Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

Grado en Ingeniería Informática

Trabajo Fin de Grado

Desarrollo de una Herramienta Basada en IA para el Apoyo en la Composición Musical

Autor: Victoria Fernández Alegría
Tutor(a): Miguel Antonio Barbero Álvarez
CoTutor(a): Alberto Belmonte Hernández

Madrid, Julio 2025

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

Trabajo Fin de Grado
Grado en Ingeniería Informática

Título: Desarrollo de una Herramienta Basada en IA para el Apoyo en la Composición Musical

Julio 2025

Autor: Victoria Fernández Alegría
Tutor: Miguel Antonio Barbero Álvarez
Departamento de Lenguajes y Sistemas Informáticos e Ingeniería de Software
Escuela Técnica Superior de Ingenieros Informáticos
CoTutor: Alberto Belmonte Hernández
Departamento de Señales, Sistemas y Radiocomunicaciones
Escuela Técnica Superior de Ingenieros de Telecomunicación
Universidad Politécnica de Madrid

A mi yaya, por enseñarme a aprender
y acompañarme siempre desde niña

Agradecimientos en particular a:
Mi familia, por hacerme sentir día a día afortunada,
Al Astudillo, por escucharme y apoyarme incondicionalmente,
y a Miguel y Alberto, que han hecho que este camino sea fácil.

Resumen

Cada vez está más presente el uso de herramientas que envuelven inteligencia artificial (IA) en todo tipo de procesos. Desde programación con herramientas como *Copilot*, integrado directamente en editores de código, hasta IAs integradas en plataformas de edición de fotos como *Photoshop*. Cada vez más personas incluyen estos herramientas como apoyo en sus procesos de creación.

En este trabajo se explora el desarrollo de una herramienta que funcione como un *Copilot* pero en el ámbito de composición musical. Es decir, un asistente que aporte sugerencias a fragmentos musicales existentes que el usuario esté creando, acompañándole en el proceso de composición.

Desde el principio de la construcción de esta herramienta se buscó un reto, con el objetivo de innovar en el proceso de entrenamiento e investigar técnicas a la vanguardia en este área. En este trabajo se explora la generalización de modelos de difusión existentes y enfocados en otros problemas como la reconstrucción de imágenes en condiciones de tiempo adversas, buscando su generalización y aplicación práctica a espectrogramas.

Para empezar, se emplearon espectrogramas, las imágenes que representan los audios de entrada, para el entrenamiento del modelo. Se buscó innovar con el *dataset*, construyendo un conjunto de datos propio. Este *dataset* está enfocado a recopilar una misma pieza musical en dos versiones distintas, una sencilla orientada a un músico principiante y otra más avanzada y compleja.

Tras el entrenamiento, se obtuvieron resultados mejores de los esperados y se continuó la implementación de esta IA de forma directa en una aplicación empleada usualmente para la composición. Se construyó un *plugin* sencillo en *Musescore* que integra el uso de este IA en el flujo usual de composición y además, se probó la utilidad de este tipo de herramienta directamente con una estudiante de música del Conservatorio profesional de música Victoria de los Ángeles.

Entonces, el trabajo engloba la exploración de este tipo de modelos, construcción de un *dataset* propio y el entrenamiento de un modelo con el objetivo final de integrarlo todo en una herramienta combinada con el uso de *Musescore* como búsqueda de un asistente a la composición.

Abstract

The use of artificial intelligence (AI) powered tools is becoming increasingly prevalent across all types of processes. From programming tools like *Copilot*, directly integrated into code editors, to AI features within photo editing platforms such as *Photoshop*, more and more individuals are incorporating these tools to support their creative processes.

This work explores the development of a tool that functions as a *Copilot* but applied to musical composition. Specifically, it is an assistant designed to provide suggestions for existing musical fragments a user is creating, accompanying them throughout the whole the composition process.

From the beginning of this tool's development, the aim was to tackle a challenge, with the goal of innovating in the training process and investigating cutting-edge techniques in this field. In this work, the generalization of existing diffusion models, primarily used for other problems such as image reconstruction under adverse weather conditions, is explored, seeking their practical application to spectrograms.

To begin with, spectrograms were used for model training. Spectrograms are the visual representations of audio. Innovation was also pursued with the dataset itself, by building a custom dataset. This dataset is focused on compiling the same music piece in two different versions: the simple one suited for a beginner musician and a more advanced and complex one.

After the training process, promising results were obtained, and the implementation continued. The AI was directly integrated into an application commonly used for composition. A simple plugin for MuseScore was built, which integrates the use of this AI into the usual musical composition workflow. Moreover, the tool's utility was directly tested with a student from the "Conservatorio profesional de música Victoria de los Ángeles".

Then, this work brings together the exploration of these types of models, the construction of a custom dataset, and the training of a model, with the main goal of integrating everything into a tool combined with the use of Musescore as a music composition assistant.

Tabla de contenidos

Capítulo 1

Introducción

1.1. Contexto y motivación

La IA generativa se encuentra ahora mismo en auge y en constante exploración. El paradigma predominante ha sido el uso de los Large Language Models (LLM), que seguramente sean el caso más cercano a la mayoría de población y perfiles no técnicos, con ejemplos como ChatGPT [1] o Gemini [2]. Existen incluso otras herramientas que han progresado de forma acelerada, como aquellas enfocadas a la generación de imágenes como Dall-e [3] hasta generación musical a partir de texto. Todas ellas han mejorado notablemente en poco tiempo.

La principal motivación de este proyecto es evitar que la generación musical se convierta exclusivamente en un proceso autónomo y totalmente independiente cuando puede ser una herramienta muy valiosa que sirva de apoyo en el proceso creativo de composición.

Está claro que la generación musical independiente es posible; pero siendo un proceso creativo que llevan a cabo la mayoría de los músicos hoy en día una herramienta tecnológica que les apoye en el proceso tendría sentido y serviría de gran utilidad. En primer lugar, esta herramienta demostraría que herramientas basadas en IA no se dedican exclusivamente a reemplazar, sino que también pueden servir de extensión creativa, sin perder el "toque personal", inevitablemente humano, de las piezas musicales. Además, de esta manera se lograría evitar una independencia absoluta permitiendo que los compositores puedan adaptar el resultado final a su visión propia, de forma flexible.

Además, a diferencia de soluciones existentes que generan música a partir de texto, esta herramienta funcionaría también recibiendo datos de audio como entrada, permitiendo que resultado final sea una solución accesible a perfiles no técnicos.

1.2. Claves del inicio del proyecto

Cuando un alumno está empezando a tocar un instrumento, va aprendiendo poco a poco. Por ejemplo, un pianista que lleva un año tocando el piano, interpreta piezas sencillas en comparación con las que interpretará en cinco años. Sin embargo, hay piezas clásicas que están presentes durante todo el período de aprendizaje de un músico. Beethoven, Mozart, Bach no compusieron distintas versiones de todas sus piezas para adaptarlas a todos estos distintos niveles por los que transiciona un alumno. Sin embargo, existen piezas complejas adaptadas a niveles iniciales, siendo posible adaptar una pieza compleja a una versión mucho más sencilla en función del nivel del alumno. ¿Y si se crease un *dataset* centrado en esta idea? La mayoría de los conjuntos de datos populares en música, se centran en temas de armonización. La idea sería construir un conjunto de datos nuevo, donde las piezas se agrupasen por dificultad: parejas de la misma pieza donde unas fueran más sencillas para principiantes y otras más complejas y completas.

A partir de este conjunto de datos, la clave del TFG surge con la pregunta: ¿y si existiera una herramienta capaz de sugerir versiones complejas a partir de una versión sencilla de la pieza? De esta manera, una pieza compleja no significará simplemente armonizarla, sino que implique musicalidad, cambios de ritmo, intensidad, etc.

Existen diversas técnicas en el área de IA musical generativa. La primera decisión que se tomó fue el formato de datos con el que se trabajaría, y surgió la idea de aplicar técnicas existentes en el área de imágenes aplicado en este caso en particular, usando espectrogramas de los audios. Se buscaba innovar y explorar áreas más recientes y entonces se entrelazaron dos ideas, a primera vista completamente distintas, pero en base similares. Un área explorada de la IA generativa es la reconstrucción de imágenes con condiciones de tiempo adversas [4], [5], [6]. Es decir, imágenes donde está lloviendo, nevando, etc, son procesadas para obtener las mismas imágenes pero sin estas condiciones adversas que "manchan" de alguna manera la imagen. Estas técnicas pueden ser empleadas para eliminar estas "manchas" o incluso hacerlo a la inversa, partir de imágenes sin estas condiciones y tras procesarlas que el resultado sean estas mismas imágenes "manchadas". ¿Cómo se podría aplicar en música? Entrenando un modelo al que le insertas un audio simple y devuelve este mismo audio "manchado" con nuevas ideas que puedan mejorarlo. Son técnicas que funcionan en otras áreas y la idea fue explorar su aplicación práctica en otro área como la música. Sería exactamente lo mismo pero en vez de alimentar el modelo directamente con audios, se haría con espectrogramas.

1.3. Descripción del problema y objetivos

El principal objetivo del proyecto es el entrenamiento de un modelo que sea capaz de "embellecer" los fragmentos musicales que reciba como *input*, incluyendo adornos o armonías generados automáticamente en función de ellos. El *output* generado se ofrecerá al usuario como sugerencia de "embellecimiento" al audio

suministrado como *input*, y servirá de apoyo en su proceso de composición. Este proceso se ha llevado a cabo explorando diferentes modelos preexistentes, seleccionando el definitivo y entrenándolo con el *dataset* creado gracias a la previa recopilación de datos. Todos los conjuntos de piezas musicales serán recopiladas y preprocesadas.

Para poder llevar a cabo este proyecto, se establecen estos objetivos:

- Estudio de la metodología empleada para el desarrollo de herramientas de *Deep learning*
- Adquirir conocimientos de las tecnologías necesarias: procesado y caracterización de audio, programación y despliegue de modelos de IA, desarrollo de software para interacción con IA
- Desarrollo de una base de datos de entrenamiento para fines específicos
- Diseño y evaluación de un prototipo
- Aplicación a un caso real
- Conclusiones sobre los resultados obtenidos

1.4. Estructura de la memoria

Esta memoria se divide en los siguientes capítulos:

Introducción Este apartado se centra en la motivación y los objetivos propuestos en el inicio del trabajo de fin de grado (TFG).

Estado del arte Este capítulo hace un extenso análisis sobre la Inteligencia Artificial Generativa y las aplicaciones existentes de la IA en música. Brinda pinceladas de las herramientas actuales y soluciones existentes en composición musical además de mencionar ciertas limitaciones de éstas. Por último, se analizan los *datasets* existentes, haciendo hincapié en el formato y características de los conjuntos de datos preexistentes.

Desarrollo Este capítulo está centrado en mostrar cuál ha sido la línea temporal de desarrollo y qué actividades se han llevado a cabo. Desde la selección de un modelo preexistente, a la construcción del *dataset*, el entrenamiento, la evaluación y pruebas. Se explica detalladamente todos los retos por los que ha pasado este trabajo siguiendo la línea temporal, con ciertos solapamientos entre unas tareas y otras.

Análisis de impacto En este capítulo se analiza el resultado final, los casos de uso y cuáles son sus ventajas frente a otras alternativas.

Conclusiones y trabajo futuro Aquí se presenta un breve resumen de los resultados, ideas sobre el trabajo a futuro y una reflexión final.

Anexo En él se incluye el *link* al repositorio del código.

Capítulo 2

Estado del arte

2.1. Inteligencia Artificial Generativa

Los modelos generativos son una rama dentro de la Inteligencia Artificial en constante cambio y crecimiento [7]. A partir de un conjunto de datos, los modelos de IA generativa entrenan, aprenden patrones y son capaces de crear nuevos datos similares a los proporcionados de entrada [8]. Estos conjuntos de datos pueden ser de cualquier tipo, incluyendo texto, imágenes o audio.

Por tanto, los modelos de IA generativa se pueden aplicar en una gran variedad de áreas según el tipo de datos proporcionados. Existe la generación de texto con los populares LLMs, como ChatGPT [1], Claude [9], o Gemini [2], o también la generación de imágenes en aplicaciones como DALL-E [3], ahora integradas en otras herramientas [8]. En el área musical también hay un gran interés desde hace años por explorar hasta dónde pueden llegar estos modelos generativos.

Además, en muchas ocasiones, los modelos generativos no solo son herramientas encargadas de generar resultados definitivos, sino que pueden llegar a ser "colaboradores" en el proceso creativo [7]. El uso de IA facilita la exploración de ideas en una primera fase de desarrollo de algún producto, cuando es vital la lluvia de ideas para generar muchas opciones y más tarde seleccionar y refinar [7]. La generación de ideas y creatividad son temas de gran interés y ya existen herramientas creadas con la intención de que sean un medio de generación de ideas [10].

2.2. Aplicaciones de la IA en música

En los últimos años, la generación automática de fragmentos musicales [11], la recomendación de canciones [12], la identificación de canciones [13], e incluso la transformación automática de una canción de un estilo a otro de otra canción [14], son algunas de las aplicaciones prácticas que han ganado popularidad [15]. Con el creciente interés en el área, se han desarrollado métodos innovadores, como por ejemplo la clasificación de sonidos de instrumentos musicales utilizando características extraídas de la "Marcha Turca" de Mozart [16].

La aplicación del *Deep Learning* (DL) a casos específicos en el área de la música no es algo nuevo. En 2021, se publicó la Décima Sinfonía de Beethoven, en su día jamás acabada por el propio compositor, pero anunciada como una pieza finalizada por Inteligencia Artificial [17]. La realidad es que el proceso de composición de la pieza fue un trabajo en equipo de musicólogos, historiadores de música e informáticos, partiendo de bocetos escritos por el propio Beethoven y mezclando todo esto con el uso de la IA [18]. Se puede considerar a éste, por lo tanto, un evento multidisciplinar que unió a profesionales de distintas áreas como músicos e investigadores en la exploración de los límites de la IA generativa. Otro ejemplo es la canción de los Juegos Olímpicos de Tokyo 2020, fue creada con inteligencia artificial y a partir de mil muestras de sonido y buscando la participación de los fans [19].

2.2.1. Técnicas y modelos

Las técnicas y modelos empleados han ido cambiando y avanzando rápidamente. Por ejemplo, existen casos del uso de redes neuronales convolucionales (CNNs) en la clasificación de géneros musicales [20].

Otro de los modelos más populares en el área de la música generada por IA son las *Generative Adversarial Networks* (GANs). Ejemplos pueden ser MuseGAN [21] y CycleGAN [22], el primero centrado en generar segmentos musicales y el segundo en transformar un segmento musical preexistente de un estilo a otro [23], [15].

Por otro lado, los transformadores o *transformers* no solo tienen éxito en el procesamiento del lenguaje natural, también en el área de la generación musical [17], [15]. Ejemplos como *Music Transformer* [24] o *Pop Music Transformer* [25] son capaces de generar segmentos musicales de calidad usando *Transformers*.

También existen aplicaciones de las Redes Neuronales Recurrentes (RNNs) y los *Variational Autoencoders* (VAEs) [26] a casos de generación musical [27]. Se han presentado combinaciones resultando en modelos híbridos compuestos por VAEs y RNNs [26], con la intención de encontrar la coherencia y diversidad en las melodías generadas; o las GANs combinadas con *transformers* en la generación de música simbólica [28].

Más allá de si un modelo es mejor o peor en el campo de generación musical, en el área de generar imágenes a partir de texto, las GANs son útiles pero con ciertas carencias a la hora de afrontar imágenes con más detalles [7]. Para afrontar esto los modelos de difusión, o *diffusion models*, son una técnica prometedora, generando el resultado a partir de añadir y eliminar ruido de forma iterativa [7].

Este TFG se centra en explorar este tipo de modelos en la generación musical, una vez transformados los audios de entrada a imágenes.

2.2.2. Soluciones existentes en composición

El crecimiento y rápida evolución no solo en inteligencia artificial sino en cómo aplicar esto a herramientas en distintas áreas está cambiando poco a poco los

procesos creativos tradicionales y generando interés en modelos generativos en música [26]. Esto implica el interés en herramientas accesibles en procesos de composición [15].

No solo existen soluciones basadas en IA, hay muchas otras soluciones previas basadas simplemente en teoría musical. Son menos complejas pero también eficientes, como por ejemplo, *"The Chord Generator"* [29], que a partir de especificar una escala musical genera una progresión de acordes aleatorios dentro de esa escala. Al final, son métodos que mediante la programación de las reglas musicales se pueden usar como sistemas automáticos de composición, pero son sistemas menos flexibles y actualmente el foco se encuentra en modelos de *deep learning* que pueden aprender reglas y patrones pero siendo menos rígidos [15].

Existen bastantes herramientas públicas y accesibles en Internet. Haciendo una simple búsqueda con el objetivo de generar música con IA se pueden encontrar numerosas opciones. Hay una serie de modelos preexistentes que hoy en día son comunes. Por ejemplo, Google Magenta [30], de código abierto y con los *datasets* que se usaron para el entrenamiento del modelo públicos. Existen también otros modelos, como por ejemplo *"Chords Progressions Transformer"* [31], que está entrenado para generar una progresión de acordes. "AccoMontage2" [32] funciona de forma que a una melodía de entrada le genera el acompañamiento completo y armonización entera de la pieza.

Además, no solo es interesante la creación de nuevas herramientas para generar música, sino también investigar aquellas que sí estaban desde un principio enfocadas a componer o a producir música desde cero. Audacity [33] y Ableton Live [34] son muy populares en temas de producción musical, siendo la primera gratuita y la segunda de pago.

Audacity tiene una serie de AI plugins [35] disponibles. Entre otros, *"Music Generation and continuation"* [36] es útil para generar música a partir de un *prompt* de texto o también de continuar una canción preexistente.

Por otro lado, Ableton Live tiene integrado Magenta Studio [37], que se encuentra integrado de manera que es muy accesible y permite experimentar con herramientas de *machine learning* a pesar de no tener conocimientos del tema, y generar nuevas melodías y ritmos.

No se obvia la existencia de muchas otras herramientas de producción musical, con mayor o menor popularidad, pero que en base son muy parecidas unas a las otras. Musescore [38] no solo es una aplicación que sirve para visualizar, modificar y crear partituras, sino también está formada por una comunidad donde se pueden encontrar todas las partituras posibles y cursos sobre música.

2.2.3. Características comunes entre soluciones web

Hay varias webs que buscan ser herramientas útiles en el la música generativa. Un ejemplo es Loudly [39], una herramienta web que se ha sumado a la ola de la IA musical generativa. En general, las herramientas web ofrecen propuestas extremadamente similares a Loudly. Por ejemplo, se ofrece distintas modalidades

desde *text to music* donde se puede poner como *input* un texto, y se generan tres posibles canciones de 30 segundos adecuadas a la propuesta del texto inicial. Además, posee otra opción que se puede combinar con la ya mencionada, en la que se sube un audio de entre 1 a 60 segundos y se selecciona qué es lo que se quiere generar entre una gran variedad de opciones (batería, bajo, teclado, guitarra, etc). El *output* después de esto es el clip inicial con lo generado encima en el mismo audio mezclado. Además, Loudly posee otra modalidad en la que se puede seleccionar el género de la canción a crear y los instrumentos que estarán presentes. Otras opciones bastante interesantes y útiles como la duración de la canción o la selección de los subgéneros ya se vuelven opciones *premium* de pago. Igualmente, existe la opción de que la propia canción incluya la letra. Esta no encuentra específicamente en Loudly pero sí en otras webs o aplicaciones con soluciones integradas de IA musical generativa.

