



EDUCACIÓN
SECRETARÍA DE EDUCACIÓN PÚBLICA



TECNOLÓGICO
NACIONAL DE MÉXICO®



TECNOLÓGICO NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE TOLUCA

DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

GENERACIÓN DE ARMONÍAS UTILIZANDO REDES NEURONALES ARTIFICIALES, UNA HERRAMIENTA PARA LA COMPOSICIÓN MUSICAL

TESIS QUE PRESENTA

RICARDO IVÁN PÉREZ CERVANTES

NO. DE CONTROL: M23280232

COMO REQUISITO PARA OBTENER EL GRADO DE

MAESTRO EN CIENCIAS DE LA INGENIERÍA

DIRECTOR: **DR. FEDERICO DEL RAZO LÓPEZ**

CODIRECTORA: **DRA. ERÉNDIRA RENDÓN LARA**

METEPEC, ESTADO DE MÉXICO, MÉXICO

JUNIO 2025



Instituto Tecnológico de Toluca
División de Estudios de Posgrado e Investigación

Metepec, Edo. de México, 17/junio/2025
DEPI-3200-335/2025

C. RICARDO IVAN PEREZ CERVANTES
CANDIDATO AL GRADO DE MAESTRO
EN CIENCIAS DE LA INGENIERÍA

De acuerdo con los Lineamientos para la Operación de los Estudios de Posgrado en el Tecnológico Nacional de México y las disposiciones de este Instituto, habiendo cumplido con todas las indicaciones que la Comisión Revisora realizó con respecto a su trabajo de Tesis titulado **"GENERACIÓN DE ARMONÍAS UTILIZANDO REDES NEURONALES ARTIFICIALES, UNA HERRAMIENTA PARA LA COMPOSICIÓN MUSICAL"**, la División de Estudios de Posgrado e Investigación de este Instituto, concede la autorización para proceder a la publicación del trabajo final a través del repositorio institucional.

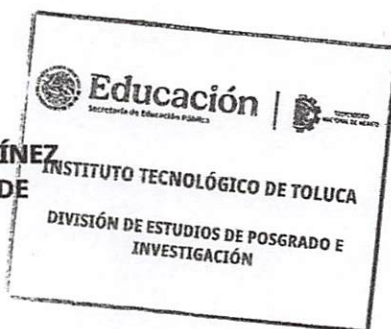
Sin otro particular por el momento, me despido de usted.

ATENTAMENTE

Excelencia en Educación Tecnológica

"Educación, integridad y ciencia"

FRANCISCO JAVIER ILLESCAS MARTÍNEZ
JEFE DE LA DIVISIÓN DE ESTUDIOS DE
POSGRADO E INVESTIGACIÓN



ccp. Archivo
FJIM/mcdh



2025
Año de
La Mujer
Indígena

Av. Tecnológico S/N, Col. Agrícola Bellavista, Metepec, Edo. de México,
C.P. 52149, Tels. Dirección: 7222087205, Conmut.: 7222087200
e-mail: info@toluca.tecnm.mx tecnm.mx | toluca.tecnm.mx





Instituto Tecnológico de Toluca
División de Estudios de Posgrado e Investigación

Metepec, Edo. de México, **26/mayo/2025**

DEPI-3200-288/2025

FRANCISCO JAVIER ILLESCAS MARTÍNEZ
JEFE DE LA DIVISIÓN DE ESTUDIOS
DE POSGRADO E INVESTIGACIÓN
PRESENTE

Por este medio comunicamos a usted que, la Comisión Revisora designada para analizar la tesis denominada **"GENERACIÓN DE ARMONÍAS UTILIZANDO REDES NEURONALES ARTIFICIALES, UNA HERRAMIENTA PARA LA COMPOSICIÓN MUSICAL"**, ha dictaminado que dicho trabajo reúne las características de contenido y calidad para proceder a la publicación del trabajo final a través del repositorio institucional. Lo anterior como parte de los requisitos para obtener el grado académico de Maestro en Ciencias de la Ingeniería que presenta el **C. RICARDO IVAN PEREZ CERVANTES** con número de control **M23280232** para sustentar el acto de Recepción Profesional.

ATENTAMENTE



FEDERICO DEL RAZO LÓPEZ
DIRECTOR DE TESIS



ERÉNDIRA RENDÓN LARA
CODIRECTORA DE TESIS



ROBERTO ALEJO ELEUTERIO
REVISOR DE TESIS



ANNE LAURENT
REVISORA DE TESIS

ccp. Archivo
mcdh



2025
Año de
La Mujer
Indígena

Av. Tecnológico S/N, Col. Agrícola Bellavista, Metepec, Edo. de México,
C.P. 52149, Tels. Dirección: 7222087205, Conmut.: 7222087200
e-mail: info@toluca.tecnm.mx tecnm.mx | toluca.tecnm.mx



Agradecimientos

Agradezco en primer lugar a todas las personas que colaboraron directa o indirectamente en la realización de esta tesis. De manera especial, al Dr. Federico López del Razo, por su valiosa asesoría, por su guía académica y por el impulso constante que brindó a este proyecto desde sus primeras etapas.

