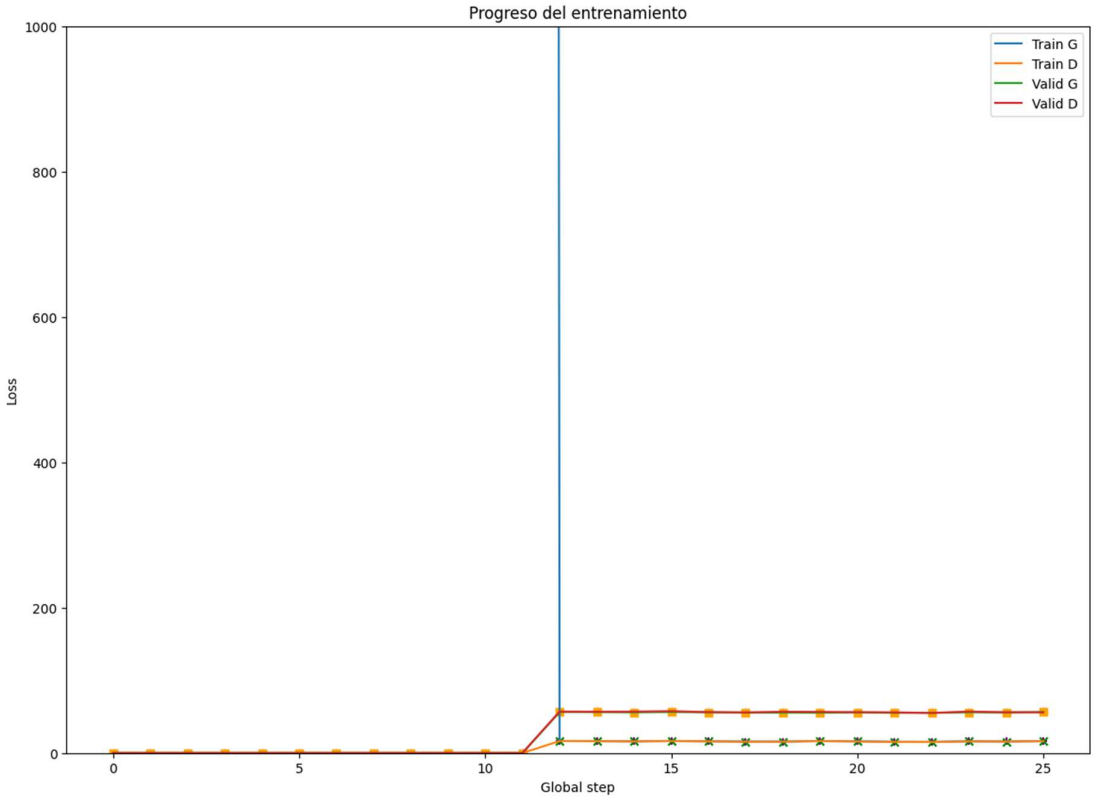

Ilustración 22 - Ampliación curvas loss entrenamiento Modelo 12

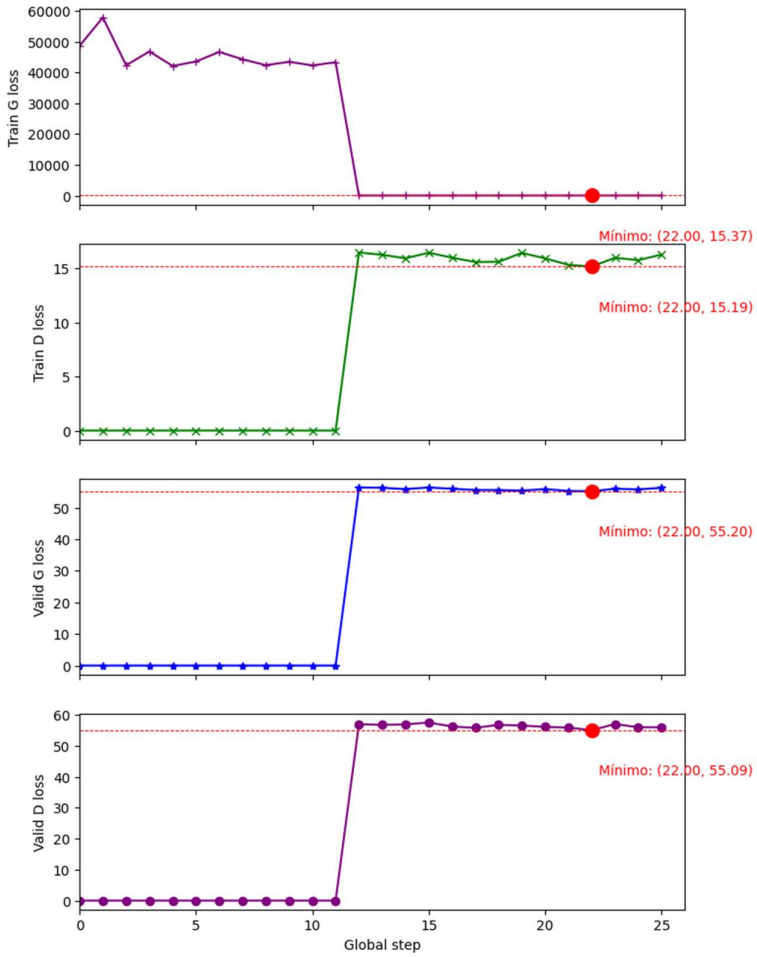

Ilustración 23 – Separación curvas loss entrenamiento Modelo 12

En este caso se observa una diferencia significativa de los valores de pérdida entre el entrenamiento y la validación. En particular, los valores de pérdida del generador y discriminador durante el entrenamiento son considerablemente más bajos durante el entrenamiento, de entorno a 20, en comparación a los valores de validación, en torno a 50.

La disparidad de valores de pérdida muestra como este modelo está experimentando sobre entrenamiento, es decir, está ajustando en exceso a los datos de entrenamiento y tiene dificultades cuanto se introduce un conjunto distinto a ese.

5.3.2. Pérdida prueba

En este apartado se realiza nuevamente la examinación de las curvas de pérdida de los modelos 6,8,10 y 12, pero esta vez para el conjunto de prueba. Estas gráficas mostraran como se comportan los datos cuando se introducen datos que no han sido vistos por el modelo anteriormente.