Estas herramientas facilitan el uso a muchos creadores de contenido, por mencionar un caso práctico de uso, que buscan crear música de una manera fácil y rápida, pero que se amolden a lo que tienen en mente y asegurándose evitar problemas con el *copyright*.

2.2.4. Limitaciones soluciones actuales

Sin embargo, estas soluciones web tienen límites. Los más obvios son que enseguida las opciones más interesantes se convierten en *premium*, y que la duración del audio, entre otras características, suele ser limitada. Es el ejemplo de Canva [40], donde el límite son 5 minutos, bastante más que la opción gratuita de Loudly que ronda el minuto.

A pesar de todo, y aún así gracias a herramientas como la mencionada, es posible amoldar la pieza resultante lo máximo posible a lo que se esperaba. Para que la realidad sea lo más cercana a la composición que el creador tenga el mente, hay que conocer cómo se desee que sea, qué sentimientos evoca seleccionando los instrumentos y los géneros que sean de interés. Sin embargo, realmente la pieza la compone el modelo, no el propio usuario. Por tanto, son herramientas muy útiles para la generación pero no parecen adecuadas para un proceso de composición que vaya de la mano y poco a poco acompañando al autor.

2.3. *Datasets*

2.3.1. Formatos y características

El mundo de la música generada por IA es extenso, tanto por la variedad de modelos como por la variedad de posibles datos como entrada. Sin ir más allá, existen sistemas de generación musical monofónicos, polifónicos, *multitrack* o de acompañamiento [17].

Hay varios formatos posibles a la hora de entrenar modelos relacionados con música. Desde *piano rolls* (figura 2.1), como el *Lakh Pianoroll Dataset* [41] usado en el entrenamiento de MuseGAN, mencionado previamente. Este dataset es derivado a partir del Lakh MIDI dataset [42], cuyo formato ya no son *pianorolls*

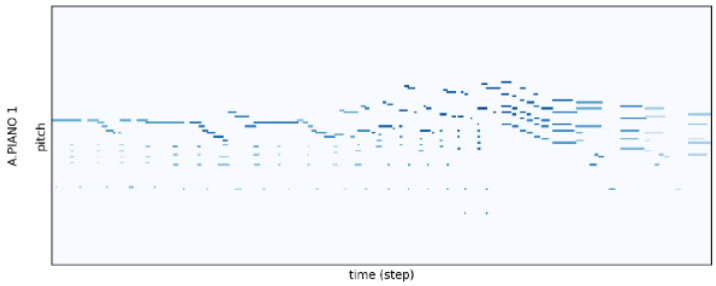


Figura 2.1: Ejemplo de una imagen representativa de *piano roll* extraída del dataset Lakh Pianoroll Dataset [41]

sino archivos MIDI. El formato MIDI codifica toda la información de un audio (tono, velocidad, volumen...), no guarda directamente el audio en crudo [43]. El uso directo de audios en crudo es bastante complicado por la magnitud de información que contienen, y suele ser más recomendable el uso de un formato más sencillo.

Otra manera son las tablaturas (figura 2.2), que es única para cada instrumento porque depende de la colocación física de las manos y dedos del músico que toca el instrumento [43].

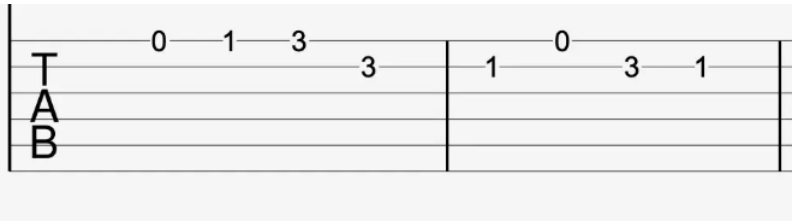


Figura 2.2: Ejemplo de una tablatura para guitarra de la web [44]

También se ha explorado el formato de audio pero en versión imagen, con espectrogramas (figura 2.3) [20]. Este ejemplo es de gran interés por el manejo de datos. En el área de clasificación de géneros musicales por medio de CNNs se usaron los audios transformados a una representación visual espectrogramas mediante la *Short Time Fast Fourier Transformation* (STFT).

Por último, uno de los formatos más obvios quizás, sea simplemente usar una imagen directamente extraída de una partitura.

2.3.2. Análisis de *datasets* relevantes

Existen una serie de *datasets* musicales que recogen horas y horas de piezas, en su mayoría para piano. Un buen ejemplo es MAESTRO [45] que contiene aproximadamente 200 horas de piezas para piano. Otro ejemplo también bastante usado es "The Lakh MIDI Dataset" [42], que es una colección de 176.581 fragmentos musicales únicos. Incluso hay *datasets* generados automáticamente como por ejemplo SunoCaps [46], que es un dataset generado por *prompts* de

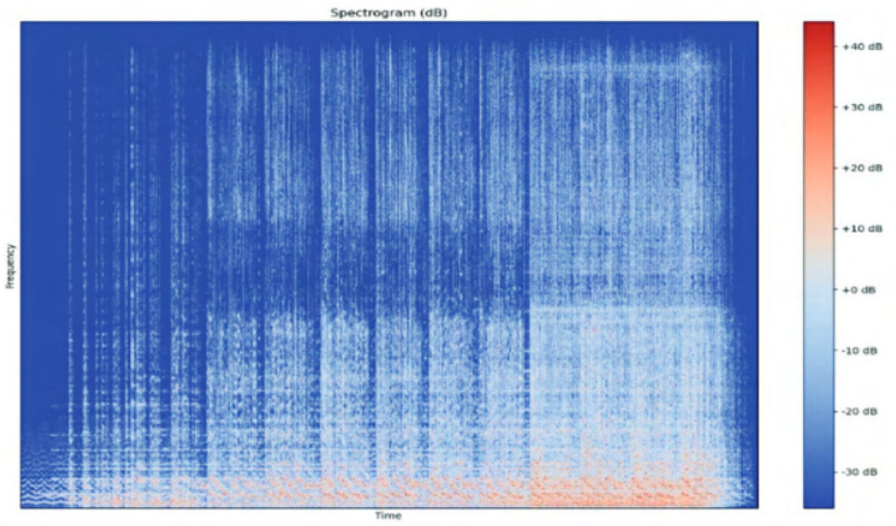


Figura 2.3: Ejemplo de un espectrograma extraído del *paper* [20]

texto usando un generador llamado Suno [47], [48]. Hay también *datasets* que tienen en cuenta temas de armonización, como por ejemplo Bach Chorales Dataset [49], pero no se ha encontrado ninguno que posea a la vez los archivos simples y los "embellecidos". No se busca una versión simplemente armonizada, sino más compleja, con ritmos más complejos y el conjunto final en general más completo.

Capítulo 3

Desarrollo

El desarrollo de este Trabajo Fin de Grado ha implicado un conjunto estructurado de etapas, que, aunque no siempre secuenciales, han contribuido de manera progresiva y complementaria al avance del proyecto. A continuación, se describen las fases principales que han conformado el proceso de trabajo, muchas de las cuales se han solapado parcialmente durante su ejecución.

- Prerrogativa: Búsqueda y selección de un modelo preexistente
- Exploración de *datasets* y construcción de uno propio
- Entrenamiento del modelo con el *dataset* definitivo
- Evaluación y pruebas
- Análisis y diseño de la experiencia de usuario

3.1. Prerrogativa: Búsqueda y selección de un modelo preexistente

Desde el comienzo del TFG, la idea fue trabajar con técnicas del campo de IA generativa con imágenes, transformando los audios musicales a espectrogramas. Existen una gran variedad de proyectos que afrontan el problema de recuperar una imagen cuando las condiciones del clima son adversas. Es decir, imágenes con lluvia, nieve u otras condiciones de clima adverso, son transformadas a imágenes de mejor calidad donde se omiten estas características que empeoran la imagen (ver figura 3.1). En resumen, las imágenes de entrada están "manchadas" y el objetivo es eliminar estas manchas. Este proceso se podría hacer a la inversa, buscando generar imágenes con estas "manchas" de lluvia o nieve a partir de otras imágenes limpias. Traduciéndolo al campo de espectrogramas, el propósito consistía en explorar la posibilidad de obtener nuevos espectrogramas complejos a partir de unos simples, añadiendo "manchas" que aprendiese el modelo, similar al proceso de añadir lluvia a una imagen limpia.

Se exploraron distintos proyectos preexistentes [4], [5], [6], que llevan a cabo la reconstrucción de imágenes en condiciones de tiempo adversas. Desde proyectos

que usan *transformers* y no tienen el código completo público [50] hasta otros, por ejemplo, [51] se centra en mecanismos de aprendizaje con arquitectura "de múltiples profesores y un estudiante".

Figura 3.1: Ejemplo de imágenes en condiciones de clima adverso y sus respectivas imágenes limpias, extraídas del repositorio del proyecto [52]

El proyecto seleccionado es [53]. Esta decisión se debe a su uso de un modelo de difusión generativo basado en parches, una técnica prometedora en el ámbito de la generación y restauración de imágenes, que se posiciona a la vanguardia en éste área. Todo el código del proyecto se encuentra público en *Github* [52], documentado de forma concisa y clara, tanto el entrenamiento como la evaluación. Es un proyecto que expone visualmente los resultados (figura 3.2), y este factor fue clave en su momento para pensar que se podría aplicar exitosamente a un caso completamente distinto como es el uso de espectrogramas.

El modelo de difusión, como se ha mencionado, es un modelo generativo cuyo aprendizaje se basa en un proceso iterativo en el que se introduce ruido gaussiano a las imágenes y después se trata de revertir el proceso hasta que la distribución de este ruido coincida con una normal estándar. Además, este proyecto [52], aborda un problema que presentaban este tipo de modelos. El problema consistía en las restricciones existentes respecto a los tamaños de las imágenes, ya que en los problemas reales, las imágenes suelen ser de distintos tamaños. Básicamente, para resolver estas restricciones, usan "parches", partes pequeñas de la imagen, permitiendo que el modelo trabaje con imágenes de cualquier tamaño simplemente combinando las partes resultantes. A parte de esto, también se emplea entrenamiento condicional, donde se proporciona la imagen de entrada con condiciones de tiempo adversas y la imagen limpia esperada.

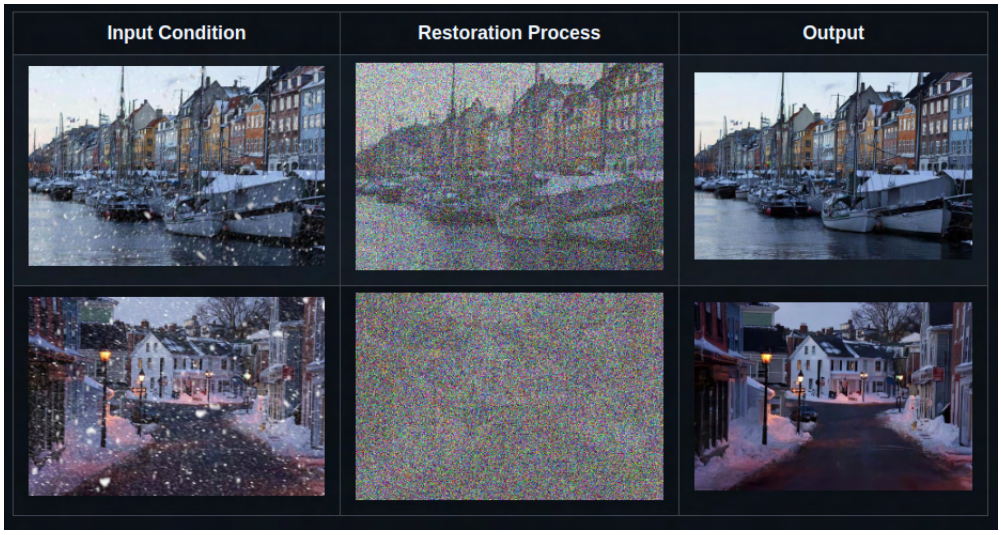


Figura 3.2: Ejemplo de imágenes de entrada, proceso de restauración y salida final del modelo para la eliminación de nieve, extraídas del repositorio del proyecto [52]

Entonces, el proceso es el siguiente. Las imágenes se subdividen en parches superpuestos de tamaños 64x64 o 128x128. Se usa una red *U-net* para estimar el ruido de cada parche. La red *U-net* tiene una arquitectura basada en *WideResNet* [53] que utiliza normalización de grupos y mecanismos de atención, procesando los datos de forma más estable y aumentando el enfoque en las partes importantes de la imagen.

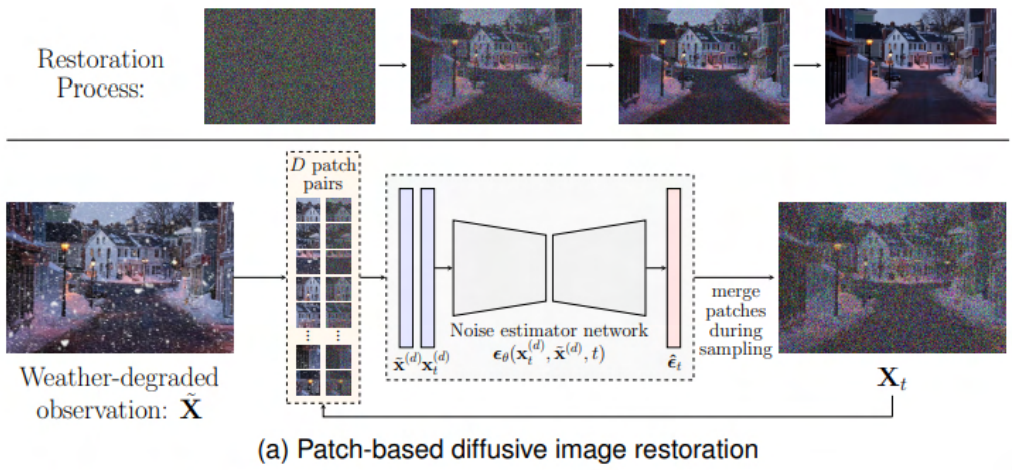


Figura 3.3: Imagen extraída del *paper* [53] que representa el proceso de restauración de imágenes con modelo de difusión basado en parches

Para llevar a cabo el entrenamiento condicionado mencionado previamente, lo que se hace es unir la imagen del parche ruidoso y la imagen del parche limpio, resultando en una imagen de seis canales de color. A partir de esto, se predice

el ruido y se toma la imagen ruidosa para restarle el ruido predicho; así, hasta que la imagen esté limpia. Una vez hecho esto por cada parche se calcula un promedio del ruido predicho entre los parches con bordes coincidentes. La figura 3.3 extraída del repositorio del proyecto [52] representa este proceso.

3.2. Construcción del *dataset*

Este punto está dedicado a explicar la construcción del *dataset*. Tanto la construcción del *dataset* como el entrenamiento, son dos etapas que se han ido desarrollando de manera entrelazada. La construcción del *dataset* definitivo ha sido un proceso iterativo, donde una vez decidido el tipo de datos, se llevaron a cabo varias pruebas para decidir el formato de las imágenes finales. En un primer momento, las imágenes serían espectrogramas, pero entonces surgen problemas a la hora de recuperar el audio a partir de solo esas imágenes y el proyecto se bifurca en dos caminos. Por un lado, se explora el uso de imágenes de partituras y paralelamente, se sigue intentando que el formato final sean espectrogramas. Entonces, el proceso finaliza con dos *datasets* definitivos, uno de imágenes de partituras y otro de espectrogramas; y se entrena un modelo con ambos aunque la herramienta final se centra en el basado en espectrogramas. La figura 3.4 esquematiza las distintas etapas del proceso.

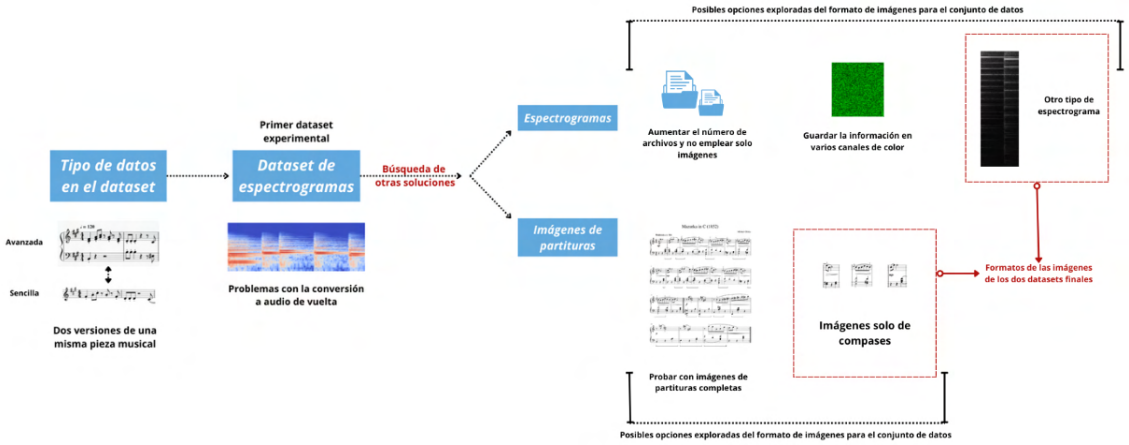


Figura 3.4: Diagrama del trabajo desarrollado

3.2.1. Tipos de datos a usar en el TFG

Entonces, el problema planteado en este TFG requiere clarificar cuando una pieza musical es simple y cuándo es más completa. Una posible manera de hacerlo es recopilar de una misma pieza musical diferentes versiones siguiendo lo mencionado previamente; una versión para un músico principiante y una versión que se considera de un nivel avanzado, completa y mucho más rica. Ambas versiones provienen de la misma pieza.

Para ello, la herramienta de *Musescore* [38] es muy útil, ya que se pueden buscar varias versiones de una misma pieza comparando directamente las partituras, y

asegurándonos de que los únicos cambios entre una y otra sean estos ajustes. Este paso puede parecer irrelevante pero es crucial. También es posible editar las partituras recopiladas, por lo que si no se encuentran dos versiones de la misma pieza que encajen, otra opción es editarlas manualmente.

Un ejemplo sencillo, trasponiendo el problema al ámbito del lenguaje para que sea más comprensible, es que sería como que el *dataset* se compusiese de oraciones simples y las mismas oraciones pero añadiendo más detalles semánticos y expresivos. Lo ideal sería que ambas estuvieran en el mismo idioma, ya que si estuviesen en idiomas distintos, sería más tedioso encontrar los patrones que embellecen y completan la oración. Lo mismo ocurre en música. Si existen dos versiones de una misma pieza hay que asegurarse que estén en la misma escala, que el ritmo sea el mismo o uno equivalente, o que ambas estén pensadas para el mismo instrumento para poder ser comparadas.