Extiendo mi más sincero agradecimiento a los miembros del comité de tesis: a la Dra. Eréndira Rendón Lara, co-directora de este trabajo, por su acompañamiento y orientación; al Dr. Roberto Alejo Eleuterio, por su tutoría y apoyo durante el proceso formativo; y a la Dra. Anne Laurent, evaluadora externa, por su tiempo, disposición y valiosas observaciones que enriquecieron este proyecto.

Asimismo, agradezco al Ing. Abraham Flores y al Ing. Xanthe de la Rosa, por su colaboración en el desarrollo de herramientas de programación, fundamentales para el procesamiento de datos y la implementación del modelo propuesto.

Mi gratitud al Instituto Tecnológico de Toluca, por proporcionar el entorno académico y los recursos necesarios para llevar a cabo esta investigación. También reconozco el compromiso y conocimiento del cuerpo docente de la Maestría en Ciencias de la Ingeniería, cuya formación fue clave en este camino.

Finalmente, agradezco a la Secretaría de Ciencia, Humanidades, Tecnología e Innovación (SECIHTI), anteriormente conocida como CONAHCYT, por el apoyo financiero brindado mediante la beca de posgrado.

Resumen

Este trabajo presenta el desarrollo de un sistema basado en redes neuronales recurrentes tipo LSTM para la generación automática de secuencias armónicas en el género Rock-Pop. Se recopilaron y procesaron archivos MIDI para extraer pistas de guitarra y bajo, las cuales fueron transformadas a matrices piano roll en un rango de notas específico. Los datos fueron utilizados para entrenar una red LSTM con el objetivo de predecir acompañamientos armónicos a partir de secuencias musicales reales. Se aplicaron filtros basados en escalas musicales para delimitar la salida del modelo. El sistema fue evaluado comparando los resultados generados con fragmentos musicales reales, identificando estructuras repetitivas y coherencia tonal. La propuesta contribuye al campo de la inteligencia artificial aplicada a la composición musical, al ofrecer una herramienta que permite generar acompañamientos con estructura cíclica y consistencia armónica.

Abstract

This work presents the development of a system based on Long Short-Term Memory (LSTM) recurrent neural networks for the automatic generation of harmonic sequences in the Rock-Pop genre. MIDI files were collected and processed to extract guitar and bass tracks, which were transformed into piano roll matrices within a specific note range. The data were used to train an LSTM network with the goal of predicting harmonic accompaniments from real musical sequences. Harmonic filters based on musical scales were applied to constrain the model output. The system was evaluated by comparing the generated results with real musical fragments, identifying repetitive structures and tonal coherence. This approach contributes to the field of artificial intelligence applied to music composition by providing a tool capable of generating accompaniments with cyclic structure and harmonic consistency.

Índice general

1. Introducción	1
1.1. Planteamiento del problema	1
1.1.1. Definición del problema	2
1.1.2. Delimitación del problema	2
1.2. Objetivos	3
1.2.1. Objetivo general	3
1.2.2. Objetivos específicos	3
1.3. Hipótesis	4
1.4. Justificación	5
2. Estado del arte	8
3. Fundamentos	14
3.1. Teoría Musical	14
3.1.1. Notas musicales	14
3.2. Representación de Audio	25
3.2.1. Forma de Onda (<i>Waveform</i>)	25
3.2.2. Espectrograma	26
3.2.3. Cromagrama	26
3.2.4. Partituras musicales	27

3.2.5. Piano Roll	28
3.2.6. Representación digital de audio - Lenguaje MIDI	28
3.3. Inteligencia Artificial	32
3.3.1. Aprendizaje Automático (<i>ML, Machine Learning</i>)	37
3.3.2. Modelo de Aprendizaje Automático	48
3.3.3. Arquitecturas de redes neuronales artificiales	51
4. Metodología	56
4.1. Desarrollo para el modelo de Red Neuronal (LSTM)	56
4.1.1. Preparación del <i>Dataset</i> (Datos de entrada)	57
4.1.2. Definición de la arquitectura del modelo LSTM	63
4.1.3. Fase de entrenamiento del modelo	64
4.1.4. Fase de Generación	68
5. Resultados	72
5.1. Análisis musical mediante un DAW	72
5.2. Comparación de tonalidad entre el modelo y la plataforma AIVA	73
5.3. Transformación con la aplicación AIVA	76
5.4. Generación de atmósferas musicales	77
6. Conclusiones y recomendaciones	79
6.1. Conclusiones	79
6.2. Recomendaciones	80
Referencias	81
Apéndice A. Modelo de Red Neuronal LSTM para la generación de armonías musicales	
- Código	83
.1. Codificación de los archivos MIDI en Tensores - Código	83
.2. Proceso de padding para los archivos MIDI transformados a Tensores:	84