1. Modelo 6

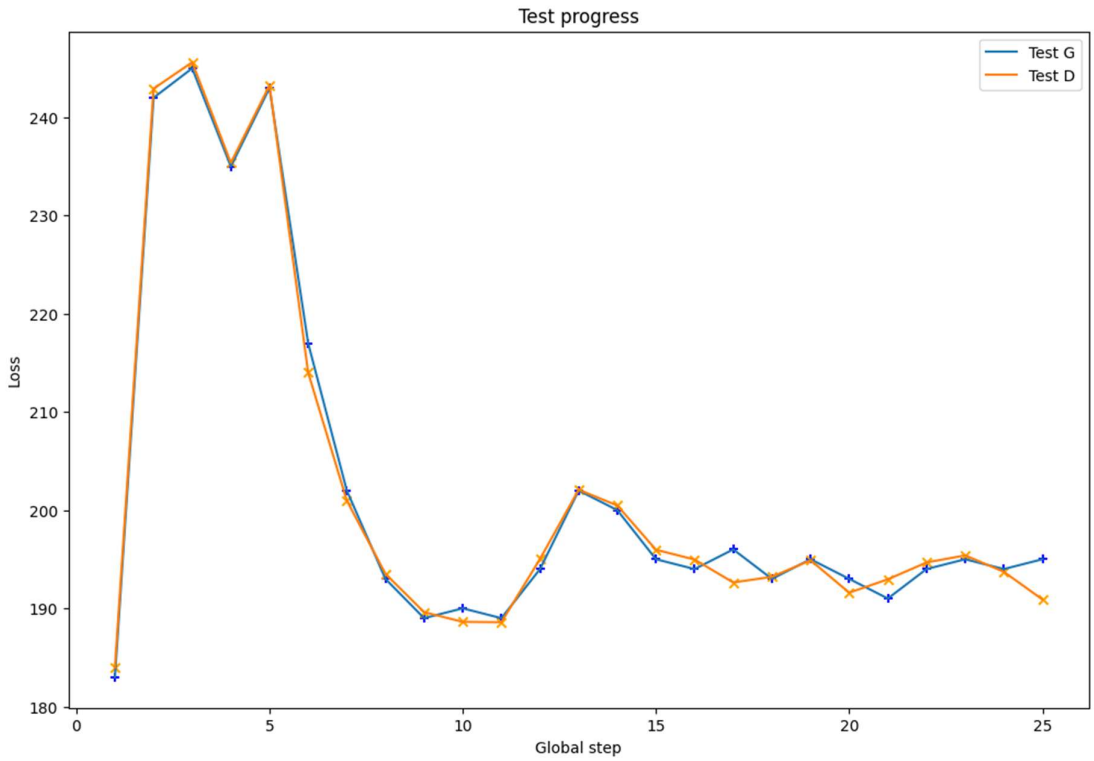

Ilustración 24 - Curvas loss test Modelo 6

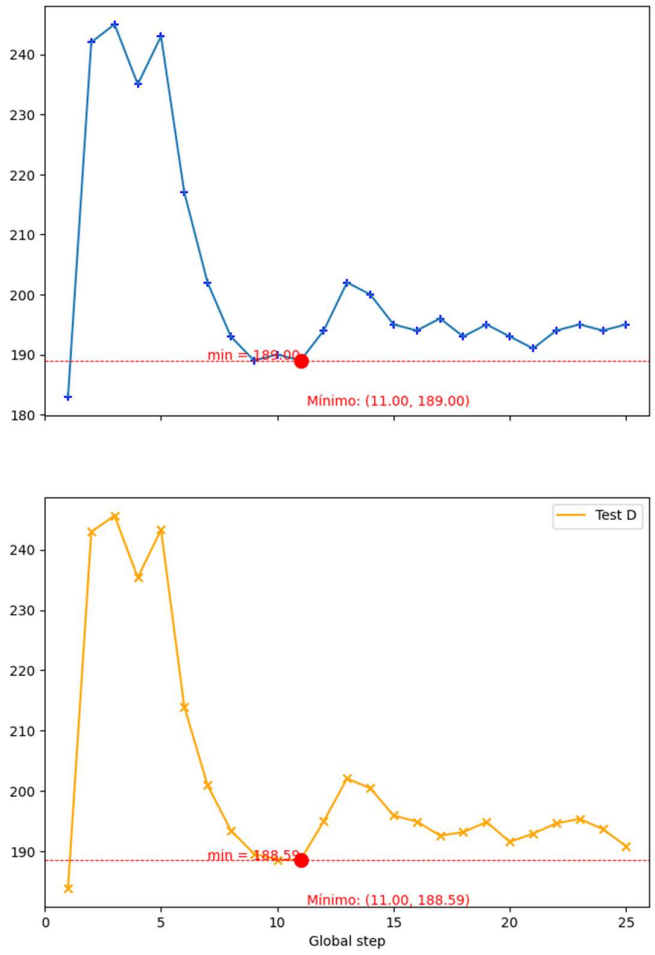

Ilustración 25 – Separación curvas loss test Modelo 6

2. Modelo 8

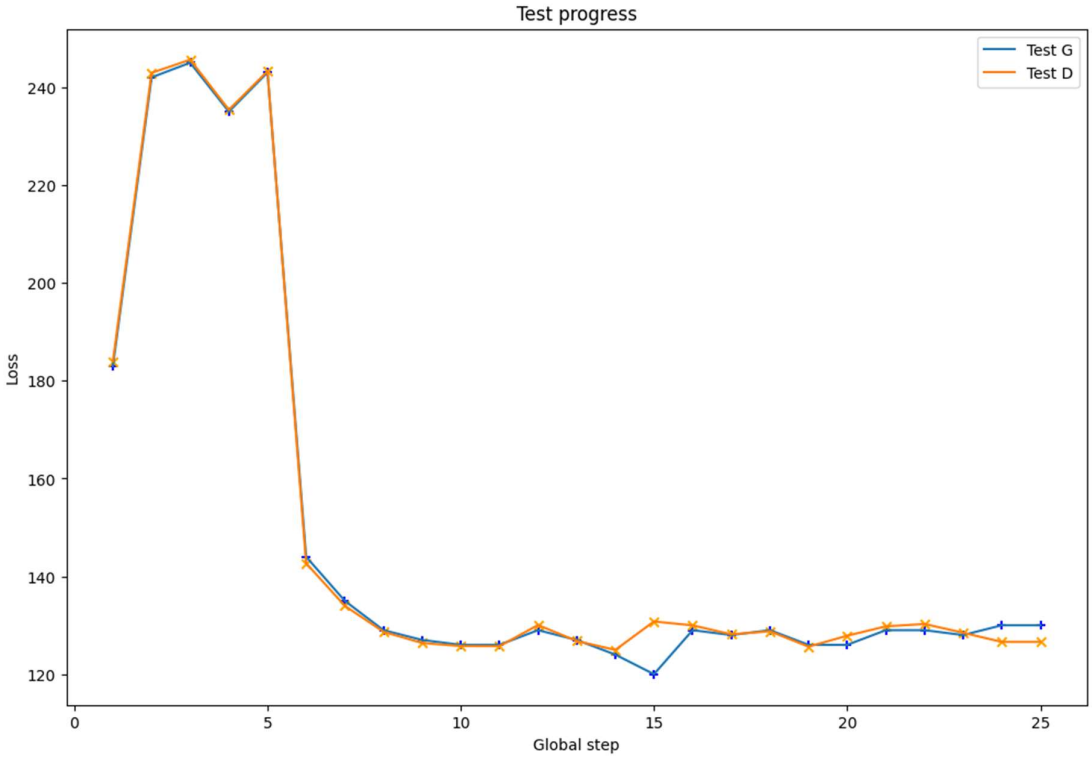

Ilustración 26 - Curvas loss test Modelo 8

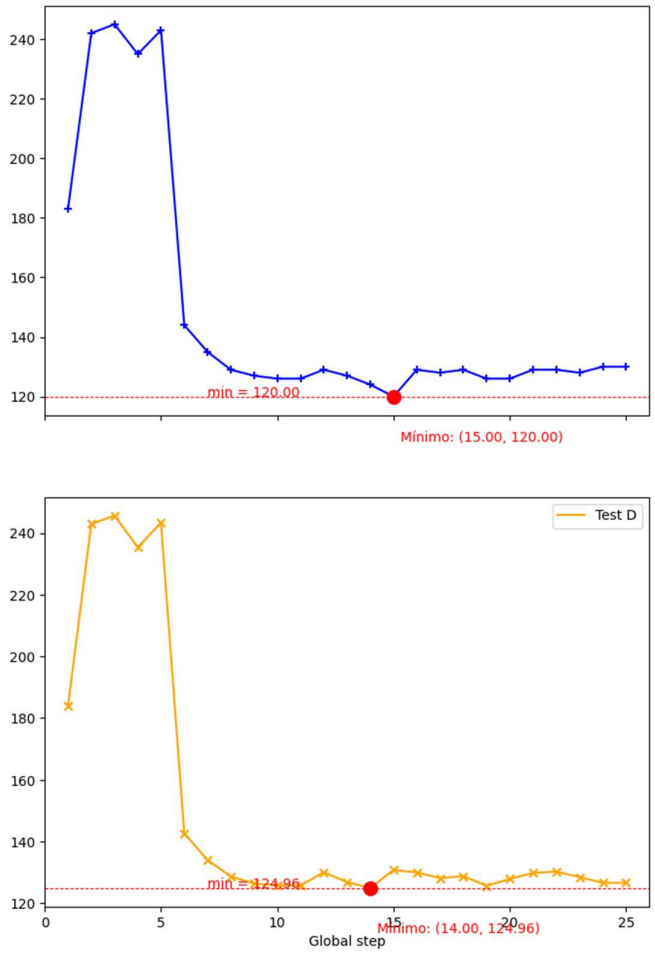

Ilustración 27 - Separación curvas loss test Modelo 8

3. Modelo 10

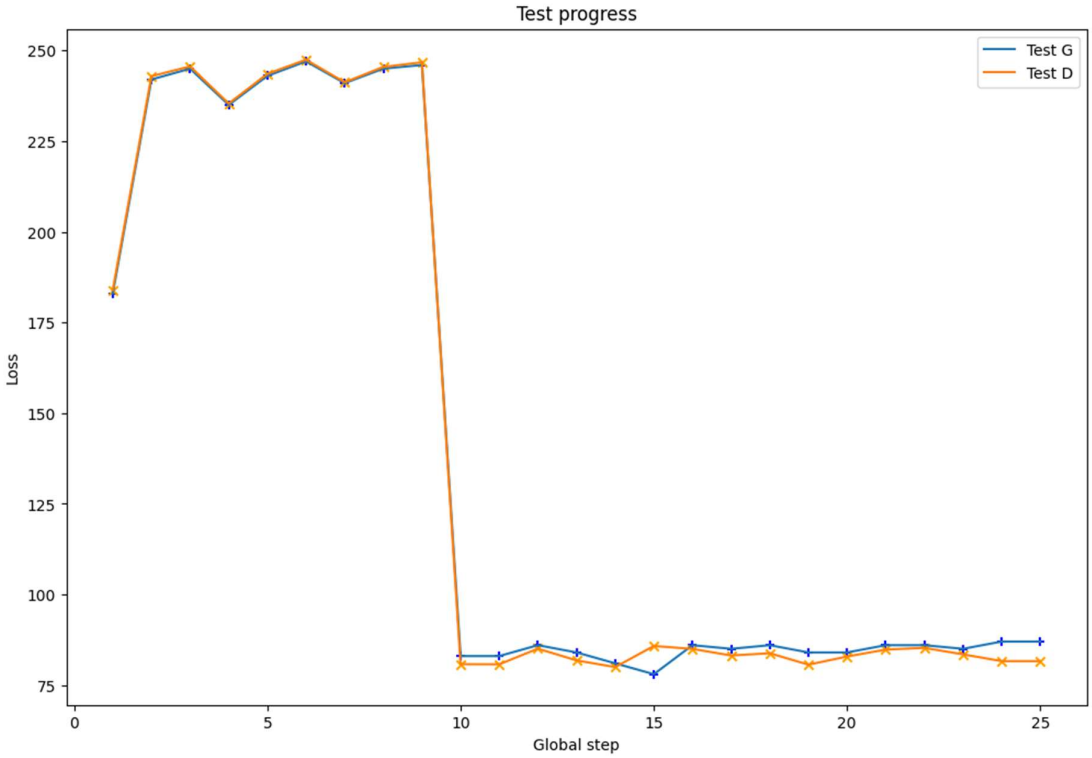

Ilustración 28 - Curvas loss test Modelo 10

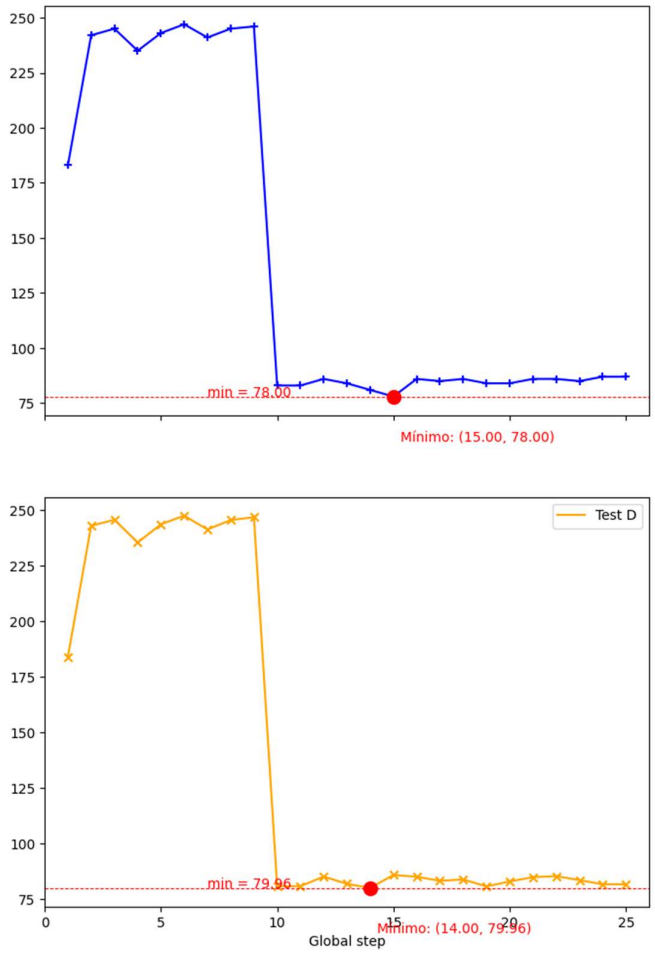

Ilustración 29 - Separación curvas loss test Modelo 10

4. Modelo 12

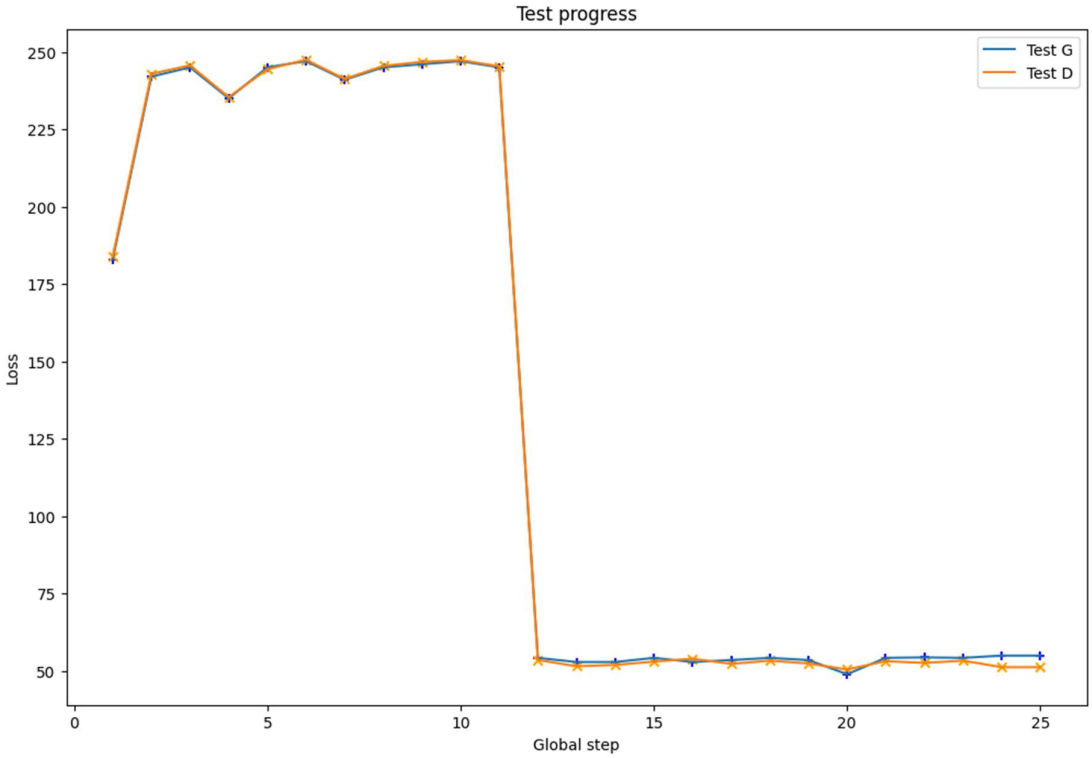

Ilustración 30 - Curvas loss test Modelo 12

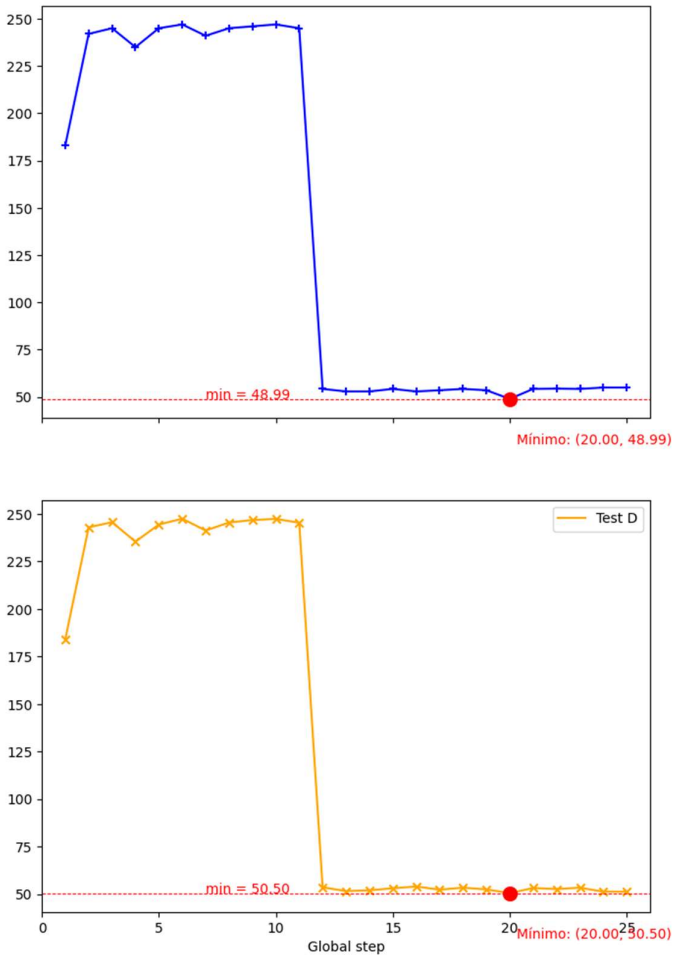

Ilustración 31 - Separación curvas loss test Modelo 10

Analizando las gráficas de pérdida en el conjunto de prueba o test, se puede observar que los valores de pérdida obtenidos con los datos de prueba son consistentes con los valores de pérdida obtenidos en los conjuntos de entrenamiento y validación. Esto indica que los modelos tienen la capacidad de generalizar y producir resultados similares en diferentes conjuntos de datos.

Al comparar los distintos modelos, se llega a la conclusión de que el modelo 10 es el que ha logrado una mejor calidad y capacidad de generalización, ya que es el que muestra un menor valor de pérdida total en comparación con los demás modelos.

Por otro lado, el modelo 12 muestra una diferencia entre el valor de pérdida de entrenamiento y en el conjunto de prueba, lo que confirma nuevamente que se produce sobreajuste en dicho modelo.

5.3.3. Comparación tiempos entrenamiento

En esta sección se muestra una tabla con los tiempos de ejecución de las etapas de entrenamiento de cada uno de los modelos, en segundos, siendo una manera de evaluar el rendimiento de la red.

Etapa	Modelo 6	Modelo 8	Modelo 10	Modelo 12
0	283	274	254	345
1	359.41	355.46	323.58	452.43
2	455.51	459.73	410.72	593.04
3	578.12	594.64	520.93	777.02
4	731.91	770.09	661.21	1020.53
5	926.64	999.11	851.78	1338.68
6	1170.68	1297.85	1090.42	1753.15
7	1478.76	1682.40	1412.24	2297.07
8	1869.67	2180.45	1800.64	3011.90
9	2369.95	2818.56	2277.82	3949.58
10	2996.24	3627.49	2865.43	5181.91
11	3789.38	4647.24	3597.87	6793.80
12	4799.32	5933.76	4506.85	8901.62
13	6076.59	7567.86	5636.54	11659.64
14	7689.83	9648.91	7038.43	15282.46
15	9715.33	12318.81	8760.58	20025.55
16	12295.42	16000.43	10863.81	26230.73

17	15566.43	21008.95	13697.78	34347.12
18	19696.58	27630.22	17307.96	45036.39
19	24962.77	36262.29	21883.63	59024.46
20	31598.45	47454.46	27688.48	77342.15
21	40086.61	61904.86	35167.77	101319.76
22	50867.92	80466.53	44726.71	132774.72
23	64455.32	103992.92	56942.94	173942.45
24	81668.63	133496.40	72781.63	229740.41
25	103497.22	170064.59	93622.70	305267.85

Ilustración 32 - Tabla tiempos de ejecución etapas de entrenamiento