En las siguientes figuras se muestran fragmentos de partituras obtenidas en *Musescore* para una composición sencilla, uno de los temas musicales de la consola Wii de Nintendo [54]. La figura 3.5 es la versión más sencilla, y la figura 3.6 es la misma versión pero más completa y pensada para músicos que tengan más nivel y lleven más tiempo tocando. Ambas están compuestas para piano y si se comprobase nota por nota, o compás por compás, se puede asegurar que la base es la misma, añadiendo ciertos componentes que lo van haciendo sonar más rico y la completan.

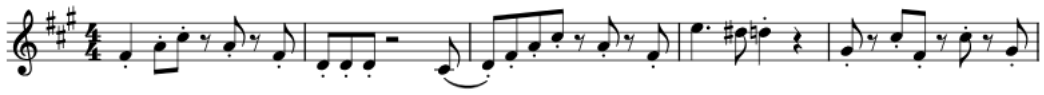


Figura 3.5: Fragmento de partitura de una canción de la Wii obtenido en Musescore

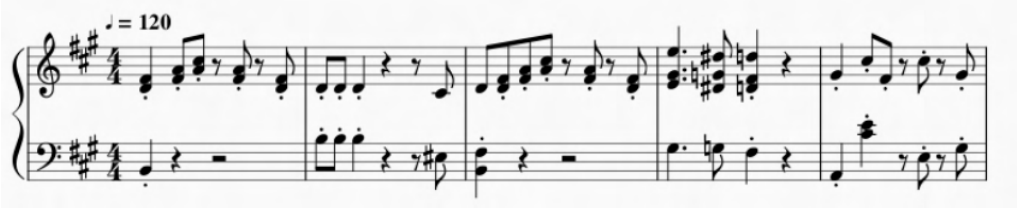


Figura 3.6: Fragmento de partitura de una canción de la Wii obtenido en Musescore

Aunque no se posean conocimientos en música, se puede ver cómo la figura 3.5 y 3.6 son similares, mientras que en la siguiente (figura 3.7) se puede observar a simple vista que es distinta. Es una versión de la misma pieza que las anteriores, pero en este caso, por ejemplo, está pensada para flauta travesera, que tiene un rango distinto y más limitado que el del piano.

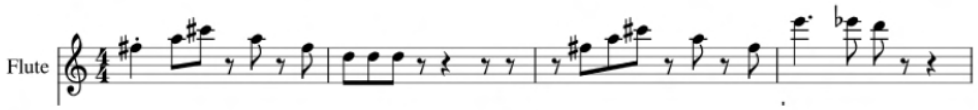


Figura 3.7: Fragmento de partitura de una canción de la Wii obtenido en *Musescore*, para flauta travesera

3.2.2. Creación del primer *dataset* experimental

Primero se construyó un *dataset* de muestra, simplemente para comprobar el funcionamiento de este modelo y su aplicación práctica a este caso. Un experimento con el que ver qué tipo de imágenes se podían obtener y si se acercaban a lo que se buscaba. Este proceso fue útil para experimentar con métodos para la recopilación de datos y explorar posibles formatos y plataformas, como se ha mencionado antes, *Musescore*.

La estructura del *dataset* sigue la empleada en el proyecto de *Weather Diffusion* [52] (el repositorio seleccionado), para su aplicación directa a éste. Básicamente, se divide en dos conjuntos, uno de entrenamiento y otro de validación. Cada conjunto está compuesto por una carpeta que contiene versiones sencillas de las piezas y otra carpeta donde se encuentran los pares complejos, como la salida deseada respectivamente. El entrenamiento es supervisado porque se utilizan pares de espectrogramas de entrada y salida conocidos. En resumen, en el entrenamiento se emplean pares de "pieza sencilla" y "pieza avanzada", donde el modelo aprende a transformar las primeras a las segundas. A partir de las primeras pruebas con el primer *dataset* experimental, se fue aumentando y refinando; ya que al principio, era reducido y estaba orientado a ser un borrador con el que se experimentar con el proyecto preexistente.

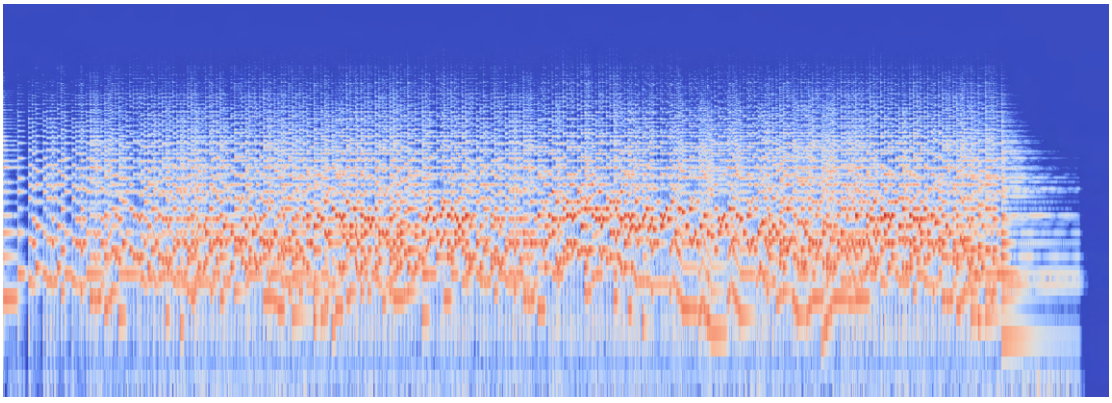


Figura 3.8: Espectrograma completo de "*Fugue* Nº7 de Bach"

El primer reto fue transformar los audios en espectrogramas. Se siguió un tutorial [55] y se usó la librería de *Python* "librosa" [56] que proporciona tanto la *Short-Time Fourier Transform* (STFT) para generar el espectrograma (figura 2.3), como el algoritmo de Griffin-Lim para la transformación de espectrograma a au-

dio de nuevo. En la transformación de audio a espectrograma inicial, se usaron los parámetros por defecto del método proporcionado por *librosa*:*n_fft* con valor de 2048 y *hop_length* con valor de 512. *N_fft* es el número de muestras sobre el que se calcula de la *Fast Fourier Transform* (FFT) [57], un algoritmo para convertir señales del dominio del tiempo al de frecuencia. Cada componente convertido es una onda sinusoidal que cuenta con su propia amplitud y fase [57]. *Hop length* es la longitud del salto, que determina cuántas muestras de audio se avanzan entre ventanas FFT seguidas e influye directamente en la resolución temporal del espectrograma. En este caso, se elige que sea menor al número de muestras para que se produzca solapamiento y haya una alta resolución temporal. Una vez realizadas estas transformaciones, se calcula la magnitud de las frecuencias, tomando el valor absoluto del resultado previo y perdiendo la información de fase. Entonces, se transforma a escala logarítmica, expresándolo en decibelios, lo que se alinea con cómo el oído humano percibe el sonido.

El primer *dataset* experimental estaba formado por dos piezas. Se hizo tan pequeño para que fuera una especie de tanteo en el proceso, mientras que a la par se exploraba el proyecto existente, su *paper* y su repositorio de código. Cada audio se segmentó cada dos segundos (figura 3.9), ya que esto facilita el trabajo a la hora de la construcción de un *dataset*, por la cantidad de información necesaria. Si cada pieza fuera tan solo un espectrograma (figura 3.8), la recopilación de datos habría sido prácticamente imposible. Sin embargo, sabiendo que las sugerencias deseadas son breves, no se necesita el audio completo. Se puede dividir una pieza en trozos más pequeños, aumentando la cantidad de información de forma considerable. Es más, si a la hora de evaluar son audios de más de un segundo, simplemente se dividen de la misma manera y tras ser transformados por el modelo, se pueden unir de nuevo en un audio completo, como se verá más adelante.

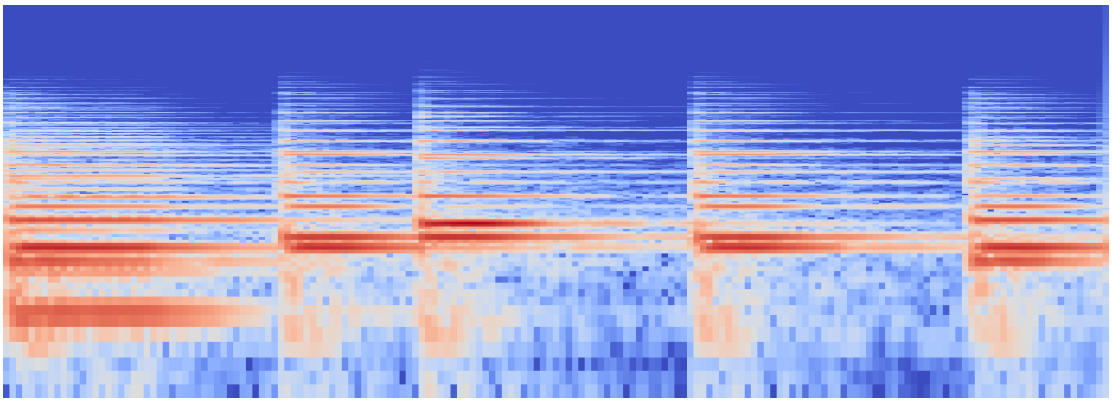


Figura 3.9: Espectrograma de un segmento de dos segundos del audio original

3.2.3. Transformación de los espectrogramas de nuevo a audios

Se realizaron unos primeros entrenamientos experimentales para ir ajustando los parámetros del modelo. Estarán explicados en la sección de entrenamiento (3.3.1). Una vez entrenado una primera versión de modelo, la idea era evaluarlo

y transformar los espectrogramas resultantes a audios de nuevo. La figura 3.10 representa las distintas etapas implicadas en el uso del modelo entrenado. Como punto de partida, se toma un audio de una versión sencilla de una pieza que se quiera transformar en una sugerencia más compleja y se divide en segmentos de un segundo, resultando en *n* segmentos. Cada segmento individual se transforma en un espectrograma y se evalúa con el modelo previamente entrenado. La salida por cada espectrograma devuelta por el modelo se transforma en audios y son fusionados en un único audio resultante. El objetivo final es que el audio inicial de entrada sea transformado por el modelo en un audio en base similar pero con sugerencias y más completo.

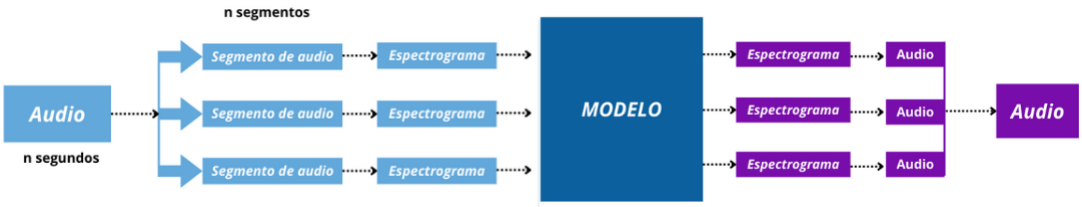


Figura 3.10: Esquema que representa las etapas del flujo de trabajo

Así que, en la primera prueba del flujo de trabajo, una vez se había probado el entrenamiento se probó el proceso inverso, la transformación de espectrogramas a audios. La técnica más común para este proceso es el uso del algoritmo de *Griffin-Lim* [58]. Este algoritmo toma como entrada la magnitud de un espectrograma y de forma iterativa refina su fase con un proceso de "ida y vuelta" aplicando la *short time fourier transform* (STFT), lo que permite la reconstrucción del audio [59].

Se aplicó este algoritmo integrado en la propia librería de *librosa*, mencionada previamente. Las primeras pruebas se hicieron con espectrogramas de los datos de entrenamiento, asegurando que el audio de salida fuera conocido para poder evaluar el funcionamiento del código de manera apropiada y más tarde, ejecutarlo con espectrogramas generados por el modelo y comprobar si realmente eran lo suficientemente buenos. Sin embargo, tras realizar múltiples pruebas, los resultados en la transformación de espectrogramas de vuelta a audios, no fueron positivos. Tal y como se estaban preparando y guardando los espectrogramas, esta transformación era posible si todo ocurría en una misma ejecución y se conocía toda la información del audio original. Si el punto de partida era un espectrograma que se cargaba como imagen directamente y sin más información se trataba de recuperar el audio original, no funcionaba.

Entonces, la raíz del problema de revertir estas transformaciones era precisamente cómo se estaban procesando y guardando estas imágenes de espectrogramas y sus transformaciones aplicadas para su visualización.

Cuando se utiliza la STFT, se obtiene un espectrograma complejo, que contiene tanto la magnitud como la fase. Sin embargo, para visualizar el espectrograma, se calcula el valor absoluto (magnitud) y se descarta la fase. Posteriormente, esta magnitud suele transformarse, como en el caso de *librosa*, donde se convierte a una escala logarítmica de decibelios. Al guardar esta imagen en un formato como

jpg o *png*, los datos de coma flotante se transforman a valores enteros. Entonces, las imágenes guardadas no contenían por si solas toda la información necesaria para la recuperación del audio y funcionaba exclusivamente cuando se realizaba en una misma ejecución porque estos datos estaban presentes en variables en memoria. A la hora de seguir el flujo de trabajo planeado e intentar cargar de nuevo la imagen transformada, ya se había perdido demasiada precisión y el algoritmo de *Griffin-Lim*, aunque capaz de estimar la fase perdida, no bastaba para compensar los datos numéricos alterados significativamente tras varias transformaciones. En resumen, no se alcanzaba una reconstrucción del audio original.

El nuevo objetivo entonces, fue encontrar otra forma alternativa de transformar los audios a espectrogramas de un modo que no perdieran información importante y que reconstruirlos a partir de solo imágenes fuera una opción.

Aumentar el número de archivos

La primera manera de afrontar este problema fue reflexionar si era posible simplemente guardar esta información en archivos aparte, ya que así, sería sencillo obtener los audios resultantes más fieles a la realidad en esta transformación. Esto no es viable en el TFG porque el objetivo desde un principio era trabajar con técnicas aplicadas a imágenes y añadir un par de archivos con la información de los audios no es compatible con este propósito.

Guardar la información valiosa en varios canales

En segundo lugar, tras analizar el problema, se pensó que quizás sería posible almacenar toda la información necesaria en distintos canales de color. Esta idea surgió de un par de prácticas similares llevadas a cabo durante el curso pasado en la optativa de Fotografía computacional. Esto era compatible con las transformaciones y los audios resultantes eran adecuados, pero las imágenes eran demasiado complicadas y distinguir entre una y otra para ver qué elementos cambiaban entre las parejas de imágenes era casi imposible. La figura 3.11 representa el tipo de imagen que se estaba obteniendo. Esto es un problema, ya que para entrenar el modelo se buscaba trabajar con imágenes que tuvieran contrastes más altos y que si se agruparan por parejas fuera posible ver las diferencias que hay de una a otra.

La figura 3.12 representa dos imágenes que se han extraído de una prueba de entrenamiento con este tipo de datos, que acabó descartándose porque son imágenes muy poco distintivas y los resultados no fueron buenos.

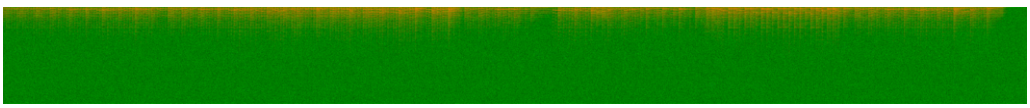


Figura 3.11: Imagen de una pieza con la información almacenada en varios canales de color.

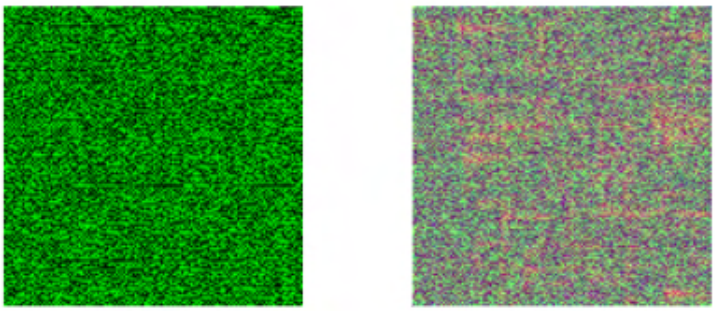


Figura 3.12: A la izquierda, *patch* de entrenamiento de referencia y a la derecha, *patch* de entrenamiento generado por el modelo

Imágenes de partituras

En el proceso de buscar una solución viable para poder transformar los audios a espectrogramas y viceversa, se pensó que otra opción de formato de imágenes del *dataset* más directa podría ser imágenes de partituras.

Hay varias maneras de construir un *dataset* de partituras, ya que podrían ser imágenes completas de toda la pieza, imágenes que contuvieran solo una línea de pentagrama por imagen o incluso imágenes por compás (figura 3.13).

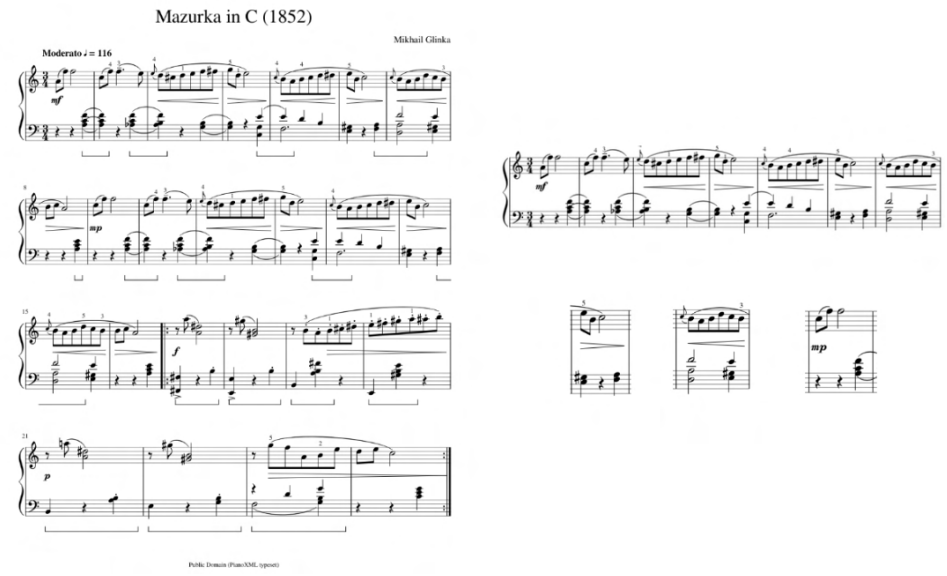


Figura 3.13: Ejemplo de imágenes representando distintos posibles formas de agrupar este formato de imágenes

Se entrenó un modelo con imágenes de partituras completas, provocando la bifurcación del proyecto en dos caminos, los espectrogramas y las imágenes de partituras. Tras esta primera toma de contacto, se decidió entrenar uno de los modelos finales con imágenes de fragmentos de partituras. En la sección (3.3) se explica cómo fue el entrenamiento y cuáles fueron los resultados.