.3.	División del <i>Dataset</i> en lotes (<i>batches</i>) de entrenamiento y validación	85
.4.	Definición de la arquitectura del modelo LSTM para generación musical	86
.5.	Fase de Entrenamiento	92
Apéndice B. Comparación de tonalidades entre el modelo y AIVA		94
.1.	Comparación de tonalidades entre el modelo y AIVA - Guitarra	94
.2.	Comparación de tonalidades entre el modelo y AIVA - Bajo	97

Índice de figuras

3.1. Disposición de las notas en un piano	15
3.2. Notas graves y agudas	15
3.3. Notas enarmónicas, sostenidos y bemoles	16
3.4. Intervalo musical	17
3.5. Escala Mayor	18
3.6. Escala Menor	18
3.7. Proceso de armonización	19
3.8. Tipos de acordes	20
3.9. Notas y sus frecuencias	22
3.10 Grados Armonicos	24
3.11 Valores Rítmicos	24
3.12 Oda A La Alegría	24
3.13 Formas de onda	25
3.14 Espectrograma	26
3.15 Cromagrama	27
3.16 Partitura	27
3.17 Tablatura de guitarra	27
3.18 Cifrado Americano	28
3.19 Piano roll	28
3.20 Asignaición numérica de notas en MIDI	29
3.21 Representación del volumen en MIDI	29
3.22 Asignación de timbre en MIDI	30
3.23 Asignación de canales en MIDI	30
3.24 Byte de Estado	32
3.25 Codificación de bytes de estado	33
3.26 Mensaje MIDI	33

3.27Talos, robot de la antigua mitología griega	34
3.28Tesla mostrando el primer dron marino	34
3.29STAR, robot inteligente para cirugía	35
3.30Guitar Player, imagen creada mediante IA	36
3.31Comparación entre la programación clásica y el aprendizaje automático.	38
3.32Inspiración para crear el Perceptrón	39
3.33Gráfica de la función sigmoide	41
3.34Gráfica de la función Tangente Hiperbólica (TanH)	42
3.35Gráfica de la función ReLU	43
3.36Gráfica de la función Leaky ReLU	43
3.37Multi Perceptrón	45
3.38Proceso de optimización de un modelo de red neuronal	47
3.39Flujo del proceso de modelado de aprendizaje automático.	48
3.40Datos de entrenamiento y datos de validación	50
3.41Red de capas completamente conectadas (<i>FCN, Fully Connected Network</i>) . . .	51
3.42Diagrama de una capa convolucional.	53
3.43Cálculo de los estados ocultos en una RNN.	55
4.1. Fases del Modelo propuesto	57
4.2. Visualización de instrumentos/canales en Reason	59
4.3. Dos canales de guitarra en un mismo Archivo MIDI	60
4.4. Combinación de 2 canales de guitarra en un mismo canal	60
4.5. Transformación de archivo MIDI a Tensor	61
4.6. Proceso de padding	61
4.7. División del dataset en entrenamiento y validación	62
4.8. Representación esquemática de la arquitectura del modelo.	64
4.9. Primeros MIDIs generados	66
4.10Gráfica de pérdida durante el entrenamiento y validación del modelo	67
4.11Visualización de una secuencia generada después de aplicar el filtrado por escala tonal	68
4.12Estructura de cación en las secuencias generadas	70
4.13Comparación de similitud entre una canción generada por el modelo y las can- ciones reales del dataset	71
5.1. Secuencias musicales generadas con una estructura cíclica	73
5.2. AIVA y la detección de tonalidad de una canción	74

5.3. Distribución de tipos de equivalencia tonal entre el modelo y AIVA	75
5.4. Composiciones generadas en AIVA a partir de secuencias producidas por el modelo LSTM.	76
5.5. Sonorización de una película	78

Índice de tablas

5.1. Resumen porcentual de equivalencias tonales entre el modelo y AIVA	75
1. Comparación completa de tonalidades entre el modelo y AIVA	95
2. Comparación completa de tonalidades entre el modelo y AIVA (continuación) .	96
3. Comparación completa de tonalidades entre el modelo y AIVA - Bajo	97
4. Comparación completa de tonalidades entre el modelo y AIVA - Bajo (continuación)	98