Al analizar la tabla de tiempos de ejecución de las etapas de entrenamiento, se puede observar que el Modelo 10 ha logrado resultados buenos respecto al resto y se asemeja en gran medida al Modelo 6 en términos de tiempo de ejecución. Sin embargo, es importante destacar que el Modelo 10 ha logrado reducir aún más el valor de la pérdida (loss) en comparación con el Modelo 6. Esto indica que el Modelo 10 ha logrado optimizar su rendimiento y ha mejorado la calidad de las predicciones generadas. Por lo tanto, se puede concluir que el Modelo 10 muestra los mejores resultados en cuanto a eficiencia y rendimiento, al tiempo que garantiza una menor pérdida en el proceso de entrenamiento.

5.4. MATRIZ DE CONFUSIÓN Y ANÁLISIS DE PRECISIÓN

En este apartado, se presenta la matriz de confusión binaria para los modelos evaluados y se realiza un análisis de su precisión.

La matriz de confusión es una herramienta usada para evaluar el rendimiento de un modelo ya que permite visualizar y cuantificar la relación entre las etiquetas reales y las predicciones realizadas por el modelo. Esta matriz cuenta con cuatro celdas principales, cada una de las cuales representa una combinación especifica de clasificaciones realizadas por el modelo.

1. Verdaderos positivos (TP)

Se refiere a las muestras que son clasificadas correctamente como muestras generadas por el modelo. Estos casos indican una coincidencia entre la etiqueta real y la predicha, confirmando que el modelo genera música de calidad.

2. Falsos positivos (FP)

Corresponde a las muestras que son clasificadas erróneamente como generadas por el modelo, cuando en realidad son muestras reales. Estos casos se consideran errores del modelo, ya que predice incorrectamente una muestra real como generada.

3. Falsos negativos (FN)

Representa las muestras que son clasificada por el modelo como reales, a pesar de ser generadas por el modelo.

4. Verdaderos negativos (TN)

Se refiere a las muestras que son clasificadas como reales y no generadas por el modelo, indicando una coincidencia entre la etiqueta real y la predicha.

A partir de estos valores, se pueden calcular diversas métricas de evaluación como la precisión, el recall y el F1-score.

5. Precisión

Es la precisión de muestras clasificadas correctamente (TP) sobre el total de muestras clasificadas (TP+FP), indicando la exactitud del modelo para predecir muestras positivas.

$$Precisión = TP / (TP + FP)$$

6. Recall

También conocido como sensibilidad, es la métrica que indica la proporción de muestras positivas que se han clasificado de manera correcta en relación con el total de muestras positivas.

$$Recall = TP / (TP + FN)$$

7. F1-Score

Es una medida combinada de precisión y real que proporciona un equilibrio entre ambas métricas calculándose como la media armónica de precisión y recall.

$$F1 - score = 2 * (Precisión * Recall) / (Precisión + Recall)$$

A continuación, se muestra la matriz de confusión binaria para cada uno de los modelos 6., 8 y 10 y la tabla con los valores de precisión, recall y f1-score. Los valores se tomaron durante el proceso de entrenamiento y en total se estudiaron 126500 muestras (o tonos) para cada modelo.

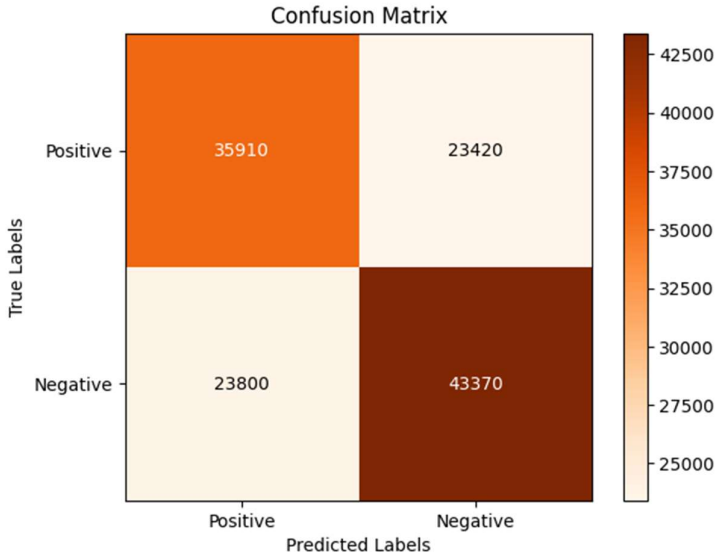

Ilustración 33 - Matriz confusión modelo 6

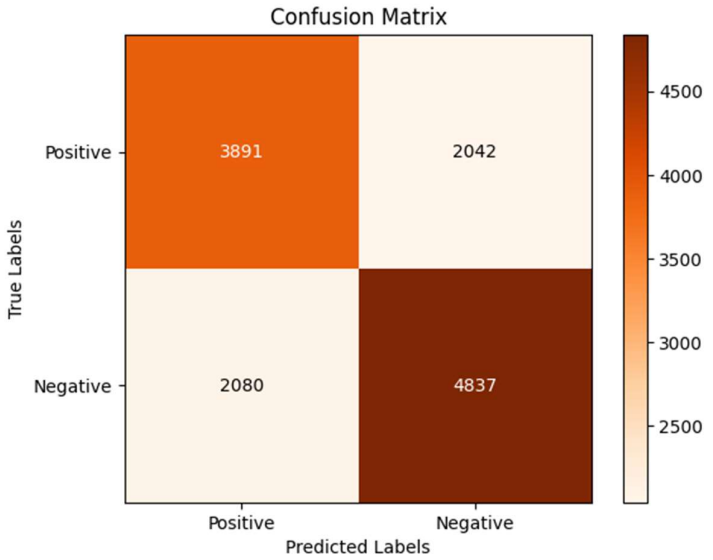

Ilustración 34 - Matriz confusión modelo 8

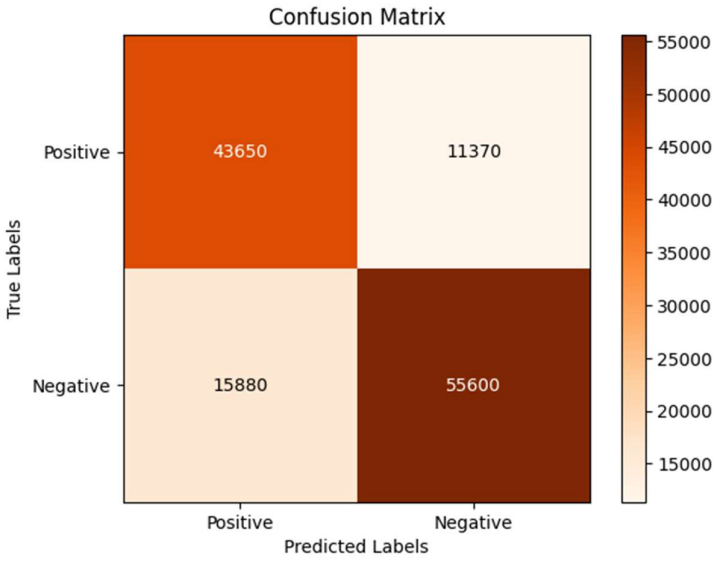

Ilustración 35 - Matriz confusión modelo 10

	Modelo 6	**Modelo 8**	**Modelo 10**
Precisión	0.605	0.656	0.793
Recall	0.601	0.652	0.733
F1-score	0.603	0.654	0.762