Espectrogramas finales

Se encontró un repositorio [60] que implementa el algoritmo *Griffin-Lim* y posee funciones muy útiles a parte del algoritmo para realizar este tipo de transformaciones. La solución final está basada en este repositorio y se usa su archivo de *audio utilities*. Básicamente, en el proyecto del repositorio se aplican las transformaciones hasta alcanzar un espectrograma de magnitud. Una vez hecho esto, se guarda la imagen del espectrograma y después, se añaden una serie de pasos para la transformación inversa (de espectrograma a audio). Sin embargo, no se aplica directamente con la imagen como entrada, sino de los datos previamente guardados durante la primera transformación de audio a espectrograma, que se encuentran variables en memoria.

Entonces, en la solución final de este TFG, se decidió mantener la implementación de la transformación de audio a espectrograma tal y como está en el repositorio y luego se implementó una forma distinta de guardar los espectrogramas, adaptada a las necesidades propias. El proceso completo es el siguiente. Primero se carga el audio y se le aplica la transformación de STFT para obtener el espectrograma complejo. Se calcula la magnitud al cuadrado, descartando la información de fase pero manteniendo la magnitud en una escala lineal. Esta magnitud lineal se normaliza y se le aplica una potencia, elevándola a 0.125, una operación que es reversible. El resultado se escala a un rango de 0-255 y se guarda como una imagen que almacena los datos ya transformados, guardando los datos numéricos en píxeles en escala de grises.

$$S(x,y) = \left(\frac{|\mathrm{STFT}(a)|^2}{\max\limits_{x,y}\left(|\mathrm{STFT}(a)|^2\right)} \right)^{\frac{1}{8}} \cdot 255 \tag{3.1}$$

donde a es la señal de audio, $\mathrm{STFT}(a)$ es la transformada corta de Fourier del audio, y $S(x,y)$ representa el valor del píxel en la imagen de salida en una escala de grises.

De esta manera, se tiene de forma muy clara y paso por paso las transformaciones que se han ido realizando al audio de entrada y resulta mucho más sencillo invertir las transformaciones una por una y aproximarse lo máximo posible a los datos iniciales, recuperando una estimación de la magnitud original en su escala lineal con una fidelidad suficiente. Esta magnitud proporciona los datos de entrada óptimos para que el algoritmo de *Griffin-Lim* (empleando directamente el del repositorio) pueda estimar la fase perdida de manera efectiva, logrando así una reconstrucción del audio significativamente más fiel a la señal original.

El resultado son espectrogramas menos "vistosos" y coloridos, pero siguen siendo distintivos entre ellos, son sencillos y lo más importante, es posible su transformación inversa a audio de nuevo.

3.2.4. Construcción del *dataset* definitivo

La construcción del *dataset* se hizo usando la herramienta *Musescore*, la cual cuenta con una comunidad muy extensa de músicos, donde se publican partitu-

ras de piezas de estilos de gran variedad. Para ello, se obtuvo la versión *premium*, para no tener restricciones a la hora de descargar piezas.

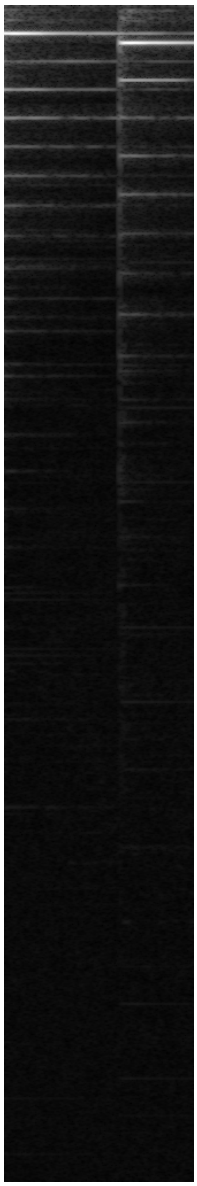


Figura 3.14: Ejemplo de espectrograma definitivo

En un primer momento, se pensó en crear un *dataset* extenso formado por bandas sonoras de películas y música de interés personal. Sin embargo, aunque existan partituras en esta plataforma bajo licencias permisibles, las piezas no lo son y se decidió por ética respetar a los autores de las obras y usar en su lugar piezas clásicas de dominio público. Entonces, todas las piezas elegidas pertenecen a las eras musicales del Barroco, Clasicismo, Romanticismo o Impresionismo (ver figura 3.17), y de dominio público; y las partituras tienen licencias de dominio público o de libre de uso bajo atribución. Se incluirá un *excel* con las piezas escogidas y sus *links* de origen en el repositorio.

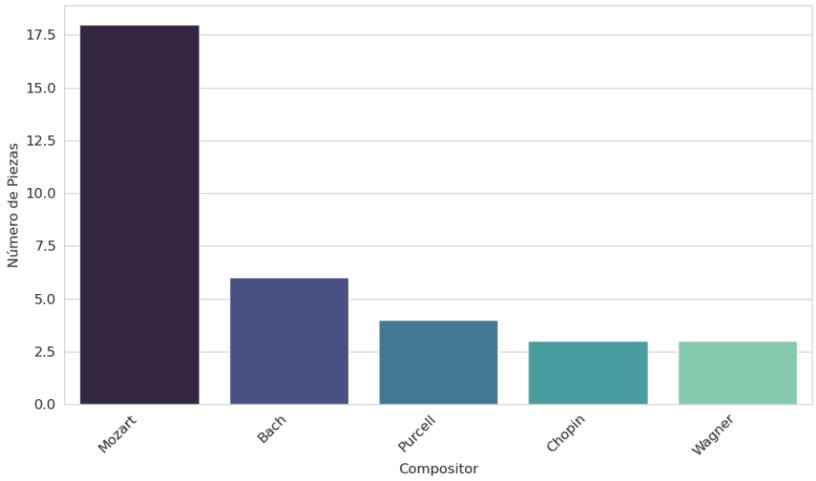


Figura 3.15: Top 5 compositores más comunes en el *dataset*

A la hora de seleccionar las piezas se tuvieron varias cosas claras:

- Debían ser piezas orientadas a piano.
- Cada pieza tiene dos claves (clave de sol y de fa), esto es lo común para las piezas de piano, donde los pianistas interpretan cada pentagrama con una mano (figura 3.13).
- Debían ser versiones complejas o más "avanzadas" para que pudieran simplificarse.
- Búsqueda de piezas clásicas de dominio público, compositores como por ejemplo Bach o Beethoven. (La figura 3.15 representa los 5 compositores más repetidos en el *dataset*.)

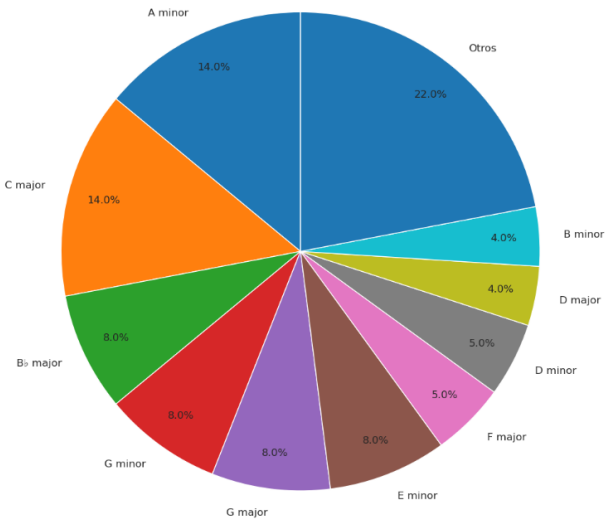


Figura 3.16: Distribución de las tonalidades de las piezas en el *dataset*

Cuando las piezas fueron seleccionadas, se fue guardando un historial de las piezas, compositores, *links* de origen, duración, tonalidades (la figura 3.16 representa la distribución de tonalidades en las piezas del *dataset*) y número de compases. Una vez seleccionadas, se tenía solamente la versión "avanzada" de cada una de ellas. En ese momento, se modificaron individualmente para obtener una versión simplificada de cada una de ellas. Esto se hizo usando aún el propio *Musescore* y guardando esas nuevas versiones. En este proceso de simplificación, se hizo lo siguiente:

- Simplificar ritmos complejos y sustituirlos por otros más simples.
- Reducir el volumen de figuras.
- Borrar adornos como por ejemplo, los trinos o mordentes (figura 3.18)
- Dejar partes solo interpretadas por la mano derecha o izquierda como existe en ejercicios para principiantes donde se toca por separado una u otra.
- Reducir indicaciones de expresividad, como ligaduras, puntillos; de intensidad, como *forte*, *piano*...

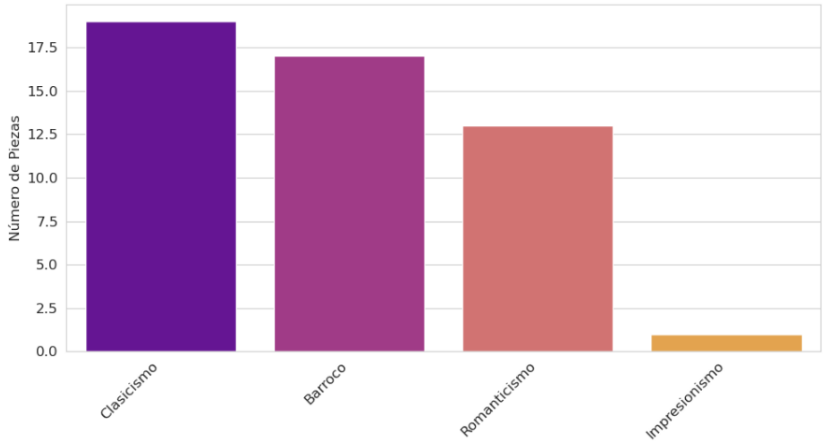


Figura 3.17: Eras musicales de los compositores de las piezas del *dataset*

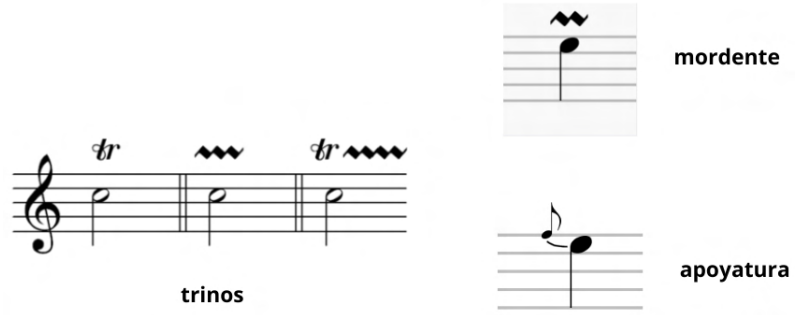


Figura 3.18: Adornos musicales como mordente [61], trino [62] o apoyaturas [63]

3.2.5. *Datasets* resultantes

Una vez se seleccionaron todas las piezas, y se tenían todas sus versiones "avanzadas" y "sencillas", se procesaron los datos y se obtuvieron dos *datasets* resultantes. El primero y principal sobre el que está construida la herramienta de composición es de espectrogramas. El segundo, es más pequeño y está compuesto por imágenes de las partituras (divididas en compases) y preparado con el objetivo de la exploración de otras opciones.

El *dataset* de espectrogramas consta de 8346 parejas de imágenes. Cada pareja es un espectrograma que representa un segmento de un segundo de una de las piezas que componen el conjunto de datos; es decir, una de "avanzada" y otra de "principiante". Está formado por dos carpetas, una de entrenamiento y otra de validación. Se dividió el total de 8346 imágenes en un porcentaje de 80 %-20 %, De manera que el conjunto de entrenamiento son 6678 pares de imágenes (principiante-avanzada) y el conjunto de validación son 1668 imágenes.

De igual manera, se distribuye el otro *dataset* de partituras que consta de un total de 3635 pares de imágenes de compases de partituras. El conjunto de entrenamiento en este caso consta de 2908 pares de imágenes y el conjunto de validación consta de 727 pares de imágenes. En el caso de las partituras, las partituras completas de cada pieza se dividieron por compás, por lo que se reduce significativamente el número de datos. Se dividió automáticamente con *Musescore*, para que todas las imágenes tuvieran el mismo formato (figura 3.19).

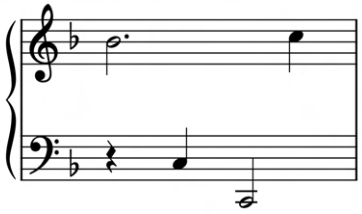


Figura 3.19: Imagen del *dataset* de partituras, como ejemplo de compás

Preprocesamiento de datos

En el caso de los espectrogramas, cada audio correspondiente a una pieza es procesado para formar parte del *dataset* resultante. El audio es dividido en segmentos de un segundo que posteriormente son transformados individualmente a espectrogramas (figura 3.14) y guardados en formato de imagen. Los audios tienen su versión correspondiente en la carpeta de "avanzado" y de "principiante" y se guardan como parejas, cada uno en su directorio correspondiente. El código de transformación de audios a espectrogramas y viceversa está disponible en el repositorio y se emplea tanto en la construcción del conjunto de datos de entrenamiento como en la evaluación.

Por otro lado, respecto a las imágenes de partituras, son directamente obtenidas en *Musescore*. Cada pieza es preprocesada con el objetivo de eliminar toda la información innecesaria para el entrenamiento del modelo, como el título de la pieza y el número de compás (figura 3.20).

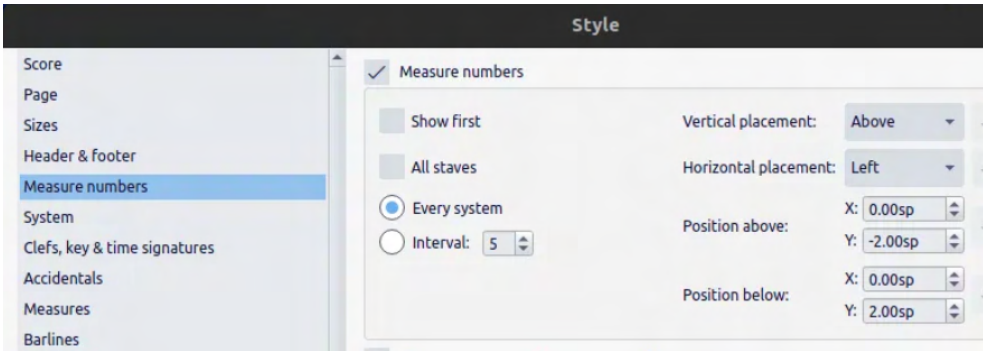


Figura 3.20: Opción de *Musescore* para eliminar el número de compás

Se ajusta cada partitura para que se muestre un compás por línea de pentagrama (figura 3.21) y después se ajusta el tamaño de página a A7 para que solo quepa un compás por página (figura 3.22). Una vez hecho esto, se exporta en formato imagen.

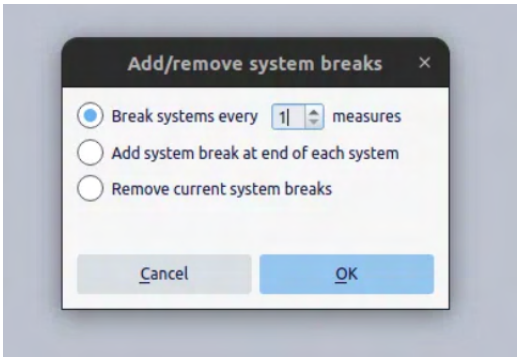


Figura 3.21: Opción de *Musescore* para ajustar un compás por pentagrama

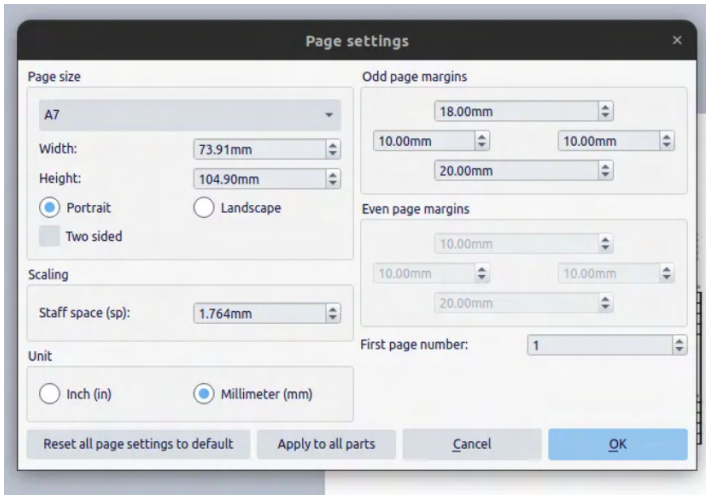
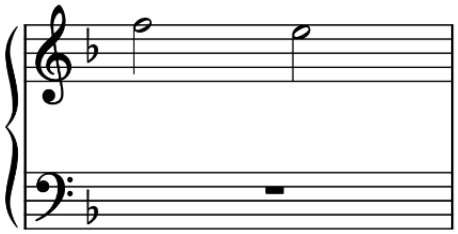


Figura 3.22: Opción de *Musescore* para ajustar el tamaño por página

Por último, con un *script* sencillo se recortaron las imágenes para omitir textos como el de la figura y números de páginas (figura 3.23).



2 Public Domain (PianoXML typeset)

Figura 3.23: Ejemplo de imagen resultante del proceso con texto en la parte de abajo

3.3. Entrenamientos

3.3.1. Primeros entrenamientos con *datasets* experimentales

Espectrogramas

Los primeros entrenamientos sirvieron para ir ajustando los parámetros de configuración para el entrenamiento. Estos parámetros, están inspirados en las configuraciones previas que ofrece el repositorio con varios cambios. Todas las adaptaciones a estos parámetros se hicieron buscando un equilibrio entre rendimiento y calidad en el entrenamiento, teniendo en cuenta la capacidad del equipo. Durante el entrenamiento se emplearon dos entornos diferentes: uno con una GPU (mig-7g.40gb) muy potente, pero donde existían problemas de conexión por sobrecarga y el uso concurrente de la misma máquina por parte de múltiples usuarios; y otro entorno más estable con una GPU de 24GB de VRAM, 16 núcleos de CPU y 32GB de RAM. El entrenamiento osciló de uno al otro, aprovechando la mayor potencia del primero cuando era estable y recurriendo al segundo para asegurar la progresión del proceso cuando el primero no estaba disponible.

Se adaptaron varios aspectos, como la resolución de la imagen. Comenzó como 64x64 píxeles para optimizar el rendimiento computacional y acelerar la fase inicial de experimentación y más adelante se hicieron pruebas con imágenes de resolución 128x128 con el objetivo de comprobar si mejoraban los resultados. Respecto a la carga de datos, se ajustó el número de trabajadores (*num workers*), para optimizar la velocidad con la que los datos se cargan durante el entrenamiento, de 32 a 4 en las primeras pruebas y más tarde a 16.