Capítulo 1

Introducción

1.1. Planteamiento del problema

La composición musical puede surgir de diversas maneras, sin seguir un orden estricto. Puede empezar con una frase, una melodía o una secuencia de acordes. Sin embargo, la creación de secuencias armónicas o progresiones de acordes suele ser más desafiante, ya que requiere un conocimiento previo de teoría musical. Incluso cuando se dispone de una melodía o una letra, el siguiente paso, que es componer su armonía, demanda un entendimiento sólido de las reglas de acordes, tonalidades y secuencias armónicas. Es precisamente en este punto donde el proceso de composición se vuelve más complejo y técnico para músicos en formación o compositores emergentes.

Hasta la fecha, diversos estudios han explorado el uso de técnicas de inteligencia artificial (IA) para abordar el desafío de la composición musical, particularmente en la generación de armonías. Entre las técnicas más destacadas se encuentran las redes neuronales convolucionales, generativas adversarias y recurrentes. Estos modelos de IA buscan emular el proceso creativo humano al momento de crear una canción, utilizando principios estadísticos como iteraciones, probabilidad y muestreo para construir progresiones armónicas basadas en los datos musicales con los que fueron entrenados.

No obstante, aunque los resultados obtenidos mediante estos enfoques han mostrado momentos en los que los archivos de audio generados se perciben como coherentes y musicales, aún presentan desafíos. En muchos casos, se observan variaciones armónicas que carecen de sentido musical, afectando la coherencia global de las piezas generadas. Este problema

plantea la necesidad de seguir perfeccionando las técnicas de IA aplicadas a la composición musical para lograr progresiones armónicas que sean consistentes y estéticamente satisfactorias.

1.1.1. Definición del problema

Este proyecto se centrará en la generación automática de armonías musicales del género Rock-Pop, utilizando técnicas avanzadas de inteligencia artificial (IA). El objetivo es desarrollar una herramienta que sirva de apoyo tanto para la composición como para el estudio de la música, facilitando el proceso creativo y analítico de los músicos. Para lograr esto, se implementarán redes neuronales artificiales recurrentes tipo LSTM (*Long Short-Term Memory*), que son especialmente efectivas para trabajar con secuencias de datos, como es el caso de la música.

1.1.2. Delimitación del problema

La presente investigación se enfoca en el análisis y generación de música en formato MIDI, delimitada específicamente al género Rock-Pop. Esta delimitación se basa en dos factores principales: la disponibilidad de archivos MIDI de este género en diversas fuentes en línea y la facilidad que este formato ofrece para su manipulación técnica. Adicionalmente, se trabajará exclusivamente con pistas de los instrumentos musicales: bajo y guitarra. Ya que estos son los principales responsables de conducir la armonía en el Rock-Pop, permitiendo un análisis más enfocado en este aspecto musical.

Es importante destacar que los datos de entrada fueron preparados manualmente. En muchos de los archivos MIDI disponibles, algunas canciones contenían hasta tres pistas de guitarra. En estos casos, se procedió a combinar las pistas de guitarra en un solo canal, para asegurar la coherencia en la generación musical y evitar redundancias en el análisis.

Asimismo, se ha delimitado el estudio al género Rock-Pop debido al conocimiento musical previo que se posee, lo que garantiza un enfoque más especializado y preciso. Aunque otros géneros e instrumentos podrían haber sido considerados, la investigación se restringe a estos parámetros con el fin de profundizar en el análisis y asegurar la consistencia en el trabajo.

1.2. Objetivos

1.2.1. Objetivo general

Desarrollar un sistema basado en redes neuronales recurrentes para generar secuencias armónicas de guitarra y bajo en el género Rock-Pop. El objetivo es mejorar la creatividad y la eficiencia en el proceso compositivo, ofreciendo un recurso tanto para músicos experimentados como para estudiantes, al tiempo que se fomenta la innovación y la experimentación tecnológica en el ámbito musical, contribuyendo a la integración de la inteligencia artificial en el proceso creativo y artístico.