Tabla 9 – Valores métricas

Mediante los resultados obtenidos, se puede concluir que el modelo 10 muestra un rendimiento superior en comparación con los modelos 6 y 8. Esto se evidencia tanto en las métricas de precisión, recall y f1-score, donde el modelo 10 obtiene los valores más altos en todas las métricas, además de su matriz de confusión, donde presenta una mayor cantidad de verdaderos positivos y negativos, mientras que tiene una menor cantidad de falsos positivos y negativos.

5.5. MODELO FINAL

Tras analizar en detalle los aspectos planteados en los apartados 5.3 y 5.4, se determina que el modelo 10 destaca como el óptimo.

Esta conclusión se basa en la examinación de las curvas de las funciones de pérdida de los modelos, donde el modelo 10 muestra un valor medio menor que el resto, además de valores mínimos, sin llegar a producirse un sobre ajuste con los datos de entrenamiento, como ocurre con el modelo 12.

Además, al examinar los distintos modelos en términos de precisión en el contexto del proyecto, el modelo 10 presenta los mejores resultados, tanto en la matriz de confusión binaria, donde se observa una clasificación más equilibrada, como en las métricas de precisión, recall y f1-score, donde se obtienen valores más altos que el resto de modelos.

Por todo esto, se utilizará dicho modelo en la implementación de la aplicación web que se describe en el siguiente capítulo.

Capítulo 6. DESARROLLO APP WEB

Este capítulo se dedica a proporcionar un análisis detallado de la aplicación web desarrollada en este proyecto, utilizando el marco de angular, más concretamente, utilizando ngx-admin. Se describe de manera detallada como la interfaz de usuario permite la interacción con el modelo obtenido y las distintas funcionalidades a las que pueden acceder los usuarios.

El capítulo se organiza en distintas secciones para cubrir todos los aspectos relevantes del desarrollo de la aplicación web.

1) Se presenta el diseño y desarrollo de la interfaz de usuario, detallando de cómo se estructurado la navegación dentro de la aplicación y como se han diseñado los componentes visuales para facilitar la interacción del usuario.
2) Se detallan las funciones de la aplicación, desde la generación de música hasta el inicio de sesión y el uso de las API utilizadas.
3) Se detalla la creación de dos API utilizando flask que facilitan la comunicación entre la aplicación y la base de datos, permitiendo el manejo de las solicitudes.
4) Se aborda la integración del modelo en la aplicación y como se presentan las muestras generadas a los usuarios.

6.2. DISEÑO Y DESARROLLO INTERFAZ DE USUARIO CON ANGULAR

El objetivo del desarrollo de una aplicación web, refenciado en el apartado 4.2 es el de proporcionar una plataforma interactiva y amigable para la generación de música a través de nuestro modelo optimizado en el apartado anterior, mediante el uso de Angular y ngx-admin.

Angular es un marco de desarrollo de aplicaciones web robusto y escalable, por lo que permite una gestión efectiva de componentes y módulos complejos. Ofrece una arquitectura modular que facilita la organización del código, así como herramientas para el enlace de datos bidireccional, permitiendo una respuesta eficiente entre la interfaz de usuario y el backend. Además, es compatible con ngx-admin, una plantilla de administración basada en

angular que ofrece un conjunto de componentes de interfaz predefinidos, permitiendo un desarrollo más rápido y eficiente.

La interfaz de usuario de nuestra aplicación está compuesta por cuatro páginas principales:

1. Generador

Esta es la página principal, la cual es la encargada de permitir al usuario acceder a la generación de música.

Al acceder a la página solo se muestras dos opciones mediante botones,Ilustración 36 - Página genradro básica, mediante los cuales el usuario puede generar música o acceder a su biblioteca de música generada anteriormente, siempre y cuando haya iniciado sesión previamente.

Ilustración 36 - Página genradro básica

Durante la generación se genera una barra de progreso que muestra al usuario el progreso del modelo mientras genera una nueva muestra. Una vez generada la muestra, se muestra al usuario dos nuevos botones, Ilustración 37 - Página generador ejecución ,para que reproduzca o pause la muestra generada.

Ilustración 37 - Página generador ejecución

2. Biblioteca

Esta página permite al usuario visualizar y acceder de forma fácil e intuitiva a las muestras generadas por el modelo anteriormente, en la sesión de dicho usuario. Para ello, se presenta una tabla desplegable que muestra los detalles de cada canción incluyendo el autor (usuario que ha generado dicha muestra), titulo, id e ítems de la biblioteca. En la parte superior de dicha tabla se muestra una barra de búsqueda que permite al usuario filtrar las canciones de su biblioteca, Ilustración 38 – Página biblioteca.

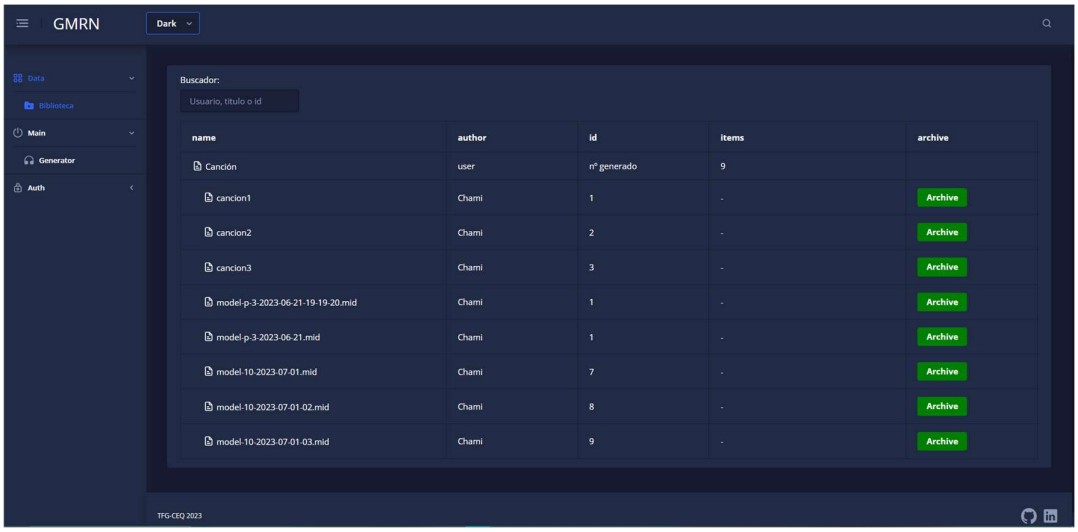

Ilustración 38 – Página biblioteca

Si se pulsa sobre el botón "Descargar", en cualquiera de las filas, se muestra un nuevo componente debajo de la tabla, que permite al usuario seleccionar el formato deseado para la descarga del archivo y realizar dicha descarga.

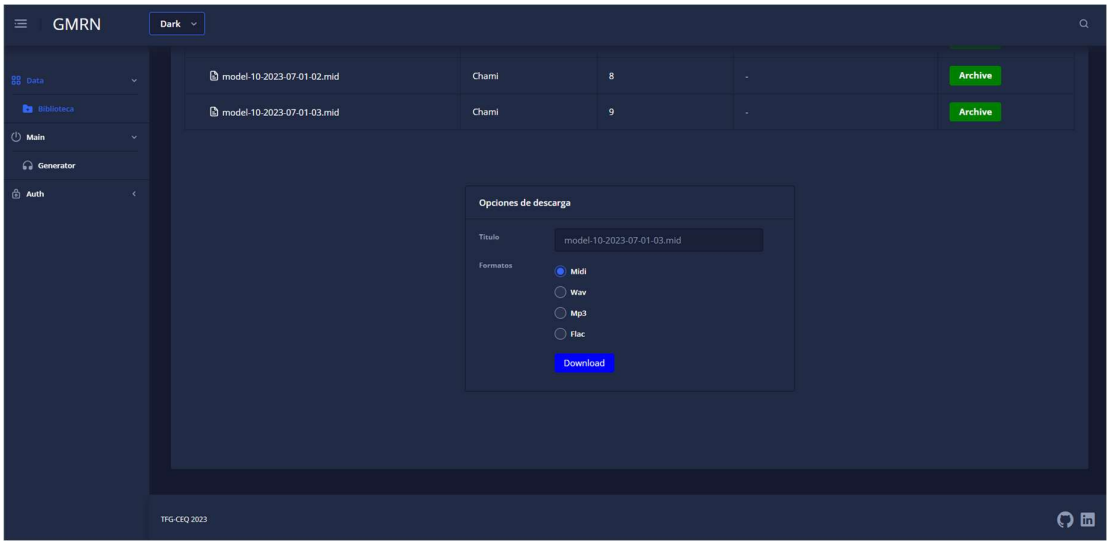

Ilustración 39 – Opción descarga

3. Login

Esta página presenta un formulario con campos para el nombre de usuario y la contraseña, para realizar el inicio de sesión. Si alguno de los campos es incorrecto aparecerá una notificación para que el usuario revise dichos campos.

4. Registro

El cometido de esta página es permitir al usuario crear una cuenta en la base de datos mediante un formulario. Si alguno de los campos introducidos por el usuario ya coincide con algún usuario ya registrado, aparecerá una notificación para que el usuario revise dichos campos.