En la arquitectura del modelo, el parámetro canales base (ch), que establece la 'anchura' inicial de la red neuronal, se redujo de 128 a 64 con el objetivo crear un modelo más ligero y eficiente en recursos. El número de bloques residuales consecutivos en cada nivel de la red *U-net*, fundamentales para que la red aprenda transformaciones complejas manteniendo un flujo de información constante, se ajustó a uno por nivel. Esto simplificó significativamente la arquitectura del modelo, haciéndola más ligera y además, acelerando el entrenamiento y reduciendo el consumo de memoria. En cuanto a los mecanismos de atención que presenta la red *U-net*, se aplicó en la resolución de 16x16, siguiendo la configuración estándar del repositorio.

El número de pasos de difusión se redujo a 500 pasos en los primeros entrenamientos de prueba para acelerar el entrenamiento y en los definitivos se duplicó al estándar usado en este proyecto, 1000 pasos. La tasa de aprendizaje (lr) se estableció en 0.0001, para promover una convergencia más rápida del modelo. El tamaño de *batch* se estableció a 10 y el tamaño del parche durante el entrenamiento se estableció a 16x16 píxeles.

En conjunto, estos fueron los ajustes a los parámetros, permitiendo el entrenamiento de estos modelos de difusión complejos, con el objetivo de que fuera más manejable y eficiente en el entorno con los recursos presentes y se pudiese experimentar con su entrenamiento.

La experimentación con el primer *dataset* se hizo mientras paralalelamente se trataba de transformar los espectrogramas de vuelta a audio (apartado 3.2.3). Las figuras 3.24 se han extraído de los resultados de evaluación del modelo que estaba siendo entrenado con el primer *dataset* experimental, cuyo tipo de espectrogramas eran del tipo de la figura 3.9.

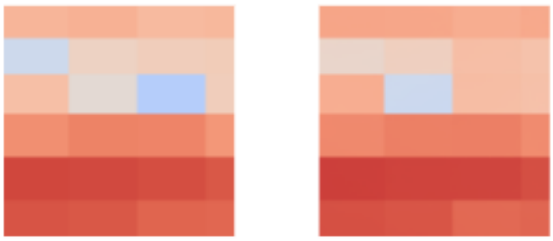


Figura 3.24: A la izquierda *patch* de entrenamiento de referencia y a la derecha *patch* generado por el modelo

3.3.2. Prueba de entrenamiento con partituras

Se realizó un entrenamiento de prueba con imágenes de partituras completas. En el proceso de preparación de imágenes, se notó que al ser partituras completas en vez de solo alguna parte de las partituras, los datos recopilados se reducían de forma creciente en número. Igualmente, se entrenó brevemente, consciente de que los datos proporcionados eran limitantes por cantidad, con el objetivo de experimentar.

Una vez entrenado un modelo con los parámetros fijados previamente se evaluó. La figura 3.25 es un ejemplo de uno de los resultados de la evaluación del modelo. No es un buen resultado porque no es la solución directa al problema, pero, hay figuras musicales que se pueden apreciar de igual manera aunque de forma caótica. Entonces, tras este entrenamiento breve con datos reducidos, se decidió usar imágenes de compases en vez de las partituras completas y probar de nuevo a entrenar el modelo de manera más seria. La mejora de los datos y ampliación a una cantidad decente podría mejorar los resultados. Esto se verá más adelante, en el entrenamiento con el *dataset* definitivo de fragmentos de partituras.

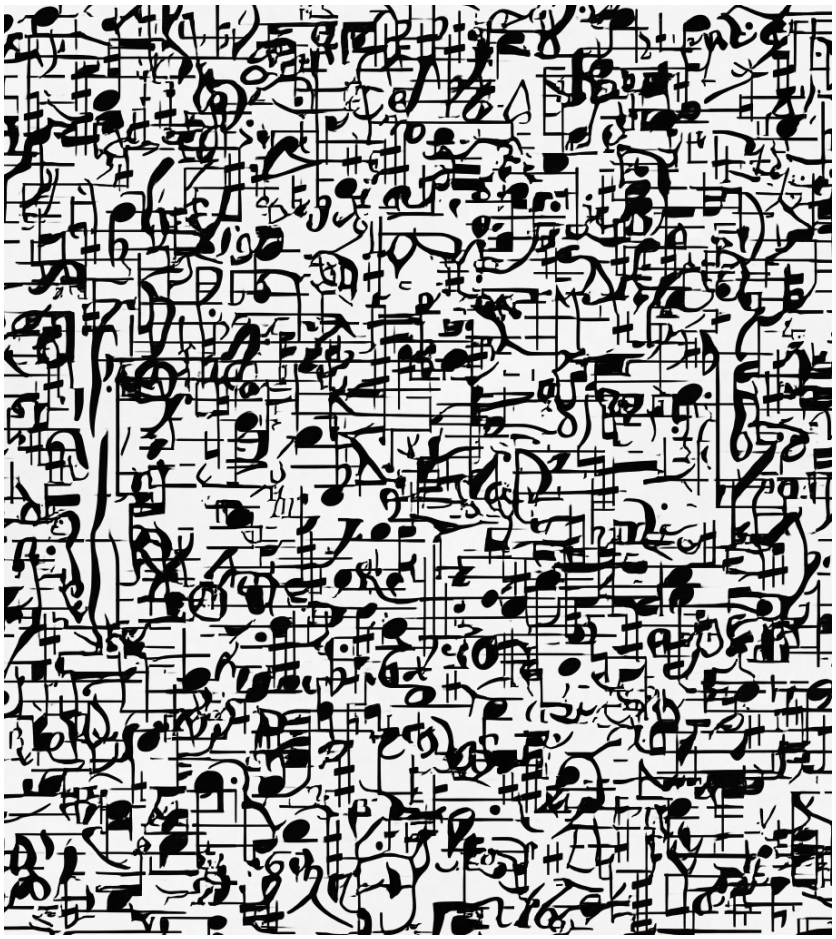


Figura 3.25: Ejemplo de uno de los resultados de la evaluación del modelo

3.3.3. Entrenamiento de los modelos definitivos

Tras los entrenamientos de prueba iniciales (cuyos detalles están en la sección 3.2.3), para los entrenamientos definitivos se establecieron una serie de parámetros que se mantuvieron firmes con ambos *datasets*. Una de las decisiones más significativas fue la reducción de la resolución de imágenes de 128x128 a 64x64 píxeles, con el objetivo de optimizar el rendimiento y reducir la carga computacional.

Por otro lado, varios parámetros mencionados previamente se mantuvieron constantes. El número de pasos de difusión se duplicó a 1000 y la atención se aplica a la misma resolución, lo cual es estándar.

Entrenamiento con fragmentos de partituras

Se entrenó un modelo con el *dataset* definitivo de partituras enfocado a compás por imagen. Los resultados fueron mejores de lo que se esperaban, considerando que el conjunto de datos no era excesivamente grande. Aún así, seguían siendo imágenes "ruidosas" y aún no se podría extraer mucha información como sugerencias de ellas.

La figura 3.26 representa la evolución visual de la generación de parches a lo largo de las iteraciones de entrenamiento del modelo de difusión y como va mejorando poco a poco.

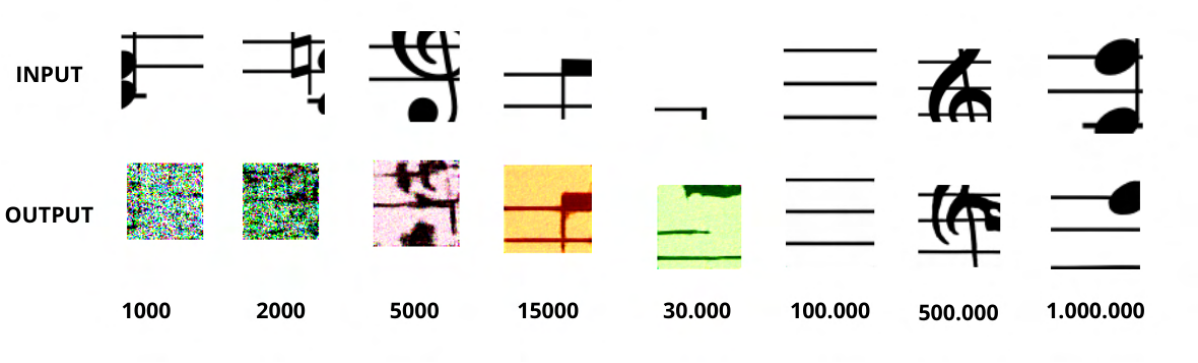


Figura 3.26: Evolución visual de la generación de parches a lo largo de las iteraciones de entrenamiento del modelo de difusión

Las figuras 3.27 y 3.28 representan dos de las imágenes de entrada usadas para evaluar al modelo resultante y sus salidas. La primera mantiene la base de entrada e incluye una nota correctamente posicionada, pero de valor rítmico inexistente; y la otra es algo más ruidosa y aunque se observan pinceladas de figuras musicales, no queda muy claro qué es lo que se representa. A pesar de que tenga estos errores, es un gran avance desde lo que se obtuvo en primeras pruebas (figura 3.25).

Se decidió no invertir más tiempo en la exploración del entrenamiento del modelo con este formato de imágenes y continuar con los espectrogramas. Sin embargo, es algo que queda abierto para continuarlo en un futuro, ya que con un mejor conjunto de datos y de mayor tamaño, se podrían alcanzar mejores resultados.

Entrenamiento con *dataset* de espectrograma definitivo

El entrenamiento definitivo se puede dividir en dos etapas claves. Durante la primera etapa se entrenó un modelo cuando aún no se había terminado el conjunto de datos definitivo y en la siguiente etapa ya se entrenó definitivamente con el conjunto de datos completo.

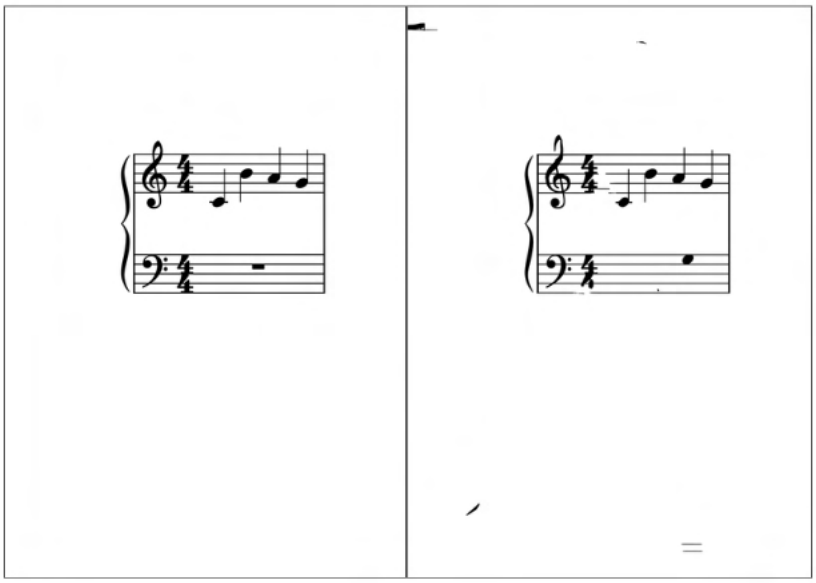


Figura 3.27: Compás usado como entrada a la izquierda y resultado de su evaluación con el modelo a la derecha

Durante esta primera etapa se entrenó un modelo con toda la información recopilada hasta el momento, y con los parámetros ya establecidos mencionados previamente, con el objetivo de comprobar varios puntos. En primer lugar, se quería comprobar si el modelo era capaz de generar un espectrograma que se pudiese transformar de vuelta a audio con el código ya preparado. Ya que a pesar de que es posible hacer la transformación inversa, que el espectrograma generado por el modelo se pudiese transformar de vuelta a un audio que sonase a música no era algo trivial. En segundo lugar, era importante que en el audio resultante sonase la capa dada como entrada superpuesta a otra capa generada por el modelo. El objetivo no era que se crease algo completamente distinto, sino que sobre la capa de entrada proporcionada se produjeran cambios o sugerencias. Este proceso de experimentación se explica de forma más extensa en la sección 3.4.1. Los resultados fueron positivos y se procedió al entrenamiento definitivo.

Entrenamiento de los modelos definitivos

Finalmente, se entrenaron los modelos definitivos. Una vez se había decidido que el modelo final de la herramienta sería uno entrenado con el *dataset* de espectrogramas, se optó por entrenar distintas configuraciones con el fin de comparar sus resultados y seleccionar la más idónea. La principal diferencia entre estos modelos fue la función de pérdida empleada.

La función de pérdida no es simplemente una fórmula matemática, sino una decisión que impacta directamente el comportamiento del modelo y las soluciones que encuentra. Se seleccionaron las siguientes: *Mean Absolute Error* (MAE), *Mean Squared Error* (MSE) y la función de pérdida *L1-Smooth* o *Huber loss*.

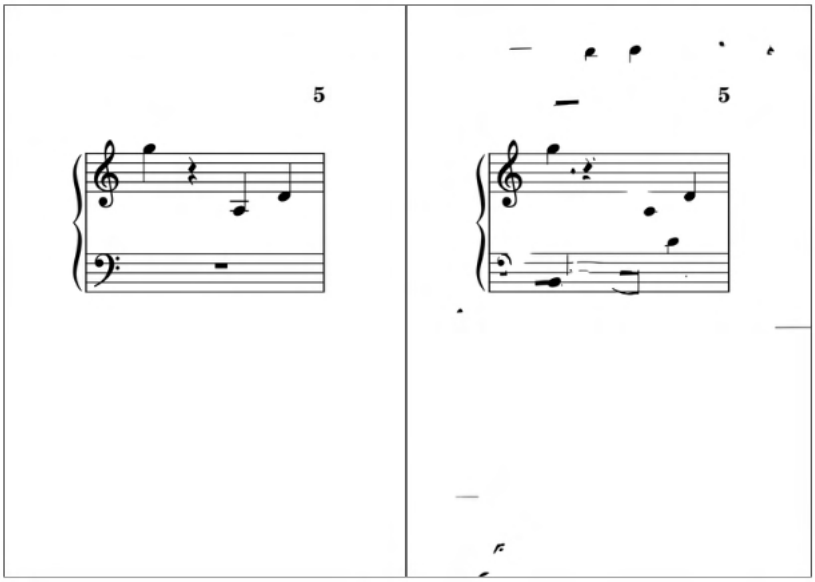


Figura 3.28: Compás usado como entrada a la izquierda y resultado de su evaluación con el modelo a la derecha

La función de pérdida MAE, es una función que calcula la media de las diferencias absolutas entre los valores predichos y los reales.

$$\sum_{i=1}^{D} |x_i - y_i|$$

Esta función penaliza los errores de forma lineal por lo que su principal ventaja es que es robusta frente a *outliers*. Básicamente, esto significa que los errores grandes no influyen de forma desproporcionada en el gradiente, lo cual es beneficioso en *datasets* con ruido considerable.

En segundo lugar, la función de pérdida MSE es una función que calcula la media de los cuadrados de las diferencias entre los valores predichos y reales.

$$\sum_{i=1}^{D} (x_i - y_i)^2$$

Esta función, a diferencia del MAE, penaliza los errores grandes de forma severa. Esto puede ser positivo porque acelera la convergencia inicial y negativo, porque le afectan bastante los *outliers*.

Por último, respecto a la función L1 *smooth* o *Huber loss*, esta es una mezcla entre ambas funciones mencionadas (MAE Y MSE). Básicamente su objetivo es combinar las ventajas de MAE y MSE. Se comporta como el MSE para errores pequeños y como el MAE para errores grandes.

$$L_\delta = \begin{cases} \frac{1}{2}(y - \hat{y})^2 & \text{si } |y - \hat{y}| < \delta \\ \delta\left(|y - \hat{y}| - \frac{1}{2}\delta\right) & \text{otherwise} \end{cases}$$

La figura 3.29 representa el comportamiento de estas tres funciones de pérdida según la magnitud del error. El eje horizontal representa el error, es decir, la diferencia entre el valor real y el valor predicho. El eje vertical representa el valor de la función de pérdida asociada a un error. Es decir, cuánto penaliza cada función un error determinado.

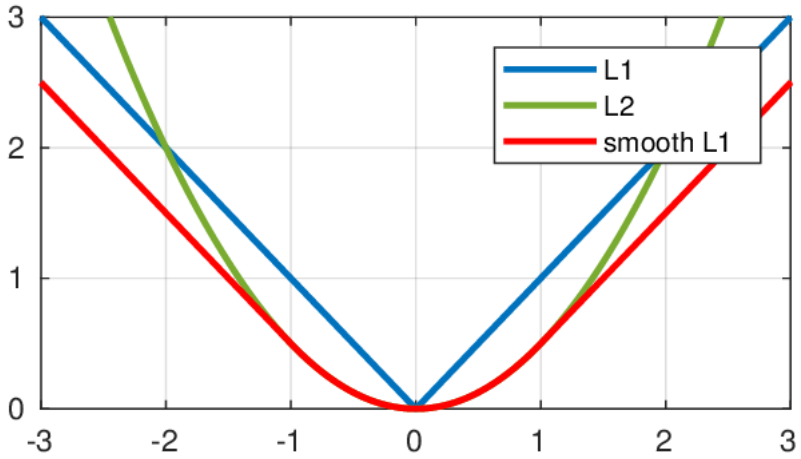


Figura 3.29: Comparación de las funciones de pérdida MAE, MSE y *L1-Smooth*, [64]

El entrenamiento se extendió por 1.000.000 de iteraciones, lo que equivale aproximadamente a 1500 épocas. Una vez entrenados, se hicieron varias evaluaciones donde se analizaba el audio resultante. También, se analizaron distintas métricas con el objetivo de comparar estos tres modelos y de verificar que el modelo había aprendido y las imágenes resultantes son de calidad. En la siguiente sección se explica detalladamente estas evaluaciones.

Las figuras 3.30, 3.31 y 3.32 representan la evolución visual del proceso de generación de parches a lo largo de las iteraciones del entrenamiento del modelo de difusión, cada una correspondiente a una función de pérdida distinta (MAE, MSE y L1-Smooth, respectivamente). En cada una de ellas, la fila superior (input) muestra el espectrograma de referencia y la inferior (output) los parches generados por el modelo.

En las primeras iteraciones, la salida del modelo consiste principalmente en ruido aleatorio y no parece tener una estructura definida. Sin embargo, a medida que el entrenamiento avanza (especialmente a partir de las 15.000 iteraciones), se empiezan a observar parches más cercanos a los de un espectrograma que se van refinando poco a poco.

En las iteraciones finales, los parches generados presentan una reducción significativa del ruido y se parecen más a la entrada condicionada. Esta mejora visual es un indicativo directo de la convergencia del proceso de aprendizaje del modelo y su capacidad para transformar el ruido en representaciones coherentes.