1.2.2. Objetivos específicos

- Diseñar e implementar un modelo de red neuronal generativa recurrente enfocado en la creación de armonías de guitarra y bajo para el género Rock-Pop.
- Recopilar, seleccionar y preparar una base de datos de canciones del género Rock-Pop en formato MIDI para ser utilizadas en el entrenamiento del modelo de red neuronal.
- Realizar el entrenamiento del modelo de red neuronal.
- Desarrollar un modelo de red neuronal, capaz de crear secuencias armónicas coherentes con el estilo del Rock-Pop.
- Evaluar los resultados generados por el modelo en función de criterios como coherencia musical, calidad armónica y adecuación al estilo Rock-Pop, comparándolos con secuencias armónicas reales.
- Explorar la escalabilidad del modelo y su potencial para adaptarse a otros géneros musicales o instrumentos, evaluando su capacidad para generar secuencias armónicas en distintos contextos musicales.
- Optimizar el modelo de red neuronal mediante la experimentación con distintas configuraciones de hiperparámetros y técnicas de regularización, con el objetivo de mejorar la calidad de las secuencias armónicas generadas.
- Documentar y divulgar los resultados obtenidos del proyecto en foros académicos y publicaciones científicas, contribuyendo al campo de la inteligencia artificial aplicada a la música.

1.3. Hipótesis

El uso de redes neuronales recurrentes tipo LSTM entrenadas con datos de canciones en formato MIDI del género Rock-Pop permitirá generar secuencias armónicas de guitarra y bajo que sean coherentes y estilísticamente consistentes con las progresiones armónicas típicas del género, mejorando así el proceso compositivo al ofrecer una herramienta automatizada que apoye la creatividad y eficiencia en la creación musical.

Evaluación Objetiva

- **Coherencia Armónica:** Analizar si las secuencias generadas siguen patrones tonales y progresiones típicas del Rock-Pop, comparando con canciones reales.
- **Evaluación de la eficiencia:** Comparar el tiempo de generación de secuencias armónicas por el modelo frente a la creación manual, midiendo la mejora en eficiencia.
- **Validación con aplicaciones que usan IA para la generación de música:** Utilizar aplicaciones ya desarrolladas de generación musical mediante IA para comparar los resultados generados por el modelo con otros sistemas existentes, midiendo la calidad y coherencia musical frente a herramientas comerciales

Evaluación Subjetiva

- **Juicio de expertos:** Músicos evaluarán las secuencias generadas en términos de coherencia musical, adecuación al estilo Rock-Pop, y calidad creativa.
- **Impacto en el proceso creativo:** Verificar si los músicos encuentran útiles las secuencias generadas para inspirar o complementar su proceso de composición.

1.4. Justificación

La presente tesis se justifica en la intersección de dos campos clave: el científico-tecnológico y el cultural-artístico. Por un lado, las técnicas de inteligencia artificial (IA), han demostrado su capacidad para modelar y generar secuencias musicales complejas, lo que las convierte en herramientas ideales para abordar el desafío de la generación automática de música. Por otro lado, la música, como una de las expresiones culturales más universales y significativas, ofrece un campo fértil para la aplicación de tecnologías emergentes, especialmente las redes neuronales.

Este proyecto busca unir el rigor científico con la creatividad artística, un enfoque que es impulsado y promovido por el **SECIHTI** (Secretaría de Ciencia, Humanidades, Tecnología e Innovación, antes CONAHCYT), que reconoce la importancia de integrar la ciencia con la cultura para el desarrollo integral de la sociedad.

Al desarrollar una herramienta capaz de generar progresiones armónicas musicales, se pretende facilitar el proceso de composición para músicos y estudiantes, permitiéndoles explorar nuevas ideas y enriquecer su trabajo. Así, no solo se impulsa la innovación tecnológica, sino que se aporta valor al ámbito cultural al poner la tecnología al servicio de la creatividad musical.

Además, en un contexto donde las tecnologías disruptivas están transformando múltiples disciplinas, esta investigación demuestra cómo es posible combinar el poder de la IA con el arte, proporcionando una herramienta que promueva tanto el estudio como la creación musical. En resumen, esta tesis contribuye a cerrar la brecha entre el desarrollo tecnológico y la exploración artística, reafirmando, como lo impulsa el SECIHTI, que la ciencia y la cultura pueden trabajar juntas para generar nuevo conocimiento y expandir los límites creativos.

El **Capítulo 1** presenta la introducción general del proyecto, enmarcando el problema que se busca resolver mediante la generación automática de secuencias armónicas utilizando redes neuronales recurrentes. Se expone cómo la automatización en la composición musical puede apoyar a músicos y productores en la creación de acompañamientos coherentes y estilísticamente adecuados, especialmente en géneros como el rock y el pop. El capítulo también establece los objetivos del trabajo, tanto generales como específicos, entre los que destaca el desarrollo de un modelo basado en LSTM capaz de generar progresiones de acordes a partir de datos musicales reales. Se formula una hipótesis relacionada con la viabilidad del uso de este tipo de redes para generar armonías con estructura musical reconocible. Finalmente, se justifica el valor del proyecto tanto desde una perspectiva tecnológica como artística, destacando su aporte al campo de la inteligencia artificial aplicada a la música.