La navegación entre páginas es facilitada por una barra de navegación vertical en la parte izquierda de la interfaz, la cual puede ser desplegable o plegada al pulsar sobre el icono de hamburguesa, situado en la parte izquierda de la cabecera. Esta cabecera muestra tres desplegables; Data, Main y Auth, conteniendo cada uno de estos desplegables el acceso a cada una de las páginas principales

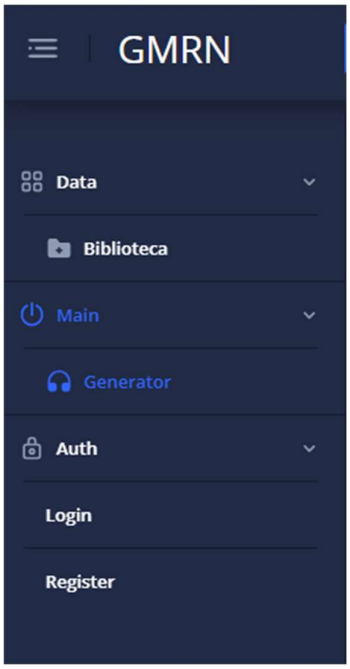

Ilustración 40 - Barra navegación

De esta manera la aplicación presenta una estructura de navegación intuitiva, como se puede apreciar en el siguiente diagrama de navegabilidad:

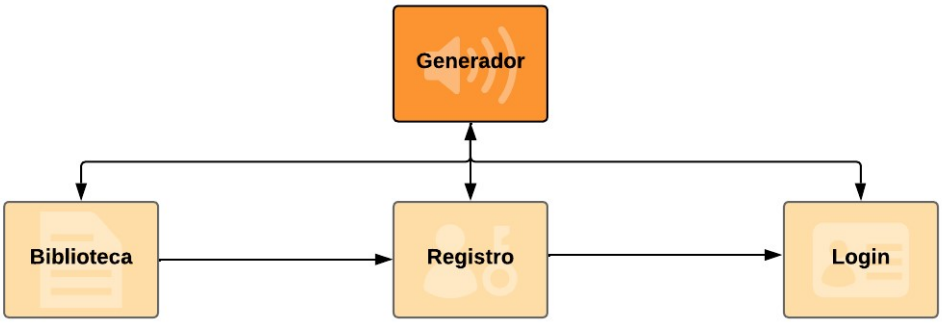

Ilustración 41 - Diagrama de navegabilidad de la aplicación

6.3. ALMACENAMIENTO Y GESTIÓN DE DATOS CON MYSQL

Para el almacenamiento y gestión de datos de la aplicación, se ha tomado la decisión de utilizar MySQL, un sistema de gestión de bases de datos relacionales, debido a su compatibilidad con el lenguaje de consulta (SQL) utilizado por las APIs desarrolladas para realizar consultas y manipulaciones.

La base de datos usada se denomina “users” y consta de dos tablas principales: “clientes” y “canciones”, cuya estructura se detalla en las siguientes (tablas):

Campo	**Tipo de valor**
Nombre	Varchar(255)
Pass	Varchar(255)

Tabla 10 - Tabla de clientes

Campo	**Tipo de valor**
Nombre	Varchar(255)
Titulo	Varchar(255)
Id	Int

Tabla 11 - Tabla de canciones

La tabla clientes (referencia) almacena la información de los usuarios registrados en la aplicación, almacenando el nombre de usuario y la contraseña asociada. La información de esta tabla es usada tanto para realizar el registro de nuevos usuarios como para el inicio de sesión de usuarios ya registrados.

La tabla canciones (referencia) se encarga de almacenar los detalles de las canciones generadas por los usuarios, guardando el nombre de usuario, título e id de la misma. La información de esta tabla es utilizada para mostrar la información de la biblioteca de un usuario.

Ambas tablas están relacionadas entre sí mediante el campo “nombre”, lo que permite establecer una relación de uno a muchos entre los usuarios y las canciones generadas por cada uno.

6.4. CREACIÓN DE APIS MEDIANTE FLASK

En la arquitectura de la aplicación web desarrollada, se han utilizado dos APIs para gestionar de manera eficaz y organizada las distintas funcionalidades requeridas. Dichas APIs han sido desarrolladas mediante el uso de Flask, un micro-framework de Python, por su versatilidad y eficiencia en la creación de aplicaciones web.

La primera API es la principal responsable de la gestión de la interacción del usuario con la aplicación. Esta API está compuesta por las siguientes rutas:

1. Login

Esta ruta es la dedicada a la autenticación de los usuarios. Cuando un usuario intenta iniciar sesión en la aplicación, los datos ingresados en el formulario de inicio de sesión se envían a esta ruta mediante una solicitud de tipo POST. Dentro de la API, los datos se comparan con los almacenados en la base de datos de usuarios. La API responde con un mensaje de éxito o error, en función de si las credenciales aportadas por el usuario son válidas o no.

2. Register

Esta ruta tiene un funcionamiento similar a la de Login. Su cometido es el de creación de cuentas de nuevos usuarios. Cuando un usuario llena el formulario de registro de la aplicación web, se envía una petición de tipo POST, que es recibida por la API. Una vez es recibida esta petición, se la API verifica la disponibilidad del nombre de usuario y la contraseña. Si ambos están disponibles, se añade el nuevo usuario con su contraseña a la base de datos y se envía un mensaje de éxito. Si por el contrario alguno de estos campos no se encuentra disponibles, se envía un mensaje de error para que el usuario introduzca nuevos campos válidos.

3. Get Canciones

Esta rusta es fundamental para proporcionar al usuario acceso a las canciones que ha generado anteriormente. Cuando un usuario, que ha iniciado sesión, navega a la biblioteca

personal en la aplicación web, se envía una solicitud GET a esta ruta. La API, realiza una consulta a la base de datos para obtener la información de las canciones relacionadas a dicho usuario y devuelve dicha información en la respuesta.

4. Insertar canción

Cuando se genera una nueva pieza musical a través del modelo, si el usuario ha iniciado sesión, los detalles son enviados a esta ruta mediante una solicitud de tipo POST. La API se encarga de añadir dicha información a la base de datos para que pueda ser accedida posteriormente si el usuario accede a su biblioteca personal.

5. Rutas de conversión

Se compone de tres rutas, para convertir archivos midi a tipo wav, mp3 y flac. Estas rutas se encargan de convertir el archivo midi que el usuario desee descargar, el cual se envía a la API mediante una aplicación de tipo POST a la ruta correspondiente. La API ejecuta la conversión de dichos archivos mediante funciones del apartado 6.5 y devuelve el archivo en el formato requerido, como se muestra en la siguiente figura:

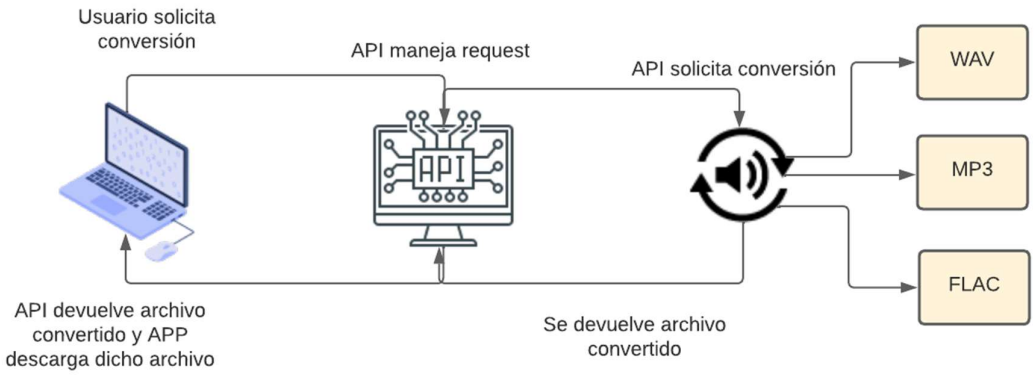

Ilustración 42 – Rutas conversión API

La segunda API, la cual se ejecuta en un puerto distinto a la anterior, es la encargada de ejecutar el modelo mediante la siguiente ruta:

1. Ejecutar script

Esta ruta se encarga de manejar la ejecución del modelo desarrollado. Cuando el usuario decide generar una pieza musical, se envía una solicitud de tipo GET. La API toma dicha solicitud, ejecuta el modelo y devuelve el archivo de música generado.

Estas APIs son el puente entre nuestra aplicación y la base de datos de la aplicación, por lo que juegan un papel fundamental y especifico en el correcto funcionamiento de esta. La combinación de estas APIs resulta en un backend que se comunica de manera correcta y trabaja con la interfaz de usuario para proporcionar una experiencia fluida y eficiente.

6.5. FUNCIONAMIENTO DE CONVERSIÓN ARCHIVOS MIDI

La conversión de archivos MIDI a los formatos WAV, MP3 Y FLAC está justificada en relación con los objetivos planteados en el apartado 4.2, además de garantizar al usuario una experiencia de reproducción más amplia, pues los archivos de tipo MIDI tienen limitaciones en términos de compatibilidad de reproductores.

La elección de los formatos elegidos para realizar dicha conversión es la siguiente:

1. **Formato WAV (Wave Audio File Format)**

Este formato es conocido por ser su característica de tipo sin pérdida que almacena datos de audio en forma de ondas no comprimidas, por lo que ofrecen una calidad de audio de alta fidelidad y una reproducción precisa. Esto también resulta en que el tamaño de estos archivos sea relativamente mayor en comparación con otros formatos comprimidos.

En comparación a MIDI, WAV es un formato estándar ampliamente utilizado en aplicaciones multimedia, siendo compatible con la mayoría de los reproductores de audio y software de edición de sonido.

2. **Formato MP3 (MPEG-1 Audio Layer 3)**

Es uno de los formatos de audio más populares y ampliamente utilizado en la actualidad. Es un formato de compresión con pérdida, por lo que se reduce el tamaño del archivo original eliminando cierta información de audio que sea menos perceptible para el oído humano. Pese a esto, el formato MP3, ofrece una excelente relación de compresión, aunque se produzca una pérdida mínima de calidad.

Como WAV, MP3 es un formato más extendido y usado que MIDI, por lo que puede reproducirse en una amplia gama de dispositivos y reproductores.

3. **Formato FLAC (Free Lossless Audio Codec)**

Es un formato de compresión sin pérdida diseñado específicamente para audio. A diferencia de MP3, este formato comprime los datos de audio sin perder ninguna información de calidad. Esto supone que los archivos FLAC ofrecen una calidad similar a los archivos WAV, pero con tamaños más pequeños debido a la compresión.

Este tipo de formato suele ser apreciado por los entusiastas del audio de alta calidad, sin embargo, no todos los reproductores y dispositivos admiten directamente este timo de formato.

El funcionamiento de la conversión de archivos MIDI a los formatos anteriores se ha realizado mediante tres funciones de Python. Estas funciones utilizan las bibliotecas midi2audio y FluidSynth, junto con la biblioteca pydub para realizar las conversiones necesarias. A continuación, se describen en detalle cada una de las funciones y su utilidad:

1. **Función midi_to_wav**

Esta función recibe como parámetro el archivo MIDI y realiza la conversión a formato WAV. Esta función utiliza la biblioteca midi2audio, que proporciona una interfaz para convertir los archivos MIDI a formatos comunes, y Fluidsynth, que es un sintetizador de sonido. En concreto se utiliza el método midi_to_audio de FluidSynth para realizar la conversión de MIDI a WAV. Su implementación es la siguiente:

```
import midi2audio
from midi2audio import FluidSynth
def midi_to_wav(midi_file):
    output_wav = midi_file.replace(".mid", ".wav")
    FluidSynth().midi_to_audio(midi_file, output_wav)

    return output_wav
```

2. Función midi_to_mp3

Igual que la función anterior, recibe el archivo MIDI y lo convierte en WAV siguiendo el mismo método, aunque el archivo de tipo WAV solo se guarda de forma temporal. Luego se utiliza la librería pydub, la cual es una biblioteca de manipulación de audio de código abierto escrita en Python, para convertir el archivo temporal WAV al formato MP3. Finalmente se elimina el archivo temporal WAV. Su implementación es la siguiente:

```
import midi2audio
from midi2audio import FluidSynth
from pydub import AudioSegment
import os

def midi_to_mp3(midi_file):
    output_mp3 = midi_file.replace(".mid", ".mp3")
    # Convertir MIDI a audio WAV
    wav_file = output_mp3.replace(".mp3", ".wav")
    FluidSynth().midi_to_audio(midi_file, wav_file)

    # Convertir audio WAV a MP3
    audio = AudioSegment.from_wav(wav_file)
    audio.export(output_mp3, format="mp3")

    # Eliminar archivo WAV temporal
    os.remove(wav_file)
```

```
    return output_mp3
```

3. Función midi_to_flac

Esta función es muy similar a la función midi_to_mp3, con el único cambio de que se utiliza la librería pydub para convertir el archivo temporal WAV al formato Flac. De igual manera se elimina el archivo temporal WAV una vez terminado el proceso de conversión. Su implementación es la siguiente:

```
import midi2audio
from midi2audio import FluidSynth
from pydub import AudioSegment
import os

def midi_to_flac(midi_file):
    output_flac = midi_file.replace(".mid", ".flac")
    # Convertir MIDI a audio WAV
    wav_file = output_flac.replace(".flac", ".wav")
    FluidSynth().midi_to_audio(midi_file, wav_file)

    # Convertir audio WAV a FLAC
    audio = AudioSegment.from_wav(wav_file)
    audio.export(output_flac, format="flac")

    # Eliminar archivo WAV temporal
    os.remove(wav_file)

    return output_flac
```

Todas las funciones devuelven el archivo convertido en el formato correspondiente, con el mismo nombre que el archivo original, cambiando la extensión por la adecuada al tipo de conversión realizada.