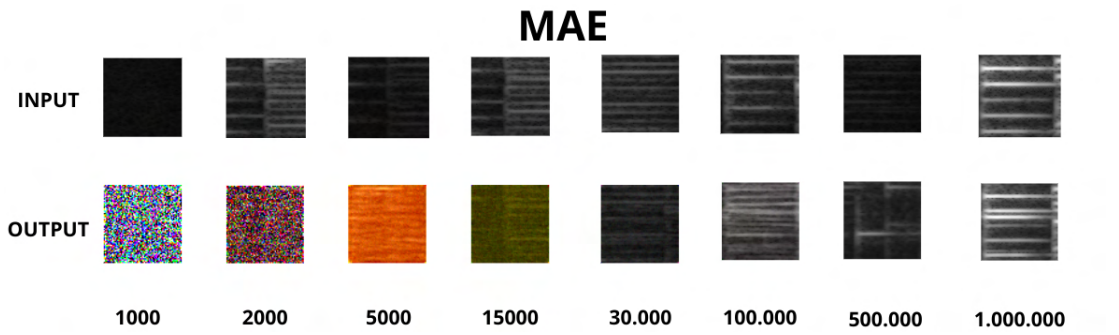


Figura 3.30: Evolución visual de la generación de parches a lo largo de las iteraciones de entrenamiento del modelo de difusión, con función de pérdida MAE

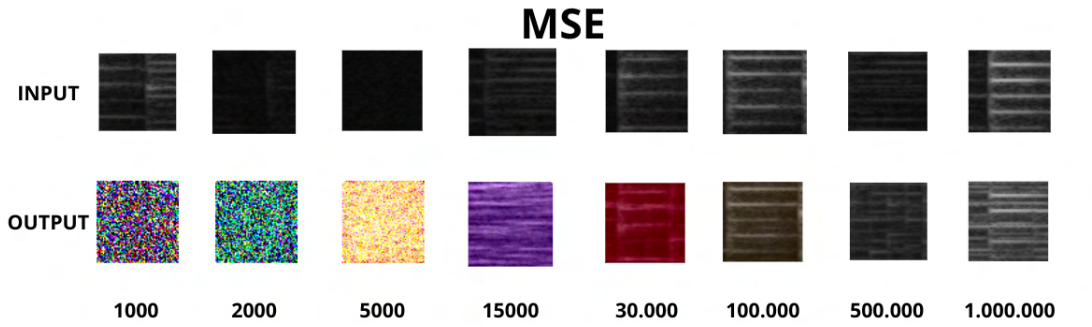


Figura 3.31: Evolución visual de la generación de parches a lo largo de las iteraciones de entrenamiento del modelo de difusión, con función de pérdida MSE

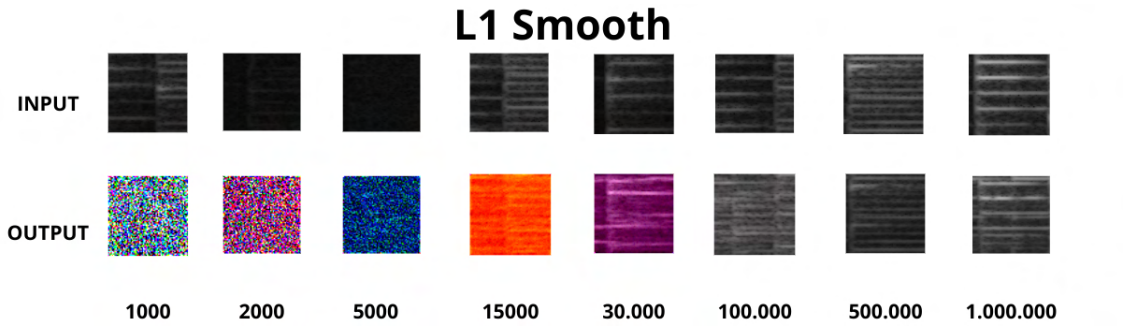


Figura 3.32: Evolución visual de la generación de parches a lo largo de las iteraciones de entrenamiento del modelo de difusión, con función de pérdida L1

3.4. Experimentos y resultados

3.4.1. Evaluación y comparación de modelos

Primeras evaluaciones

Como se mencionó en el apartado 3.3.3, el entrenamiento definitivo se dividió en dos etapas. Tras la primera, en la que se entrenó un modelo cuando aún no

se había finalizado el conjunto de datos definitivo, se realizó una primera toma de contacto con el proceso de evaluación. El objetivo principal era comprobar si tenía sentido seguir entrenando con espectrogramas, y para ello había que comprobar que los espectrogramas generados por el modelo tuvieran calidad suficiente como para transformarlos de nuevo a audio.

Para la evaluación del modelo entrenado, se siguió un proceso similar al que se busca más tarde automatizar para el usuario, comprobando que todos los pasos encajaban. Se preparó una pequeña "muestra" como composición en *Musescore*, exportándolo a audio y se evaluó con el primer modelo que había acabado de entrenar. Esta primera evaluación no solo evaluaba el modelo, sino cómo sería el proceso de reconstrucción, ya que había que transformar los espectrogramas de salida en un formato comprensible para el usuario final.

Entonces, primero se preparó en *Musescore* en versión partitura (figura 3.33). En las evaluaciones se ha trabajado con partituras enteras, no solo con compases individuales, para aprovechar las evaluaciones llevadas a cabo. Entonces, el audio resultante exportado de esta partitura se preprocesa, dividiéndola por segundos y transformando cada fragmento resultante en un espectrograma (siguiendo el esquema de la figura 3.10). Luego se evalúa con el modelo entrenado y los fragmentos devueltos por éste, son procesados de nuevo por otro *script*, de manera que los espectrogramas se transforman a audio de nuevo y se unen, resultando en un único audio. La figura 3.35 representa dos parejas de espectrogramas donde en ambos fragmentos el de la izquierda es el espectrograma de entrada y el de la derecha el de salida. Entonces, se finaliza con un audio original y un audio 'embellecido' con sugerencias por el modelo. Una vez se ha obtenido el audio final, se procesa con una aplicación llamada *Klangio*, para transcribirla a partitura (figura 3.34).

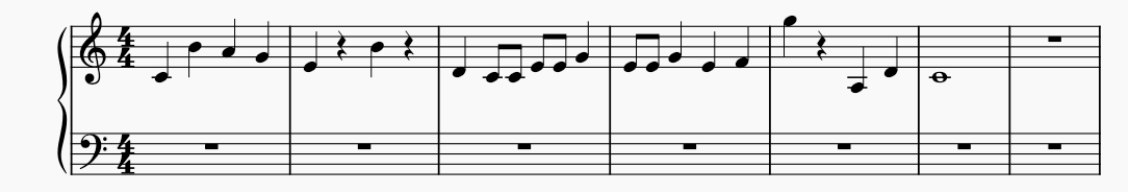


Figura 3.33: Partitura usada como entrada en la evaluación

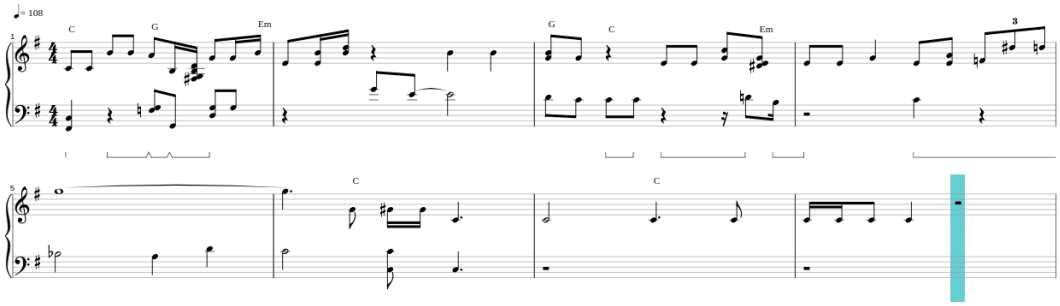


Figura 3.34: Partitura resultado del modelo y transcrito con *Klangio*

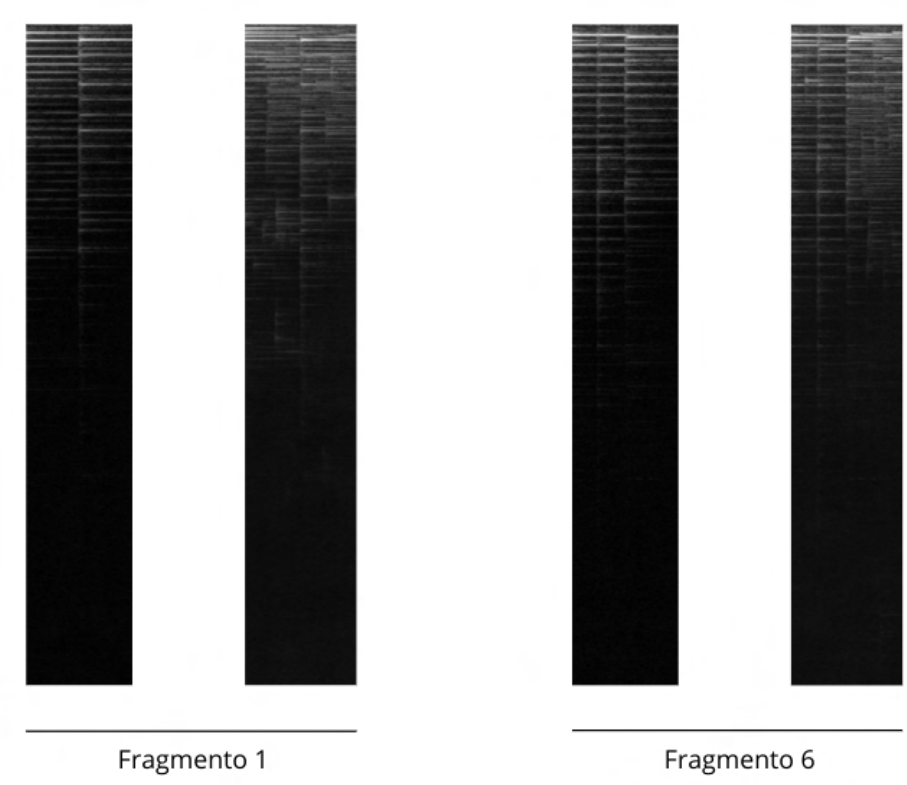


Figura 3.35: Espectrogramas usado como entrada en la evaluación y resultantes

La primera vez que se llevó a cabo este proceso de manera completa fue un gran paso. El modelo aún no había finalizado el entrenamiento por completo, pero se quería comprobar que se podía obtener un resultado positivo. El resultado deseado era que el audio resultante de los espectrogramas producidos por el modelo primero, siguiera sonando música y además se escuchase lo original mezclado con nuevos sonidos. El resultado fue positivo, aún quedaba trabajo por delante, pero se escuchaba la melodía original con sonidos que recordaban a los de un piano, aunque aún en una etapa muy temprana, y el entrenamiento continuó. Se pueden comparar los dos ejemplos de parejas de espectrogramas extraídos de esta evaluación viendo la figura 3.35, donde el espectrograma de entrada está menos "cargado" que el de salida.

Modelos definitivos

Una vez entrenados los tres modelos definitivos, se hizo una evaluación exhaustiva con el objetivo de compararlos y analizar de forma objetiva sus resultados.

Se preparó un conjunto de datos para la evaluación, con piezas musicales distintas a las empleadas en el entrenamiento, creando un conjunto de datos de 502 pares de imágenes de espectrogramas de piezas musicales existentes. Siguiendo el ejemplo del *paper* original, se calcularon varias métricas para medir objetivamente la calidad de las imágenes de espectrogramas resultantes, comparándolos con las salidas esperadas.

Primero, *Peak Signal-to-Noise Ratio* (PSNR) es una métrica que compara la potencia de la señal útil en una imagen con la potencia del ruido y cuanto más alto sea su valor, de mayor calidad es la imagen [65]. Otra métrica empleada es el índice de similitud estructural (SSIM), cuyo objetivo es medir la calidad percibida de una imagen a partir de otra imagen de referencia evaluando su similitud estructural [66].

La tabla 3.1 representa las métricas resultantes obtenidas distinguiendo los tres modelos entrenados. Analizando los resultados, el modelo entrenado con la función de pérdida MAE obtuvo el mejor rendimiento en ambas métricas. En contraste, el modelo entrenado con la función de pérdida MSE mostró un rendimiento inferior. Estos resultados se alinean con las características inherentes de cada función. El MAE penaliza los errores de forma lineal, lo que tiende a soluciones más nítidas y robustas frente a outliers; y el MSE penaliza los errores cuadráticamente, lo que a menudo resulta en imágenes más suavizadas o "borrosas" al buscar un promedio de las posibles soluciones. Por último, el modelo que utilizó la función L1-*Smooth* presentó un rendimiento muy similar al MSE, sin presentar las mejoras observadas con MAE en este contexto.

Basándose en estas métricas, la función de pérdida MAE es la más adecuada para el problema y el *dataset* específico de este proyecto y es el modelo que se empleará en la herramienta final.

Cuadro 3.1: Métricas de calidad (PSNR y SSIM) para modelos entrenados con distintas funciones de pérdida, donde valores más altos indican mayor calidad.

Modelos por función de error	PSNR	SSIM
MAE	27.3373	0.8068
MSE	26.7896	0.7848
L1 Smooth	26.7898	0.7894

	Snow100K-S [3]		Snow100K-L [3]	
	PSNR ↑	SSIM ↑	PSNR ↑	SSIM ↑
SPANet [44]	29.92	0.8260	23.70	0.7930
JSTASR [58]	31.40	0.9012	25.32	0.8076
RESCAN [43]	31.51	0.9032	26.08	0.8108
DesnowNet [3]	32.33	0.9500	27.17	0.8983
DDMSNet [59]	34.34	0.9445	28.85	0.8772
SnowDiff$_{64}$	**36.59**	**0.9626**	**30.43**	**0.9145**
SnowDiff$_{128}$	36.09	0.9545	30.28	0.9000
All-in-One [9]	-	-	28.33	0.8820
TransWeather [7]	32.51	0.9341	29.31	0.8879
WeatherDiff$_{64}$	**35.83**	**0.9566**	**30.09**	**0.9041**
WeatherDiff$_{128}$	35.02	0.9516	29.58	0.8941

(a) Image Desnowing

	Outdoor-Rain [14]	
	PSNR ↑	SSIM ↑
CycleGAN [46]	17.62	0.6560
pix2pix [45]	19.09	0.7100
HRGAN [14]	21.56	0.8550
PCNet [53]	26.19	0.9015
MPRNet [54]	28.03	0.9192
RainHazeDiff$_{64}$	**28.38**	**0.9320**
RainHazeDiff$_{128}$	26.84	0.9152
All-in-One [9]	24.71	0.8980
TransWeather [7]	28.83	0.9000
WeatherDiff$_{64}$	29.64	**0.9312**
WeatherDiff$_{128}$	**29.72**	0.9216

(b) Image Deraining & Dehazing

	RainDrop [12]	
	PSNR ↑	SSIM ↑
pix2pix [45]	28.02	0.8547
DuRN [56]	31.24	0.9259
RaindropAttn [55]	31.44	0.9263
AttentiveGAN [12]	31.59	0.9170
IDT [6]	31.87	0.9313
RainDropDiff$_{64}$	32.29	**0.9422**
RainDropDiff$_{128}$	**32.43**	0.9334
All-in-One [9]	**31.12**	0.9268
TransWeather [7]	30.17	0.9157
WeatherDiff$_{64}$	30.71	**0.9312**
WeatherDiff$_{128}$	29.66	0.9225

(c) Removing Raindrops

Figura 3.36: Comparaciones de las métricas PSNR y SSIM extraído del *paper* original [52]

Para contextualizar los resultados obtenidos en estas pruebas, la figura 3.36 representa las métricas de calidad (PSNR y SSIM) reportadas en el *paper* original [52] del modelo de difusión base empleado en este trabajo. Es importante tener

en cuenta que el *paper* original se basa en la reconstrucción de imágenes visuales bajo condiciones climáticas adversas (nieve, lluvia, niebla) mientras que en este TFG se aborda un área totalmente distinta a ésta. Por esto, la comparación numérica directa de los valores de las métricas no es válida, pero los valores de PSNR (entre 26.79 dB y 27.34 dB) y SSIM (entre 0.78 y 0.81) obtenidos con los modelos entrenados son prometedores y demuestran la notable capacidad de generalización de la arquitectura del modelo de difusión.

3.4.2. Prueba de usuario

El día 5 de junio de 2025, se realizó una visita al **Conservatorio Profesional de Música Victoria de los Ángeles**, donde Marta Moreno, estudiante de 4º curso del grado profesional de flauta travesera, hizo una prueba de la herramienta usando la versión del prototipo desarrollada hasta ese momento. El principal objetivo era poner a prueba la utilidad real de la herramienta para un músico a la hora de componer.

El proceso consistió en lo siguiente: Marta compuso una breve pieza a piano de 8 compases y al mismo tiempo la fue transcribiendo a partitura usando *Musescore*. La figura 3.37 es la partitura que representa esta pieza compuesta por Marta sin el uso de la herramienta.

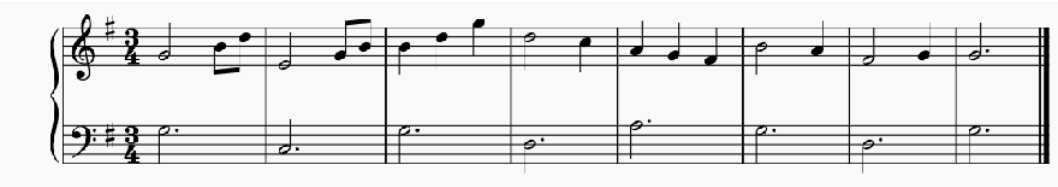


Figura 3.37: Pieza inicial compuesta por Marta Moreno

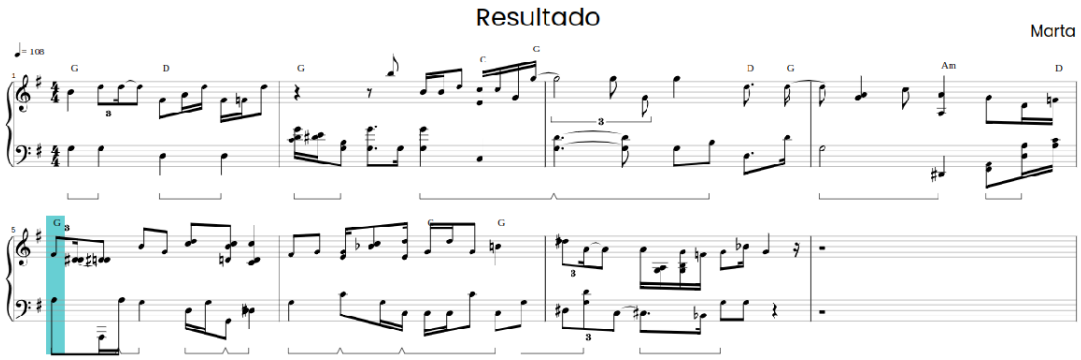


Figura 3.38: Versión generada por el modelo y transcrita con *Klangio* [67]

Una vez lista esta primera versión sencilla, automáticamente, se exportó la pieza a audio y se evaluó con el modelo. El audio resultante se transformó a una partitura gracias a *Klangio* y se le mostró en pantalla. La figura 3.38 es la partitura proveniente del audio generado por el modelo a partir de la versión inicial como entrada y la transcripción hecha con la herramienta de *Klangio* [67].