El **Capítulo 2** presenta una revisión general de los principales métodos utilizados para la generación automática de música. Se describen diferentes enfoques desarrollados en investigaciones previas, incluyendo el uso de inteligencia artificial para crear melodías y acompañamientos. El capítulo destaca el uso de redes neuronales como una herramienta común en este campo y señala los avances recientes que han permitido generar música con mayor coherencia. También se mencionan algunas limitaciones encontradas en estudios anteriores, lo que justifica la propuesta de este trabajo.

El **Capítulo 3** presenta los fundamentos teóricos que sustentan el desarrollo del proyecto. En la primera parte se explican conceptos básicos de teoría musical, como escalas, acordes y progresiones armónicas, necesarios para entender la estructura de la música utilizada. La segunda parte introduce los principios generales de la inteligencia artificial y el funcionamiento de las redes neuronales, con énfasis en las redes recurrentes tipo LSTM, debido a su capacidad para trabajar con secuencias temporales como las que se encuentran en la música.

El **Capítulo 4** describe la metodología utilizada para desarrollar el modelo de red neuronal. Primero se detalla el proceso de preparación de los datos musicales, extrayendo y transformando archivos MIDI en un formato adecuado para el modelo. Luego se presenta la arquitectura del modelo LSTM, explicando su configuración básica. Posteriormente se expone la fase de entrenamiento, donde el modelo aprende a partir de ejemplos reales. Finalmente, se describe la fase de generación, en la que se crean nuevas secuencias musicales utilizando el modelo entrenado.

El **Capítulo 5** presenta los resultados obtenidos a partir del modelo entrenado. Se realizó un análisis musical de las secuencias generadas utilizando un entorno de producción musical (DAW), evaluando aspectos como estructura, repetición y coherencia tonal. También se comparó la tonalidad de los resultados del modelo con los generados por la plataforma AIVA, identificando similitudes y diferencias. Además, se experimentó con la transformación de los resultados utilizando AIVA, observando cómo estos se integraban en nuevas composiciones. Finalmente, se exploró la creación de atmósferas musicales, evaluando la calidad estética y expresiva del material generado.

El **Capítulo 6** presenta las conclusiones generales del trabajo, destacando que el modelo desarrollado fue capaz de generar acompañamientos armónicos con estructura y coherencia en el estilo Rock-Pop. Se comprobó que el uso de redes LSTM, junto con la preparación adecuada de los datos, permitió obtener resultados musicales funcionales. Además, se identificaron áreas de mejora para futuras investigaciones, como la integración de dinámicas, estructuras más complejas o elementos melódicos. Finalmente, se sugieren recomendaciones para extender

el modelo, evaluar otras arquitecturas y aplicar la metodología a diferentes géneros musicales.

Capítulo 2

Estado del arte

Actualmente existen diversas herramientas que generan música mediante Inteligencia Artificial, pero no logran tener aún una calidad que se compare a la que tiene una canción compuesta por un humano y grabada en un estudio profesional, por lo que aún se está lejos de poder hacer música de buena calidad en su totalidad con la IA disponible en este momento (año 2024) sin la necesidad de la intervención humana.

Algunos de los trabajos más sobresalientes con respecto al uso de redes neuronales y composición musical se citan a continuación:

Jukebox, es una aplicación creada por la compañía OpenAI, en 2019. Para entrenar a este modelo se utilizó un “dataset” (base de datos) de 1.2 millones de canciones, 600,000 de éstas en idioma inglés. Esta base de datos también incluye información adicional como la letra, artista, álbum, género y año de la canción, lo cual permite que la predicción de audio sea más apegada a un estilo o artista en particular. Este modelo también aprende como agrupar artistas y géneros similares. De acuerdo con la explicación dada por los ingenieros de OpenAI, primero se comprimieron las canciones con las que se entrenó al modelo, ya que se requería mucha memoria para procesar el “dataset” sin comprimir. Posteriormente, los datos se transformaron en imagen mediante redes neuronales convolucionales y se generaron nuevas imágenes condicionadas a la letra de las canciones, para después regresarlas al campo del audio. Los resultados encontrados fueron muy buenos, ya que podría decirse que las canciones generadas sí sonaron a música en algunos periodos de tiempo. Una característica muy interesante es que esta aplicación es libre y la puede utilizar cualquier persona, el usuario da “inputs” (datos de entrada) a la aplicación para generar una canción nueva. Estas entradas

pueden ser: artista, canción, letra y género de canciones ya existentes; y una vez ingresados estos datos, la aplicación crea un audio que trata de emular el estilo musical de las entradas dadas. (<https://openai.com/index/jukebox/>, 2019)