6.6. FUNCIONAMIENTO GLOBAL DE LA APLICACIÓN

El funcionamiento global de la aplicación se basa en un flujo de trabajo que permite a los usuarios, tanto registrados como no registrados, generar música, iniciar sesión y registrarse. Además, se permite a los usuarios que hayan iniciado sesión acceder a su biblioteca y realizar conversiones y descargas de las canciones en diferentes formatos.

El flujo principal del usuario comienza en la página principal de la aplicación, donde se encuentra el generador de música. En esta página, el usuario puede pulsar el botón "Generar" para iniciar el proceso de generación de música mediante el modelo desarrollado en el apartado 5.5, para lo que se envía la petición a la API "ejecutar_script". Durante el proceso de generación se muestra una barra de progreso que indica las etapas del proceso de generación, con el objetivo de que el usuario tenga una idea clara sobre el avance.

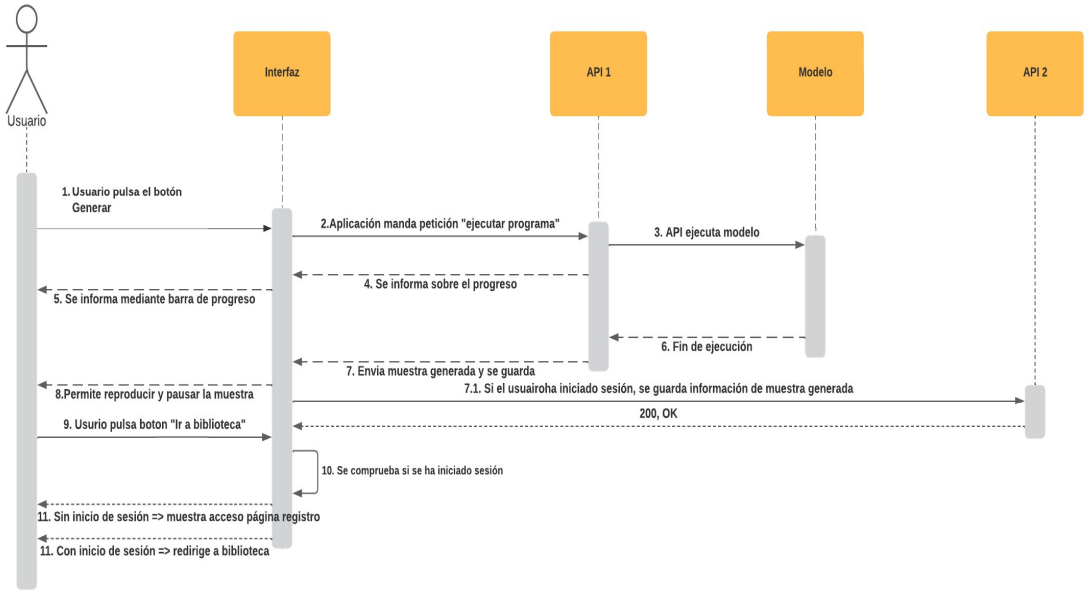

Ilustración 43 - Diagrama secuencia página Generar

Una vez se completa la generación de la muestra, la aplicación muestra al usuario el resultado obtenido, mediante la posibilidad de reproducir o pausar la muestra. La muestra será

guardada en la carpeta "Generated" dentro de la aplicación, para facilitar su posterior acceso. Si el usuario ha iniciado sesión previamente, cuando se recibe dicha muestra, la aplicación guarda la información de dicha canción, mediante la petición a la API en la ruta "insertar_cancion". Si el usuario quiere acceder a la biblioteca mediante el botón "Ir a la biblioteca" antes de iniciar sesión, se le redirige a la página de Registro desde la que también se puede acceder a la página de Inicio de sesión.

La página de la biblioteca solo mostrará la información a los usuarios que hayan iniciado sesión previamente, dando acceso a la página de Register directamente. Para ello realiza una consulta a la API mediante la ruta "get_canciones". Si un usuario pulsa sobre el botón descargar de alguna de las canciones de la tabla, y posteriormente selecciona el formato de descarga deseado, se envía una petición a la ruta correspondiente, cuya respuesta de descarga automáticamente una vez recibida.

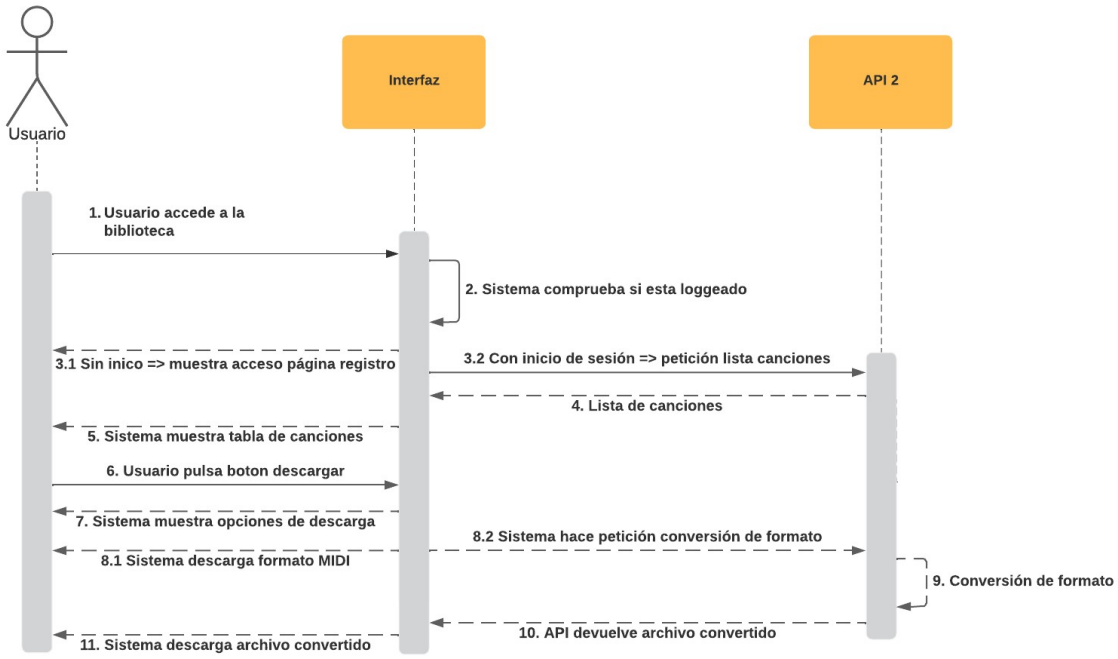

Ilustración 44 - Diagrama secuencia página Biblioteca

Las páginas del inicio de sesión y el registro, permiten al usuario autenticarse y crear una cuenta en a la aplicación. Estas páginas utilizan peticiones a la API mediante las rutas “login” y “register”. Dichas rutas se encargan de verificar que la información sea correcta, única y valida, informando en caso de que no lo sea en su respuesta para que dicho mensaje sea mostrado en la aplicación.

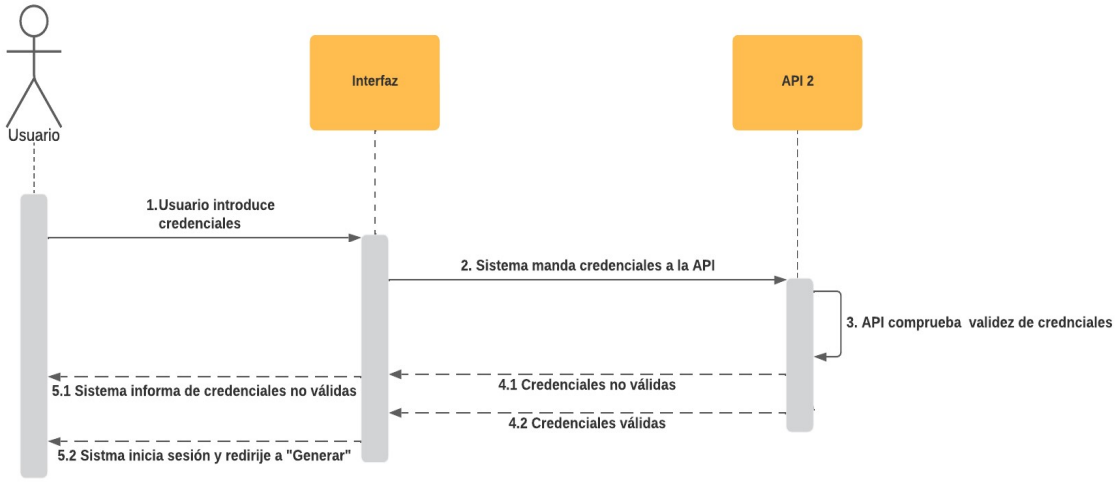

Ilustración 45- Diagrama secuencia página Login

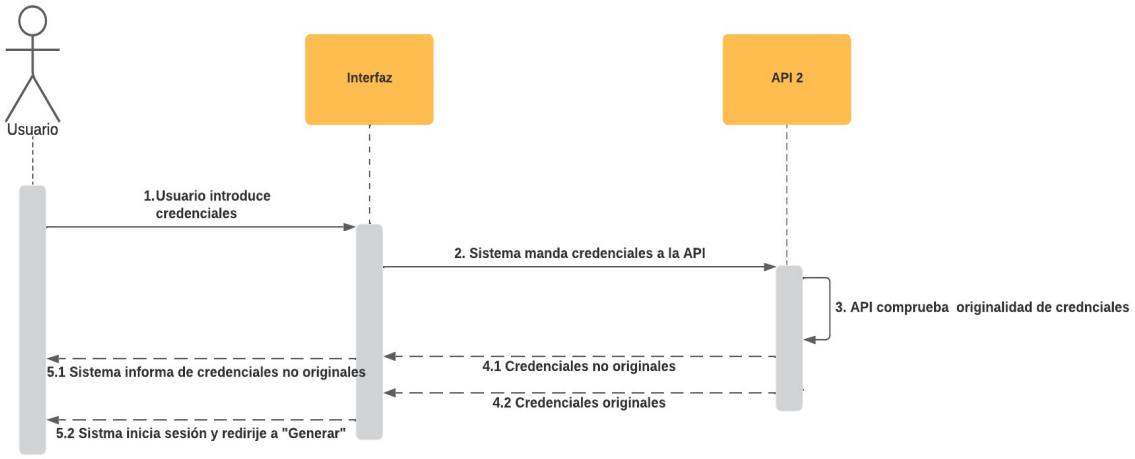

Ilustración 46- Diagrama secuencia página Register

Capítulo 7. ANÁLISIS DE RESULTADOS

En este capítulo se presenta un análisis de los resultados obtenidos con el modelo 10, el cual ha sido entrenado utilizando una biblioteca de muestras de archivos MIDI de piano diferente a la empleada anteriormente para comprobar su funcionamiento. El objetivo es evaluar el desempeño del modelo con nuevas muestras, ya que, si el modelo ha sido ajustado correctamente, los resultados deberían ser similares al tratarse de archivos MIDI del mismo tipo de instrumento.