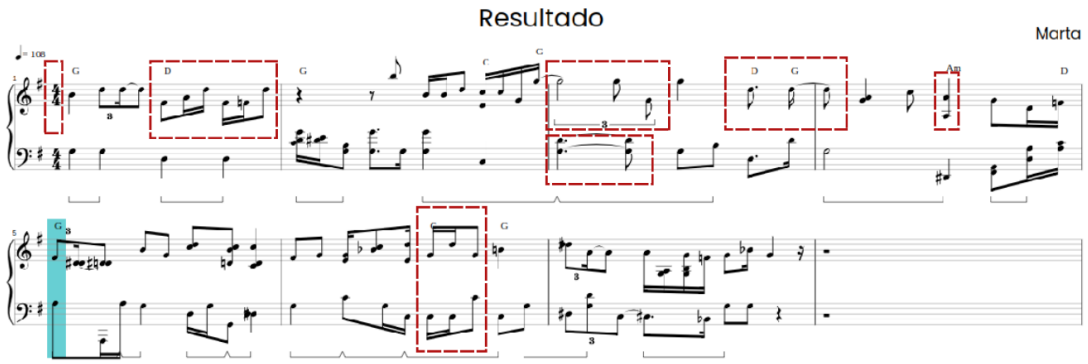


Figura 3.39: Versión generada por el modelo y transcrita con *Klangio* [67], esta vez con anotaciones propias que señalan las partes en las que se inspiró Marta para los cambios que dan lugar a la pieza final

Esta versión generada por el modelo, fue analizada y de ella se extrajo información útil que Marta consideraba buenas ideas para añadir a la composición original, como melodía, acompañamiento e incluso ritmos. Los fragmentos marcados en la figura 3.39 son varias de las características en las que se inspiró para realizar cambios. El resultado de este proceso iterativo fue la pieza final (ver figura 3.40).

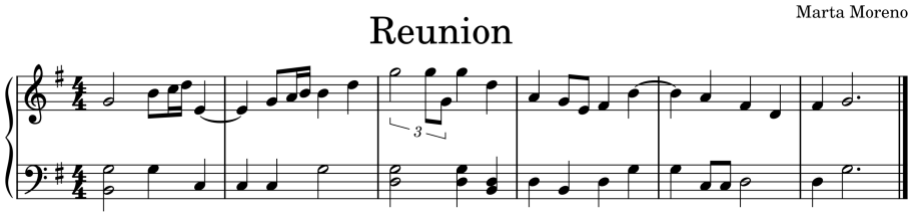


Figura 3.40: Pieza final compuesta por Marta Moreno e inspirada en sugerencias de la herramienta trabajada

Figura 3.41: Reunión para poner a prueba la herramienta en una sala de ensayo del conservatorio Profesional de Música Victoria de los Ángeles

La experiencia fue positiva. Marta como estudiante profesional de música dejó claro que la herramienta le había parecido útil y había disfrutado experimentando con ella. Uno de los aspectos clave que más le gustó fue que usar la herramienta había sido un proceso en el que no simplemente copiaba y pegaba una sugerencia sino que podía obtener ideas nuevas que quizás no se le hubieran ocurrido, mientras transformaba su versión inicial a su gusto.

3.5. Diseño del prototipo final

La implementación del modelo en una herramienta sencilla pero accesible directamente en el proceso de composición se enfocó a su uso directo de la mano de la aplicación *Musescore*. La idea es que el compositor pueda trabajar directamente escribiendo partituras en esta aplicación, popular entre músicos y no tenga que hacer grandes esfuerzos para usar la herramienta.

Se implementó una interfaz muy sencilla por medio de un *plugin* integrado en *Musescore*. El flujo de trabajo es el siguiente: mientras se trabaja en la aplicación, se pueden seleccionar los compases de los que se buscan sugerencias (figura 3.42) o incluso toda la partitura y se guardan en un archivo (figura 3.43). Entonces, se puede activar el *plugin* (figura 3.44), seleccionando en *Musescore* la opción de "Generar sugerencia" (figura 3.45).

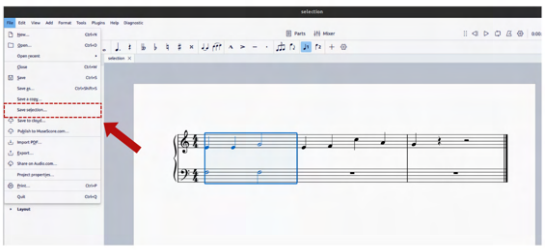


Figura 3.42: Selección de un fragmento de la partitura

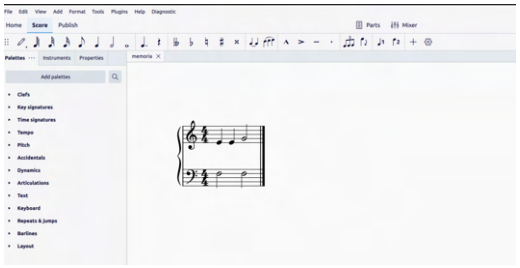


Figura 3.43: Archivo resultante del fragmento seleccionado

Entonces, una vez se ha seleccionado "Generar sugerencia", se abre la interfaz de la herramienta (figura 3.46). El primer paso es la selección del archivo guardado a partir del cual se quiere generar una sugerencia (figura 3.47).

Una vez seleccionado, se cargan los datos y realizan todas las transformaciones pertinentes automáticamente. Primero se obtiene el audio de la partitura seleccionada. Este audio se subdivide por segundos y cada fragmento del audio inicial se transforma a espectrogramas, para evaluarse directamente con el modelo. Todo ocurre mientras en la interfaz se muestra un texto de "Cargando los datos" (figura 3.48).

Cuando el preprocesamiento de datos ha terminado, se evalúan los espectrogramas de entrada y se obtienen los de salida. Los espectrogramas obtenidos como salida se transforman de vuelta a audio (figura 3.50 y se unen en un solo audio resultante que se guarda y se puede escuchar directamente.

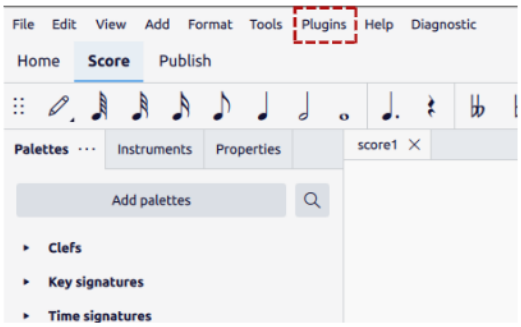


Figura 3.44: Selección del *plugins* dentro de *Musescore*

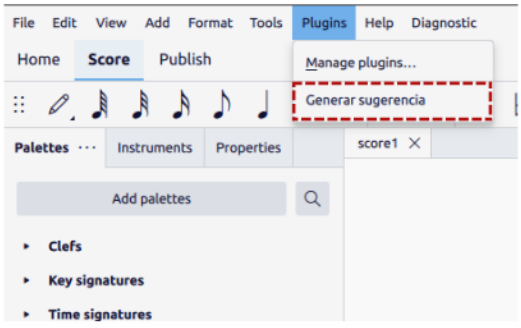


Figura 3.45: Selección del *plugin* "Generar sugerencia" dentro de *Musescore*

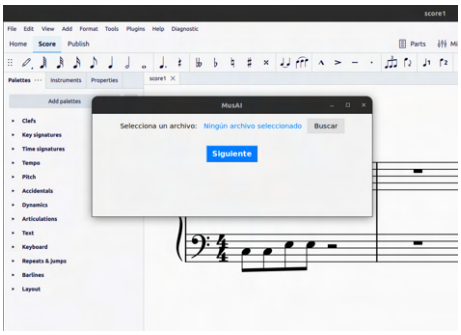


Figura 3.46: Interfaz abierta llamada a través del *plugin* de *Musescore*

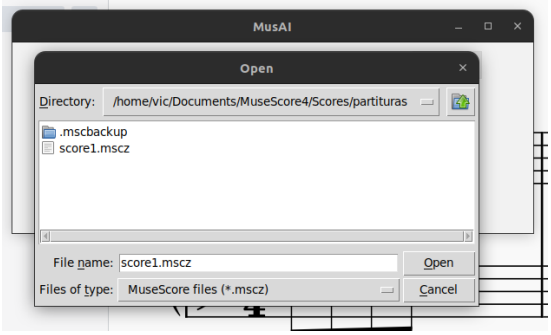


Figura 3.47: Selección del archivo a partir del cual se quiere generar una sugerencia

Además de esto, el objetivo principal era que este audio resultante se visualizara en formato de partitura. Hasta el momento se había empleado la web de *Klangio* para estas transformaciones. Entonces, para la integración en la aplicación se empleó la API, haciendo que la herramienta se conecte y haga una petición de transcripción del audio resultante en partitura (fichero 3.51). El fichero resultante se abre directamente en formato *PDF*, para permitir su visualización de forma rápida (figura 3.52).

Este es el flujo final de la herramienta. Durante su construcción hubo un par de problemas. Inicialmente, la idea consistía en que el *plugin* activara directamente la ventana de la interfaz para permitir los siguientes pasos del proceso. Sin embargo, esta aproximación se vio frustrada por limitaciones en la última versión de *Musescore*, que no permite la ejecución directa de programas externos. Esto sumado a otras restricciones complicó el desarrollo de la interfaz.

Entonces, el problema se afrontó reevaluando el proceso a seguir y adaptándose a las funcionalidades que la aplicación sí permite. La solución implementada consiste en que, lo que realmente hace el *plugin* al activarse, es mandar una señal escribiendo en un archivo específico, que al mismo tiempo está siendo monitorizado por un proceso externo, que cuando recibe la señal es el encargado

de lanzar la interfaz de usuario. Así, el flujo de la aplicación es el mismo y está integrado con *Musescore* facilitando su uso.

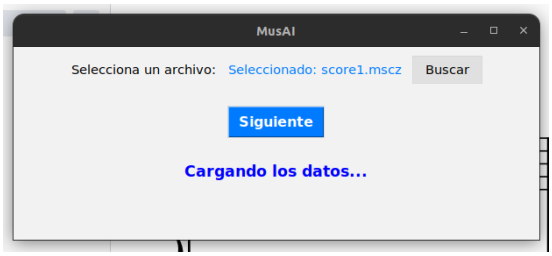


Figura 3.48: Preprocesamiento de datos

Figura 3.49: Evaluando los datos de entrada con el modelo

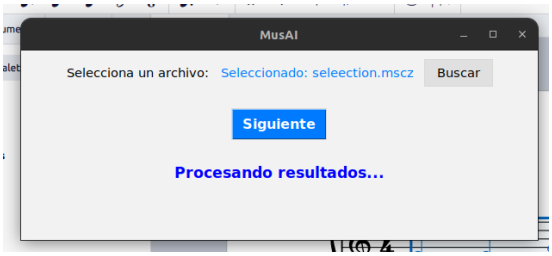


Figura 3.50: Procesando los datos resultantes

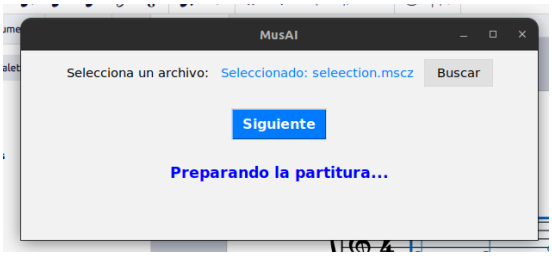


Figura 3.51: Ventana de generación de la partitura resultante mientras se conecta a la API de *Klangio*

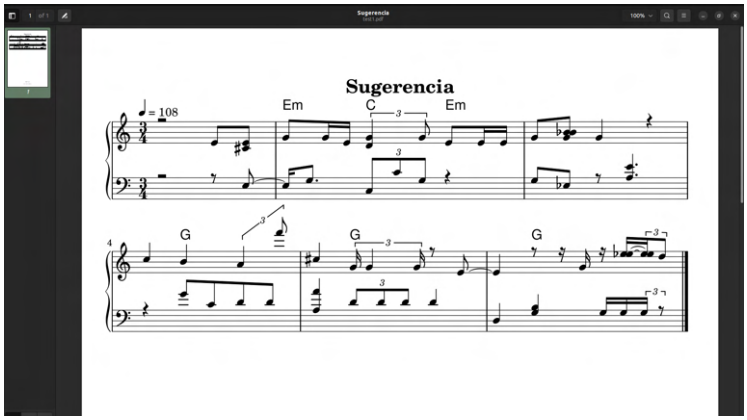


Figura 3.52: Ejemplo de sugerencia en formato partitura PDF recibida llamando a la API de *Klangio*

Capítulo 4

Análisis de impacto

En este capítulo se realiza un análisis del impacto potencial de los resultados obtenidos.

4.1. Casos de uso

Existen varios potenciales usuarios de la herramienta. En primer lugar, los usuarios directamente beneficiados con la herramienta son músicos profesionales y compositores que suelan utilizar *Musescore* para componer. Podrían usar la herramienta como un asistente de generación de ideas cuando sufran bloqueos creativos o simplemente quieran probar nuevas formas de componer.

Por otro lado, en el ámbito académico, cualquier estudiante de música podría usar la herramienta. Desde estudiantes de conservatorio hasta incluso estudiantes de instituto. Por ejemplo, para los estudiantes de instituto, la asignatura de música podría ser mucho más que simple teoría y podrían surgir mezclas interesantes donde explorasen esta intersección de tecnología y música. En el caso de los estudiantes de conservatorio, podrían usarla para practicar composición.

En el área de investigación, investigadores en música e IA podrían experimentar con la herramienta. La IA generativa es un área en constante exploración y se podrían hacer cosas como por ejemplo, emplear los audios generados para mejorar algoritmos de transcripción musical automática incluso con audios de menor calidad.

Por último, los potenciales usuarios más numerosos son la comunidad. Esta herramienta está conectada directamente a *Musescore*, muy popular y con una extensa comunidad de músicos que comparten y se descargan partituras. Es ventajoso porque esta comunidad podría emplear la herramienta para apoyarse en sus composiciones y además, acerca la composición también a personas con conocimiento más básicos de música, para que puedan experimentar y componer de forma lúdica.

4.2. Ventajas frente a alternativas

En el ámbito **personal**, la mayor ventaja que proporciona esta herramienta frente a otras es que busca la colaboración entre persona e IA, de manera que es un asistente que proporciona sugerencias o nuevas perspectivas y la persona que compone decide qué encaja y qué no. No es una herramienta que genera una pieza independientemente del artista sino que funciona a partir de lo se le proporciona y acompaña al humano durante esta tarea. Esto además puede ser de ayuda si existe bloqueo creativo o incluso si la gente que compone no tiene conocimientos muy extensos en teoría musical, proporcionando nuevas perspectivas que no se habrían considerado previamente.

Otra de sus ventajas es su **accesibilidad** frente a otras herramientas que están enfocadas a componer pero directamente sin mostrar partituras, sino audios. Esta herramienta está combinada con *Musescore*, de manera que se compone escribiendo en partituras directamente y las salidas del modelo también se muestran generando una partitura, además del audio. Esto facilita la accesibilidad para personas con discapacidad auditiva, que pueden visualizar todo el proceso. Además, también es accesible a personas con dificultades motoras que no puedan tocar físicamente instrumentos o personas sin conocimientos suficientes como para hacerlo. De hecho, es ventajosa culturalmente porque es una herramienta simple y permite que personas sin mucha experiencia con aplicaciones complejas o teoría musical puedan experimentar y participar en la creación musical.

En el ámbito **medioambiental**, la decisión de componer digitalmente directamente es clave. Al generar partituras digitales, se reduce el consumo de partituras a papel.

Se ha analizado el potencial impacto respecto a los **Objetivos de Desarrollo Sostenible (ODS)**, de la Agenda 2030 [68] siguiendo las ventajas mencionadas. El objetivo 10, reducción de las desigualdades, más concretamente el 10.2: “10.2 De aquí a 2030, potenciar y promover la inclusión social, económica y política de todas las personas, independientemente de su edad, sexo, discapacidad, raza, etnia, origen, religión o situación económica u otra condición”. Esta herramienta cumple con esto por su accesibilidad e inclusión de personas sin importar su nivel de conocimientos de teoría musical o posean instrumentos físicos.

Otro objetivo que se cumple, según lo explicado de la reducción en el consumo de papel es la meta 12, más concretamente: “12.5 De aquí a 2030, reducir considerablemente la generación de desechos mediante actividades de prevención, reducción, reciclado y reutilización”.

Por último, con la búsqueda de extender la investigación de IA en el área musical se cumple uno de los objetivos de la meta 9, “9.5 Aumentar la investigación científica y mejorar la capacidad tecnológica de los sectores industriales de todos los países, en particular los países en desarrollo, entre otras cosas fomentando la innovación y aumentando considerablemente, de aquí a 2030, el número de personas que trabajan en investigación y desarrollo por millón de habitantes y los gastos de los sectores público y privado en investigación y desarrollo”.

4.3. Efectos adversos

Existen ciertos efectos adversos a tener en cuenta. En primer lugar, es posible que los usuarios le den un uso excesivo y sufran cierta dependencia al usar esta herramienta a pesar de que su objetivo principal sea de asistencia no de reemplazo.

También, el uso constante de esta herramienta podría provocar que las piezas compuestas consten de ciertas características que suela sugerir el modelo y que parezca que haya piezas algo uniformes o de estilo similar. Esto se puede combatir en el futuro con un *dataset* más grande y diverso e implementando mejoras.

Por último, a pesar de que se reduzca el uso del papel, es cierto que también hay un consumo energético por los recursos necesarios a la hora de procesar los datos y generar las sugerencias. En el futuro, se implementarán algoritmos que optimicen lo máximo posible la eficiencia energética.

Capítulo 5

Conclusiones y trabajo futuro

5.1. Conclusiones

El desarrollo de este TFG ha sido extenso. Comenzó con la idea de crear una herramienta o *plugin* dentro de una aplicación y se transformó en un trabajo de investigación en el área de IA generativa.

Se han alcanzado resultados mejores de lo esperados y académicamente ha sido muy interesante trabajar en un proyecto que ha pasado por tantas fases, desde la investigación, preparación de un *dataset*, entrenamiento, evaluación y prueba con usuarios. Desde luego, profundizar en este área ha sido aún más motivador por ver todo lo que aún queda por descubrir y trabajar.

Para empezar, la construcción de un *dataset* propio con el objetivo de innovar, fue una de las semillas iniciales de este TFG. La recopilación de datos fue un aprendizaje continuo, donde resultó mucho más manual y lento de lo esperado en un primer momento. Además, se realizaron múltiples pruebas para dar con el formato de imagen final empleado en este proyecto. A la par que se construía el conjunto de datos, se empezaba a explorar el entrenamiento de un modelo preexistente. Esta etapa estuvo llena de altibajos e investigación de soluciones existentes, hasta encontrar una manera en la que obtener soluciones buenas.