En el artículo *Counterpoint by Convolution* publicado en 2019 por Cheng-Zhi, Cooijmans, Roberts, Courville y Eck, se explica un modelo de redes neuronales convolucionales utilizado para la generación de un contrapunto musical. La principal ventaja del modelo convolucional utilizado en este proyecto, es que emula más el proceso creativo de un humano en comparación con otros modelos, ya que generalmente los modelos de Machine Learning para creación de música dividen la composición en un orden cronológico, generando una pieza paso a paso desde el inicio hasta el final. Por el contrario, un compositor humano escribe canciones en un proceso no lineal, probando diferentes ideas y motivos en diferentes partes de la canción, y a menudo retomando ideas previamente desechadas. Con el fin de tener una mejor aproximación a este proceso creativo, los autores entrenaron al algoritmo para crear bloques musicales y hacer combinaciones de dichos bloques utilizando una distribución estadística conocida como Gibb Sampling. De esta manera el modelo y el proceso generativo no estuvieron sujetos a una dirección en particular de composición. Los autores concluyeron que el uso del “sampleo Gibb” mejoró significativamente la calidad de las iteraciones a la hora de reescribir una porción de una canción, así mismo concluyeron que este método es más rápido y genera mejores resultados que el modelo “ancestral sampling”, también utilizado y evaluado por los autores. (Huang, Cooijmans, Roberts, Courville, y Eck, 2019)

El sistema Wavenet fue presentado por Van den Oord, Dieleman, Zen, Simonyan, Ninyals, Graves, Kalchbrenner, Senior y Kavukcuoglu en 2018. Se trata de un modelo basado en redes neuronales convolucionales que trabaja con imagen de onda de audio como entrada. En el artículo se expone que la arquitectura se probó para resolver cuatro problemas: reconocimiento del habla, texto a habla, generación multi hablante y, por último, generación musical. Su principal característica fue el uso de capas convolucionales causales con el fin de obtener percepción de patrones en “steps” (pasos) anteriores. Los autores comentan que en los experimentos probaron con dos “datasets” musicales. El primero de ellos formado por 200 horas de audio, dividida en clips de 29 segundos, los cuales eran etiquetados por género e instrumentos. El segundo “dataset” estuvo formado por 60 horas de música de piano. Los resultados fueron coherentes armónicamente, pero fallaban en conseguir reproducir patrones a lo largo del tiempo, cambiando de género, volumen y calidad a los pocos segundos. (Aäron

van den Oord, 2016)

En 2018, Lim, Rhyu y Lee propusieron un sistema basado en LSTM bidireccionales (Long Short-Term Memory, por sus siglas en inglés), para generar acompañamiento musical en forma de una progresión de acordes dada una melodía. Esta red fue entrenada con un corpus de 2252 partituras de música moderna (rock, jazz, pop, etc.), resultando en un total de 1802 canciones. Para representar las notas de la melodía, se tuvieron en cuenta 12 posibles valores (no se contó con ninguna octava extra) y para los acordes, únicamente se utilizaron triadas (acordes de tres notas). El entrenamiento se llevó a cabo presentando a la red varios grupos de cuatro tiempos de melodía y comparando a la salida los acordes correspondientes. En la fase de generación, iterativamente se pasó a la red bloques de cuatro tiempos, conectando los acordes de salida hasta tener la progresión completa. Los autores compararon el sistema con otros modelos y lo evaluaron cuantitativamente (con matrices de confusión) y cualitativamente (con 25 oyentes). Los resultados mostraron mayor precisión que con los otros modelos, así como una tendencia por parte de los usuarios a elegir estas composiciones frente a las de los otros modelos. (Lim H., 2017)

El libro publicado en 2018 por Briot, Hadjeres y Pachet titulado “Deep learning techniques for music generation – a Survey” es un compilado de experimentos y análisis del uso de Deep Learning (Deep artificial neural networks) para la generación musical. Este estudio dividió las composiciones que se deseaban generar de la siguiente manera: melodía, polifonía, acompañamiento y/o contrapunto, además de que considera dos vertientes, si la composición generada ha de ser interpretada por un humano (grabación) o por una máquina (archivo de audio). Asimismo, los experimentos que se registraron en esta bibliografía incluyeron el uso de varios tipos de redes neuronales, entre las que destacan: feedforward network, recurrent network y generative adversarial network. Dependiendo el tipo de red neuronal, se hizo uso de datos de entrada de diferente tipo, entre otros: señales de audio (waveform), señales de audio transformadas a imagen de la forma espectrograma, y representación de audio en estilo chromagrama. Esta referencia también menciona las limitaciones y/o complicaciones de cada uno de los experimentos que se llevaron a cabo y como lo solucionaron, detallando todas las posibilidades evaluadas y dando conclusiones de por qué se escogió la solución dada. Por ejemplo, cómo representar la duración de dos notas que tienen ligadura, para formatos MIDI y Piano roll. Finalmente, el artículo ofrece una opinión sobre la evaluación de los resultados arrojados por el modelo, explicando que todas las composiciones creadas con IA

fueron dependientes de los datos de entrada, por lo que la apreciación musical será subjetiva, y entre más distantes estén los datos de entrada más difícil será juzgar el género musical obtenido; la ventaja es que se podrían encontrar resultados más creativos desde el punto de vista musical. (Jean-Pierre Briot, 2019)