Para ello se han utilizado 390 nuevas muestras de canciones de piano del género musical jazz, en el conjunto de entrenamiento. Esta variación de canciones proporciona una nueva base diversa para el entrenamiento del modelo, permitiendo obtener una visión más completa de las capacidades del modelo para generar muestras originales y aleatorias en otro contexto musical.

Al igual que el capítulo 5, se han utilizado diversas métricas para proporcionar una medida objetiva y cuantitativa del funcionamiento del modelo en términos de su capacidad para generar música coherente.

7.1. RESULTADOS DE GRÁFICAS DE PÉRDIDA

A continuación, se muestran las curvas de las funciones de pérdida para el generador y discriminador en las tres etapas siguientes:

- Validación (16 % de muestras utilizadas)
- Prueba (20 % de muestras utilizadas)
- Entrenamiento (64 % de muestras utilizadas)

Con el fin de diferenciarlo del modelo 10 original, se refiere a este modelo como modelo 10’.

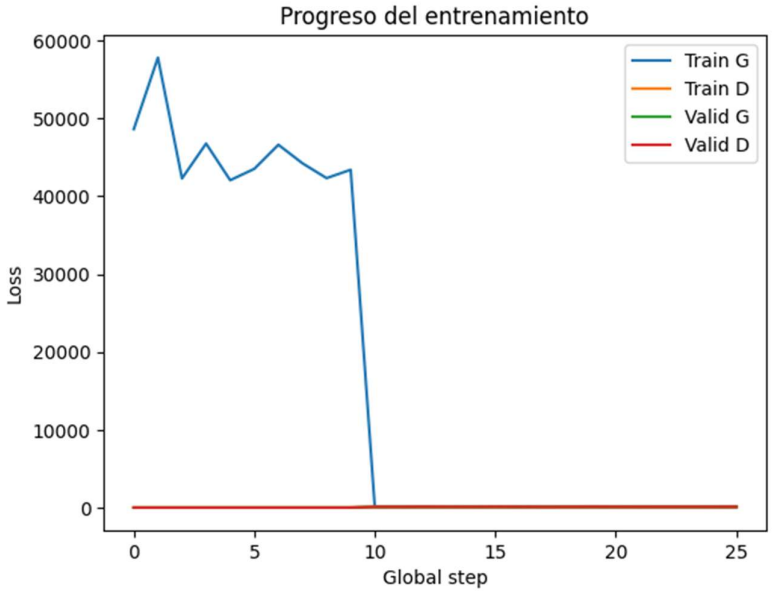

Ilustración 47 - Curvas loss entrenamiento Modelo 10'

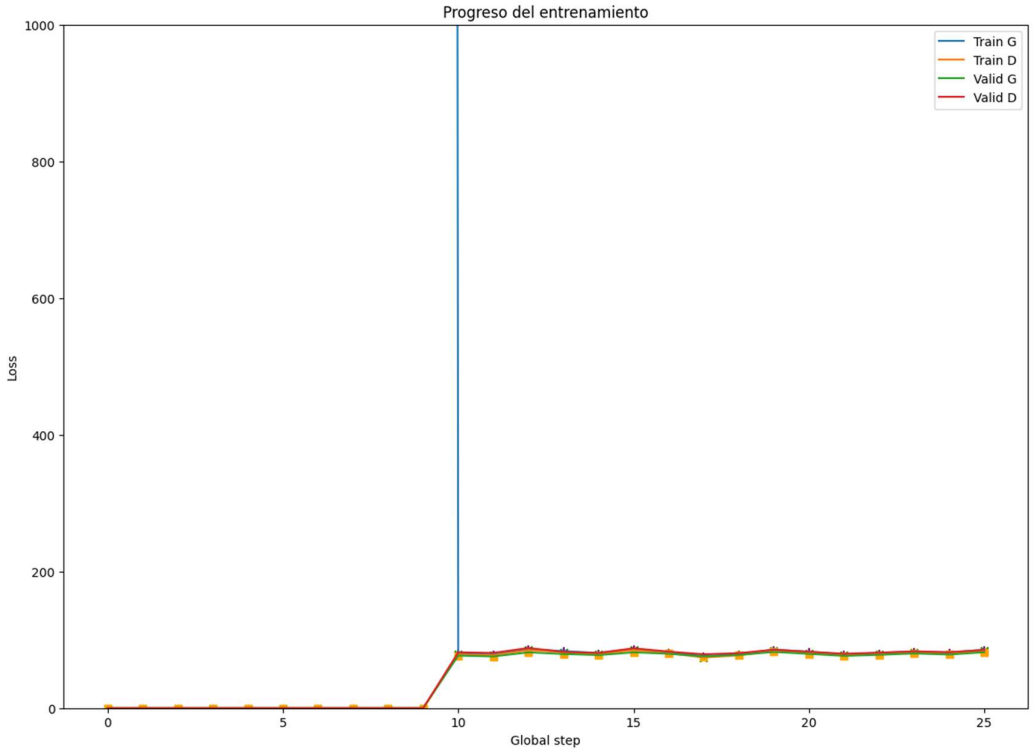

Ilustración 48 - Ampliación curvas loss entrenamiento Modelo 10'

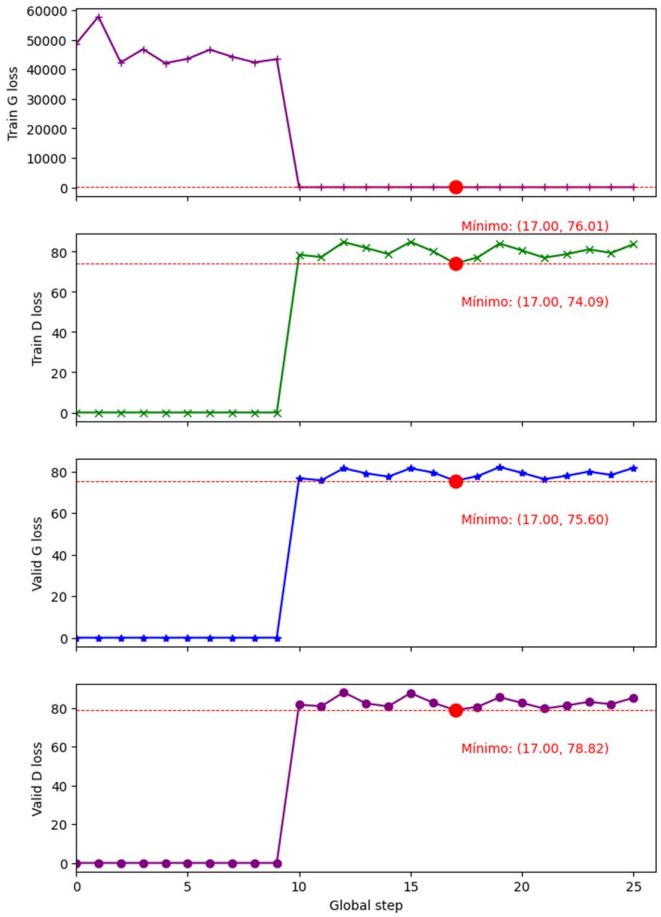

Ilustración 49 - Separación curvas loss entrenamiento Modelo 10'

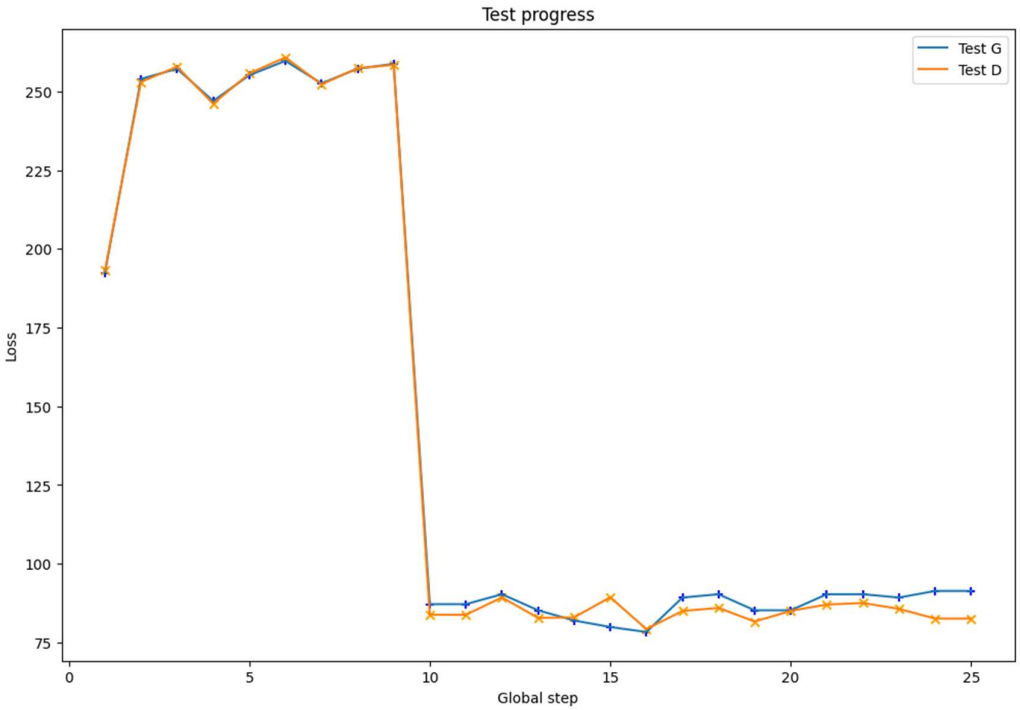

Ilustración 50 - Curvas loss test Modelo 10'

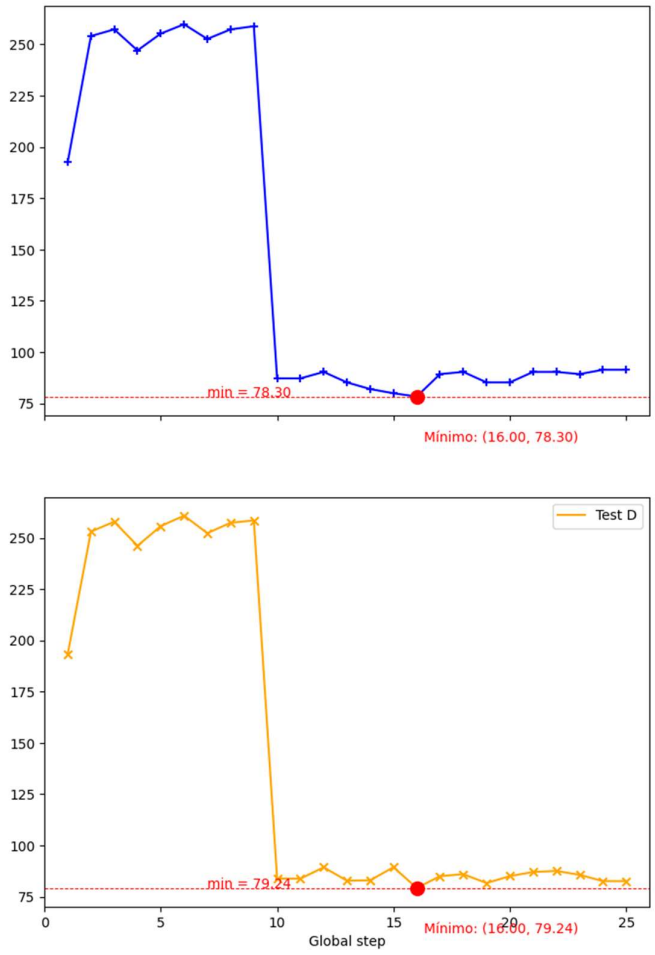

Ilustración 51 - Separación curvas loss test Modelo 10'

Tras analizar las curvas de las funciones de pérdida de este modelo mediante el análisis comparativo con los resultados previos, se demuestra que su funcionamiento es correcto y consistente, ya que las gráficas de perdida muestran valores similares, con una disminución gradual de ellos sin llegar al sobre ajuste de la red.