Tras este largo proceso, hay una serie de conclusiones a las que se ha llegado. Primero, es un gran paso haber obtenido resultados donde parte de la propia música inicial y le suma pinceladas. Es muy positivo que el modelo sea capaz de generar espectrogramas en base a otros de entrada condicionales que tengan sentido y calidad suficiente como para transformarse de vuelta a audio. Esto es un gran paso, tras todos los intentos previos durante este mismo proyecto enfocados en obtener algo así del modelo. Aunque la realidad es que al transformar los espectrogramas de vuelta a audios, éstos audios no son de calidad perfecta, la calidad es suficiente para que aplicaciones con *Klangio* puedan hacer la transformación a partitura transcribiéndola y cumpliendo su función esencial que es generar ideas que le sirvan de alguna manera al compositor. En el futuro se puede seguir trabajando para mejorar los sonidos generados y añadidos por el modelo y que sea difícil diferenciarles del piano original.

Respecto a la interfaz de la herramienta, algo muy positivo es que es sencilla y busca no romper con el flujo de trabajo ya construido por los compositores, integrándose directamente en *Musescore*. El desarrollo de la interfaz fue algo tedioso por ciertas restricciones de la propia aplicación para el desarrollo de *plugins* con los que no se contaba en un principio. Sin embargo, una vez estos problemas se resolvieron, el resultado final queda bien integrado y es sencillo de utilizar.

En resumen, el objetivo de este TFG, que era construir una herramienta generativa que brinde sugerencias al usuario, se ha alcanzado y en un futuro se desarrollarán las mejoras propuestas.

5.2. Trabajo futuro

Algunas de las mejoras propuestas para el futuro de este TFG son:

Mejora del dataset Se podría seguir trabajando en el *dataset*, quizás fusionándolo con más información y expandiéndolo o incluso probar con alguno de los *datasets* populares en IA generativa. Idealmente, se colaboraría con músicos para seguir la línea de un *dataset* propio y novedoso.

***End-user* mucho más fluido y simplificado** El prototipo hace su función y es útil, sin embargo, se podría incluir directamente dentro de una aplicación; de manera que dentro de la aplicación pudieras seleccionar los cambios y fusionarlos con lo ya establecido, mejorando el flujo de trabajo.

Entrenamiento Los resultados obtenidos han sido mucho mejores de los esperados. Aún así, podría seguir entrenándose el modelo y haciendo pruebas con cambios en el *dataset* para comprobar si se puede llegar a un audio generado con el que sea muy difícil discernir entre la capa existente de entrada y lo generado por IA.

Añadir una capa de correcciones Añadir una capa en la aplicación que procese el resultado del modelo y elimine de forma automática y determinista partes que no cumplan reglas armónicas. Una capa similar a un repaso de "ortografía" pero musical. Esto mejoraría la calidad de las sugerencias musicales.

Formato de imágenes como partituras Desarrollar más en este ámbito y con un conjunto de datos más extenso y comprobar si se pueden obtener mejores resultados.

***Input* de audio directamente** El músico podría tocar un instrumento directamente y la aplicación tomaría este audio como entrada de forma integrada en la aplicación.

Bibliografía

[1] *ChatGPT*, Sitio web. dirección: https://chatgpt.com/ (visitado 04-06-2025).

[2] *Gemini*, Sitio web. dirección: https://gemini.google.com/ (visitado 04-06-2025).

[3] *Dalle*, Sitio web. dirección: https://openai.com/es-ES/index/dall-e/ (visitado 04-06-2025).

[4] Y.-W. Chen y S.-C. Pei, «Always Clear Days: Degradation Type and Severity Aware All-In-One Adverse Weather Removal», *arXiv preprint arXiv:2310.18293*, 2023.

[5] Y. Zhu, T. Wang, X. Fu et al., «Learning Weather-General and Weather-Specific Features for Image Restoration Under Multiple Adverse Weather Conditions», en *Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023.

[6] P. W. Patil, S. Gupta, S. Rana, S. Venkatesh y S. Murala, «Multi-weather Image Restoration via Domain Translation», en *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, oct. de 2023, págs. 21 696-21 705.

[7] B. Wang, J. Han, X. Zhao, Y. Yin, L. Chen y P. Childs, «Creative Combinational Design through Generative AI in Different Dimensional Representations: An Exploration», *Design and Artificial Intelligence*, pág. 100 006, 2025, ISSN: 3050-7413. DOI: https://doi.org/10.1016/j.daai.2025.100006. dirección: https://www.sciencedirect.com/science/article/pii/S3050741325000060.

[8] S. Feuerriegel, J. Hartmann, C. Janiesch y P. Zschech, «Generative AI», *Business & Information Systems Engineering*, vol. 66, n.º 1, págs. 111-126, sep. de 2023, ISSN: 1867-0202. DOI: 10.1007/s12599-023-00834-7. dirección: http://dx.doi.org/10.1007/s12599-023-00834-7.

[9] *Claude*, Sitio web. dirección: https://claude.ai/ (visitado 04-06-2025).

[10] J. Han, F. Shi, L. Chen y P. R. Childs, «The Combinator – a computer-based tool for creative idea generation based on a simulation approach», *Design Science*, vol. 4, e11, 2018. DOI: 10.1017/dsj.2018.7.

[11] P. Dhariwal, H. Jun, C. Payne, J. W. Kim, A. Radford e I. Sutskever, *Jukebox: A Generative Model for Music*, 2020. arXiv: 2005.00341 [eess.AS]. dirección: https://arxiv.org/abs/2005.00341.

[12] *SpotifyRecomm*, Sitio web. dirección: https://research.atspotify.com/search-recommendations/ (visitado 07-06-2025).

[13] *Shazam*, Sitio web. dirección: https://www.shazam.com/es-es (visitado 08-06-2025).

[14] O. Cífka, U. Simsekli y G. Richard, «Groove2Groove: One-Shot Music Style Transfer With Supervision From Synthetic Data», *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. PP, págs. 1-1, ago. de 2020. DOI: 10.1109/TASLP.2020.3019642.

[15] Y. Zhang, «An IoT-enhanced automatic music composition system integrating audio-visual learning with transformer and SketchVAE», *Alexandria Engineering Journal*, vol. 113, págs. 378-390, 2025, ISSN: 1110-0168. DOI: https://doi.org/10.1016/j.aej.2024.10.115. dirección: https://www.sciencedirect.com/science/article/pii/S1110016824012808.

[16] M. Chen, D. Tang, Y. Xiang et al., «Instrument sound classification using a music-based feature extraction model inspired by Mozart's Turkish March pattern», *Alexandria Engineering Journal*, 2025. dirección: https://api.semanticscholar.org/CorpusID:275871126.

[17] M. Civit, J. Civit-Masot, F. Cuadrado y M. Escalona, «A systematic review of artificial intelligence-based music generation: Scope, applications, and future trends», *Expert Systems with Applications*, vol. 209, pág. 118 190, jul. de 2022. DOI: 10.1016/j.eswa.2022.118190.

[18] A. Elgammal. «AhmedArticle». Sitio web. (), dirección: https://theconversation.com/how-a-team-of-musicologists-and-computer-scientists-completed-beethovens-unfinished-10th-symphony-168160?xid=PS_smithsonian (visitado 07-06-2025).

[19] *TokyoSong*, Sitio web. dirección: https://www.olympics.com/ioc/news/tokyo-2020-launches-make-the-beat-a-project-inviting-fans-worldwide-to-energise-venues-and-encourage-athletes (visitado 07-06-2025).

[20] L. Almazaydeh, S. Atiewi, A. A. Tawil y K. M. Elleithy, «Arabic Music Genre Classification Using Deep Convolutional Neural Networks (CNNs)», *Computers, Materials & Continua*, 2022. dirección: https://api.semanticscholar.org/CorpusID:248336865.

[21] H.-W. Dong, W.-Y. Hsiao, L.-C. Yang e Y.-H. Yang, *MuseGAN: Multi-track Sequential Generative Adversarial Networks for Symbolic Music Generation and Accompaniment*, 2017. arXiv: 1709.06298 [eess.AS]. dirección: https://arxiv.org/abs/1709.06298.

[22] J.-Y. Zhu, T. Park, P. Isola y A. A. Efros, *Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks*, 2020. arXiv: 1703.10593 [cs.CV]. dirección: https://arxiv.org/abs/1703.10593.

[23] G. Brunner, Y. Wang, R. Wattenhofer y S. Zhao, *Symbolic Music Genre Transfer with CycleGAN*, 2018. arXiv: 1809.07575 [cs.SD]. dirección: https://arxiv.org/abs/1809.07575.

[24] C.-Z. A. Huang, A. Vaswani, J. Uszkoreit et al., *Music Transformer*, 2018. arXiv: 1809.04281 [cs.LG]. dirección: https://arxiv.org/abs/1809.04281.

[25] Y.-S. Huang e Y.-H. Yang, *Pop Music Transformer: Beat-based Modeling and Generation of Expressive Pop Piano Compositions*, 2020. arXiv: 2002.00212 [cs.SD]. dirección: https://arxiv.org/abs/2002.00212.

[26] H. Zhao, S. Min, J. Fang y S. Bian, «AI-driven music composition: Melody generation using Recurrent Neural Networks and Variational Autoencoders», *Alexandria Engineering Journal*, vol. 120, págs. 258-270, 2025, ISSN: 1110-0168. DOI: https://doi.org/10.1016/j.aej.2025.02.013. dirección: https://www.sciencedirect.com/science/article/pii/S1110016825001802.

[27] R. Guo, I. Simpson, T. Magnusson, C. Kiefer y D. Herremans, *A variational autoencoder for music generation controlled by tonal tension*, 2020. arXiv: 2010.06230 [cs.SD]. dirección: https://arxiv.org/abs/2010.06230.

[28] A. Muhamed, L. Li, X. Shi et al., «Symbolic Music Generation with Transformer-GANs», *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, n.º 1, págs. 408-417, mayo de 2021. DOI: 10.1609/aaai.v35i1.16117. dirección: https://ojs.aaai.org/index.php/AAAI/article/view/16117.

[29] *ChordsGenerator*, Sitio web. dirección: https://github.com/rxdha1/The-Chord-Generator/tree/main (visitado 07-06-2025).

[30] *GoogleMagenta*, Sitio web. dirección: https://magenta.withgoogle.com/ (visitado 07-06-2025).

[31] *ChordProgression*, Sitio web. dirección: https://github.com/asigalov61/Chords-Progressions-Transformer (visitado 07-06-2025).

[32] *AccoMontage2*, Sitio web. dirección: https://github.com/billyblu2000/AccoMontage2/tree/master (visitado 07-06-2025).

[33] *Audacity*, Sitio web. dirección: https://www.audacityteam.org/ (visitado 07-06-2025).

[34] *AbletonLive*, Sitio web. dirección: https://www.ableton.com/en/ (visitado 07-06-2025).

[35] *AudacityPluginsAI*, Sitio web. dirección: https://plugins.audacityteam.org/ai-plugins/ai-plugins (visitado 07-06-2025).

[36] *Documentación, tutorial y ejemplos del plugin "Music Generation & Continuation*, Sitio web. dirección: https://github.com/intel/openvino-plugins-ai-audacity/blob/main/doc/feature_doc/music_generation/README.md (visitado 07-06-2025).

[37] *Ableton Live - Magenta Studio Tool*, Sitio web. dirección: https://www.ableton.com/en/blog/magenta-studio-free-ai-tools-ableton-%20live/ (visitado 07-06-2025).

[38] *Musescore*, Sitio web. dirección: https://musescore.com/ (visitado 07-06-2025).

[39] *Loudly*, Sitio web. dirección: https://www.loudly.com/music/ai-music-%20generator (visitado 07-06-2025).

[40] *Canva*, Sitio web. dirección: https://www.canva.com/es_mx/funciones/ia-crear-musica/ (visitado 07-11-2024).

[41] *Lakh Pianoroll Dataset*, Sitio web del dataset. dirección: https://hermandong.com/lakh-pianoroll-dataset/ (visitado 08-06-2025).

[42] *The Lakh MIDI Dataset v0.1*, Sitio web del dataset. dirección: https://colinraffel.com/projects/lmd/ (visitado 08-06-2025).

[43] T.-C. Pricop y A. Iftene, «Music Generation with Machine Learning and Deep Neural Networks», *Procedia Computer Science*, vol. 246, págs. 1855-1864, 2024, 28th International Conference on Knowledge Based and Intelligent information and Engineering Systems (KES 2024), ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2024.09.692. dirección: https://www.sciencedirect.com/science/article/pii/S1877050924027522.

[44] *Web sobre tablaturas de guitarra*, Sitio web. dirección: https://guitarlions.com/blog/post/52/tablatura-en-guitarra/ (visitado 08-06-2025).

[45] *MAESTRO Dataset*, Sitio web del dataset. dirección: https://magenta.tensorflow.org/datasets/maestro (visitado 08-06-2025).

[46] *SunoCaps Dataset*, Sitio web del dataset en la plataforma Kaggle. dirección: https://www.kaggle.com/datasets/2ab339f9bbc64cf1b13beb52b0bef0a3f8401 (visitado 03-06-2025).

[47] *Suno*, Sitio web. dirección: https://suno.com/home (visitado 03-06-2025).

[48] M. Civit, V. Drai-Zerbib, D. Lizcano y M. Escalona, «SunoCaps: A novel dataset of text-prompt based AI-generated music with emotion annotations», *Data in Brief*, vol. 55, pág. 110743, 2024, ISSN: 2352-3409. DOI: https://doi.org/10.1016/j.dib.2024.110743. dirección: https://www.sciencedirect.com/science/article/pii/S2352340924007078.

[49] *Bach Chorales Dataset*, Sitio web del dataset. dirección: https://archive.ics.uci.edu/dataset/25/bach+chorales (visitado 08-06-2025).

[50] J. M. J. Valanarasu, R. Yasarla y V. M. Patel, *TransWeather: Transformer-based Restoration of Images Degraded by Adverse Weather Conditions*, 2021. arXiv: 2111.14813 [cs.CV].

[51] W.-T. Chen, Z.-K. Huang, C.-C. Tsai, H.-H. Yang, J.-J. Ding y S.-Y. Kuo, «Learning Multiple Adverse Weather Removal via Two-stage Knowledge Learning and Multi-contrastive Regularization: Toward a Unified Model», 2022.

[52] O. Özdenizci y R. Legenstein, «Restoring vision in adverse weather conditions with patch-based denoising diffusion models», *IEEE Transactions on Pattern Analysis and Machine Intelligence*, págs. 1-12, 2023. DOI: 10.1109/TPAMI.2023.3238179.

[53] O. Özdenizci y R. Legenstein, *Restoring Vision in Adverse Weather Conditions with Patch-Based Denoising Diffusion Models*, 2022. arXiv: 2207.14626 [cs.CV]. dirección: https://arxiv.org/abs/2207.14626.

[54] *Tema musical de la Wii de Nintendo*, Sitio web. dirección: https://www.youtube.com/watch?v=x2NzoLMWAwQ&ab_channel=GamingSoundFX (visitado 30-06-2025).

[55] *Tutorial transformación audios a espectrogramas*, Sitio web. dirección: https://www.kaggle.com/code/msripooja/steps-to-convert-audio-clip-to-spectrogram/notebook#Step-1:-Let's-import-all-the-required-libraries (visitado 17-06-2025).

[56] B. McFee, M. McVicar, D. Faronbi et al., *librosa/librosa: 0.11.0*, ver. 0.11.0, mar. de 2025. DOI: 10.5281/zenodo.15006942. dirección: https://doi.org/10.5281/zenodo.15006942.

[57] *Fast Fourier Transformation (FFT)*, Sitio web. dirección: https://www.nti-audio.com/en/support/know-how/fast-fourier-transform-fft (visitado 27-06-2025).

[58] D. Griffin y J. Lim, «Signal estimation from modified short-time Fourier transform», *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 32, n.º 2, págs. 236-243, 1984. DOI: 10.1109/TASSP.1984.1164317.

[59] *Explicación del algoritmo de Griffin Lim por la universidad de Aalto en Finlandia*, Sitio web. dirección: https://speechprocessingbook.aalto.fi/Modelling/griffinlim.html (visitado 23-06-2025).

[60] *Griffinlim*, Repositorio de código. dirección: https://github.com/bkvogel/griffin_lim/tree/master (visitado 18-06-2025).

[61] *Imagen de mordente (adorno musical)*, Sitio web. dirección: https://es.wikipedia.org/wiki/Mordente (visitado 24-06-2025).

[62] *Imagen de trino (adorno musical)*, Sitio web. dirección: https://es.wikipedia.org/wiki/Trino_(m%C3%BAsica) (visitado 24-06-2025).

[63] *Imagen de apoyatura (adorno musical)*, Sitio web. dirección: https://es.wikipedia.org/wiki/Apoyatura (visitado 24-06-2025).

[64] Z.-H. Feng, J. Kittler, M. Awais, P. Huber y X.-J. Wu, «Wing Loss for Robust Facial Landmark Localisation with Convolutional Neural Networks», jun. de 2018. DOI: 10.1109/CVPR.2018.00238.

[65] *Peak signal to noise ration (PSNR)*, Sitio web. dirección: https://www.sciencedirect.com/topics/computer-science/peak-signal-to-noise-ratio (visitado 27-06-2025).

[66] *Índice de similitud estructural*, Sitio web. dirección: https://www.kaggle.com/code/alsaniipe/image-similarity-index-ssim-analysis (visitado 27-06-2025).

[67] *Klangio*, Sitio web. dirección: https://klang.io/es/ (visitado 27-06-2025).

[68] *Objetivos de Desarrollo Sostenible (ODS) de la Agenda 2030*, Sitio web. dirección: https://www.un.org/sustainabledevelopment/es/objetivos-de-desarrollo-sostenible/ (visitado 25-06-2025).

Anexos

Apéndice A

Primer anexo

Repositorio de Github con todo el código desarrollado

https://github.com/vickynnni/HerramientaComposicionIA

En él se encuentra todo el código del proyecto, desde el modelo hasta el código del *plugin* de *Musescore*.

En la carpeta modelo se encuentra el código del repositorio [52] del modelo de difusión empleado, mezclado con algunos archivos propios como la configuración, los datos de entrenamiento y un modelo entrenado.

En la carpeta de archivos se encuentran ejemplos realizados, todos los modelos entrenados y los archivos brutos de la evaluación de las métricas expuestas.

En la carpeta principal se encuentran los *scripts* principales de la interfaz (*app.py*, *main.py* y *watcher.py*) y otro archivo *audio utilities* que es del repositorio [60], empleado para la transformación de audio a espectrograma y viceversa.

Este documento esta firmado por

	**Firmante**	CN=tfgm.fi.upm.es, OU=CCFI, O=ETS Ingenieros Informaticos - UPM, C=ES
	Fecha/Hora	Wed Jul 02 18:24:35 CEST 2025
	Emisor del Certificado	EMAILADDRESS=camanager@etsiinf.upm.es, CN=CA ETS Ingenieros Informaticos, O=ETS Ingenieros Informaticos - UPM, C=ES
	Numero de Serie	561
	Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)