7.2. RESULTADOS DE MATRIZ DE CONFUSIÓN Y PRECISIÓN

De la misma manera que en el apartado 5.4, se ha realizado el análisis del modelo 10' mediante la matriz de confusión y métricas de precisión para evaluar su exactitud.

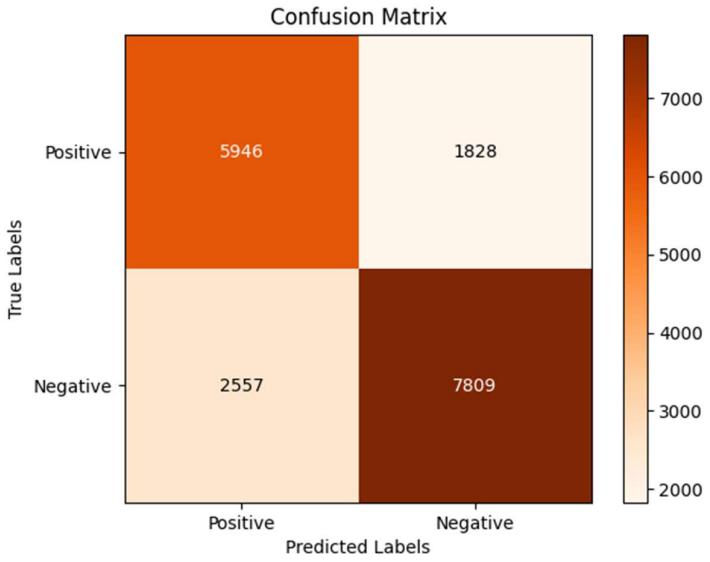

Ilustración 52 - Matriz de confusión Modelo 10'

Precisión	**Recall**	**F1-Score**
0.76	0.69	0.73

Tabla 12 - Métricas precisión Modelo 10'

El análisis de la matriz de confusión y las métricas de precisión del modelo 10' revela resultados similares a los obtenidos previamente. A pesar de que el número total de muestras de tonos en la matriz es de 18,140 debido a la adición de 390 nuevas pistas de la biblioteca de archivos MIDI de piano, las métricas de precisión, recall y F1-score muestran un desempeño consistente demostrando su capacidad para clasificar de manera precisa y equilibrada las muestras generadas. Estos resultados respaldan la calidad y la confiabilidad del modelo en la generación de música artificial.

Capítulo 8. CONCLUSIONES Y TRABAJOS FUTUROS

En este capítulo se comentan las conclusiones del proyecto, además de los objetivos cumplidos y las contribuciones realizadas del proyecto desarrollado.

8.1 CONCLUSIONES

El presente proyecto se ha desarrollado de manera exitosa, cumpliendo con los objetivos planteados y proporcionando soluciones al ámbito de la generación de música artificial. A continuación, se resumen las principales conclusiones:

1. Se ha logrado mejorar y optimizar el funcionamiento de un modelo de red neuronal basado en la arquitectura C-RNN-GAN. Esto se ha logrado mediante el estudio de las tecnologías empleadas en este tipo de redes y la evaluación del rendimiento de varios modelos mediante la optimización de sus hiperparámetros y la exploración de diferentes técnicas de normalización, eliminación de ruido y corrección de errores, mejorando la calidad de las muestras generadas.

   Además, se ha trabajado con diversas librerías de Python como tensorflow y python-midi para la ejecución de los modelos obtenidos, además de explorar distintas formas de evaluar la válidez de los modelos mediante curvas de pérdida, matrices de confusión y precisión de los modelos obtenidos.
2. Se ha logrado implementar de manera correcta una interfaz de usuario mediante una aplicación web que permita a los usuarios acceder y ejecutar el modelo obtenido. De la misma manera, se ha logrado implementar correctamente un back-end que permita la creación y manejo de los datos de los usuarios además de las canciones generadas y peticiones realizadas por la aplicación web mediante la implantación de flask api rest.

3. Se han explorado e implementado diversos métodos para la conversión de archivos de música de tipo MIDI a otros formatos para permitir al usuario obtener las muestras generadas en el formato deseado para su descarga.

8.2 TRABAJOS FUTUROS

A pesar de los objetivos cumplidos durante la elaboración de este trabajo, aún existen diversas áreas que podrían ser exploradas en trabajos futuros para mejorar el funcionamiento de este.

1. Mejora de la aplicación web para permitir a los usuarios definir los hiperparámetros del modelo

Se podría desarrollar una interfaz más avanzada que permita a los usuarios personalizar los hiperparámetros del modelo antes de realizar su ejecución. Sin embargo, se debe tener en cuenta que modificar los hiperparámetros de un modelo de redes neuronales es una tarea delicada, ya que cambios incorrectos pueden llevar a una inestabilidad o bajo rendimiento del modelo. Por lo tanto, es importante proporcionar orientación y restricciones claras para garantizar un uso adecuado de esta funcionalidad.

2. Mejora de la aplicación web para permitir el intercambio de muestras generadas entre usuarios

Sería interesante implementar un sistema que permita a los usuarios compartir las muestras de música generadas a través de la aplicación. Esto fomentaría la colaboración y el intercambio de ideas entre los usuarios, creando una comunidad en la que se puedan explorar y discutir las diferentes composiciones generadas.

3. Implementación de otros tipos de instrumentos en el modelo:

Actualmente, el modelo está enfocado en la generación de música para piano. Sin embargo, podría resultar interesante ampliar la variedad de instrumentos disponibles y permitir a los usuarios elegir entre diferentes opciones, como guitarra, violín, batería, entre otros. Esto

ampliaría las posibilidades creativas de los usuarios y permitiría generar composiciones musicales más diversificadas.

Capítulo 9. BIBLIOGRAFÍA

1. C-RNN-GAN: Continuous recurrent neural networks with adversarial training ,Olof Mogren ,Chalmers University of Technology, Sweden http://mogren.one/publications/2016/c-rnn-gan/mogren2016crnngan.pdf
2. Generative Adversarial Networks" (GANs) de Ian Goodfellow et al. (2014):https://arxiv.org/abs/1406.2661
3. Conditional Generative Adversarial Nets" de Mehdi Mirza et al. (2014) https://arxiv.org/abs/1411.1784
4. Long Short-Term Memory" (LSTM) de Sepp Hochreiter y Jürgen Schmidhuber (1997):https://www.bioinf.jku.at/publications/older/2604.pdf
5. Generating Sequences with Recurrent Neural Networks" de Alex Graves (2013):https://arxiv.org/abs/1308.0850
6. Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning. MIT Press.
7. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., ... & Ghemawat, S. (2016). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org. https://arxiv.org/abs/1603.04467
8. McKinney, W. (2010). Data structures for statistical computing in Python. En Proceedings of the 9th Python in Science Conference (Vol. 445, No. 2). https://conference.scipy.org/proceedings/scipy2010/pdfs/mckinney.pdf
9. Flask Documentation. (2023). Flask: Web Development with Python. https://flask.palletsprojects.com/
10. Angular Documentation. (2023). Angular: One framework. Mobile & desktop. https://angular.io/
11. C-RNN-GAN figure http://mogren.one/publications/2016/c-rnn-gan/mogren2016crnngan.pdf
12. LSTM figure https://www.researchgate.net/figure/Cell-of-LSTM-The-computing-node-is-composed-of-input-doors-output-doors-forget-doors_fig3_348375466
13. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks https://arxiv.org/abs/1511.06434
14. Music Generation with Recurrent Neural Networks https://arxiv.org/abs/1612.04928

15. Angular Development with TypeScript https://www.oreilly.com/library/view/angular-development-with/9781491902218/
16. Ngx-admin: https://akveo.github.io/ngx-admin/
17. Midi2audio https://pypi.org/project/midi2audio/
18. MySQL Explained: Your Step-by-Step Guide https://www.apress.com/gp/book/9781484214861
19. Repositorio de Github con el código del proyecto : https://github.com/Waterclau/GMRN-TFG.git

ANEXO I: ALINEACIÓN DEL PROYECTO CON LOS ODS

En este capítulo se muestra la relación que tiene este Trabajo de Fin de Grado con los Objetivos de Desarrollo Sostenible, también conocidos como ODS, aprobados por la ONU en 2015.

Este proyecto se alinea con el Objetivo de Desarrollo Sostenible (ODS) número 1 de la ONU: Poner fin a la pobreza en todas sus formas en todo el mundo. Aunque este proyecto no tiene como objetivo directo abordar la pobreza desde una perspectiva económica, contribuye a este objetivo al proporcionar un servicio de acceso público.

La pobreza no solo se refiere a la falta de recursos económicos, sino también a la privación de oportunidades y servicios básicos. El acceso a la música puede ser considerado como un componente de calidad de vida y bienestar. Al eliminar las barreras económicas y permitir que cualquier persona, independientemente de su situación financiera, pueda disfrutar de la música, se está proporcionando una experiencia enriquecedora que puede mejorar su bienestar emocional y cultural.

Una vez finalizado, el proyecto estará disponible de forma pública y accesible a través de internet, sin ningún costo de uso. Esto significa que cualquier persona con acceso a internet podrá utilizar el servicio sin restricciones económicas. Al ofrecer una solución de generación de música artificial, los usuarios podrán acceder a un número ilimitado de pistas de audio sin la necesidad de pagar suscripciones o tarifas adicionales. Este acceso libre y seguro a la música garantiza que todas las personas, independientemente de sus recursos económicos, puedan disfrutar de la música y beneficiarse de esta forma de expresión artística.

Al proporcionar un servicio de música gratuito y accesible, el proyecto contribuye a eliminar barreras económicas y garantiza que el acceso a la música no esté restringido por limitaciones financieras. Esto fomenta la igualdad de oportunidades y promueve la inclusión, permitiendo que personas de diferentes contextos socioeconómicos puedan disfrutar y explorar la música de manera equitativa. Además, al ser una aplicación basada en internet, el proyecto tiene el potencial de llegar a comunidades y regiones remotas que pueden tener acceso limitado a otros servicios de música.