Chaun y Herremans propusieron en 2018 un modelo para generar música polifónica mediante representaciones gráficas. En este trabajo el audio fue representado en una imagen 2D utilizando varias herramientas de teoría musical. El modelo es de tipo predictivo y se utilizaron redes neuronales convolucionales con celdas Long Short-Term Memory (LSTM, por sus siglas en inglés), para generar secuencias temporales de audio. Para esta red neuronal se utilizaron 3 variantes: LSTM sin pre-entrenamiento, LSTM con pre-entrenamiento y representación de música gráficamente en una matriz tipo Tonnetz para música polifónica. Los resultados experimentales mostraron que la representación musical en la matriz de Tonnetz producía secuencias tonales más estables y tuvo más patrones repetitivos que las secuencias obtenidas mediante representaciones en formato Piano roll, lo cual fue útil para abordar los retos actuales en música e IA. (Chuan C.-H., 2018)

La composición musical mediante inteligencia artificial (IA) ha avanzado significativamente. A continuación, se presenta un resumen del estado del arte.

■ **Generación de Melodías y Armonías:**

Los modelos de aprendizaje profundo, como las redes neuronales recurrentes (RNN) y las redes generativas adversarias (GAN), han demostrado ser eficaces en la generación de melodías y armonías musicales. Algunas plataformas y algoritmos permiten a los usuarios interactuar con modelos pre-entrenados para componer música personalizada.

■ **Estilo y Emulación:**

Se han desarrollado modelos capaces de emular estilos musicales específicos, incluso imitando a compositores famosos. Esto se logra mediante el entrenamiento de modelos con grandes conjuntos de datos de composiciones existentes.

■ **Colaboración con Compositores Humanos:**

Muchos proyectos exploran la colaboración entre compositores humanos e IA, donde la IA actúa como una herramienta creativa que sugiere ideas, arreglos o incluso completa composiciones basándose en la entrada del compositor humano.

- **Transformación y Remezcla:**

Algoritmos de IA permiten la transformación y remezcla de composiciones existentes. Esto incluye la capacidad de cambiar estilos, instrumentación o incluso combinar elementos de diferentes piezas musicales.

- **Avances en Procesamiento del Lenguaje Natural (PLN) Musical:**

Se ha trabajado en el desarrollo de modelos de PLN específicos para música, permitiendo a los sistemas comprender y generar música basándose en descripciones textuales o conceptos abstractos.

- **Uso de Redes Neuronales Recurrentes a Largo Plazo (LSTM) y Transformers:**

Modelos basados en arquitecturas LSTM y Transformers han ganado popularidad debido a su capacidad para capturar patrones temporales complejos y relaciones a largo plazo en la música.

- **Énfasis en la Creatividad y Expresividad:**

Se ha prestado atención a mejorar la creatividad y la expresividad de las composiciones generadas por IA. Esto implica la incorporación de elementos emotivos y subjetivos en la música generada.

- **Desafíos Éticos y Derechos de Autor:**

A medida que la IA se vuelve más competente en la composición musical, surgen desafíos éticos, especialmente en términos de derechos de autor y la atribución de obras generadas por máquinas.

Aplicaciones para la generación de música mediante IA

- **Amper Music**

Amper Music utiliza inteligencia artificial para crear música personalizada en función de las preferencias y requisitos del usuario. Puede generar música para diversos propósitos, como bandas sonoras de películas, anuncios y más.

- **Jukedeck**

Jukedeck es una plataforma que utiliza algoritmos de aprendizaje automático para componer música original. Permite a los usuarios personalizar la música según sus necesidades, como la duración, el estado de ánimo y el estilo.

- **AIVA**

AIVA (Artificial Intelligence Virtual Artist) es una plataforma que emplea la inteligencia artificial para componer música clásica. Puede generar partituras completas y ha sido utilizada en colaboración con compositores humanos.

- **Google Magenta Studio**

Magenta Studio es un conjunto de herramientas de Google que utiliza IA para la creación musical. Ofrece funciones como la generación de melodías y acordes, así como la experimentación con modelos de aprendizaje profundo.

- **IBM Watson Beat**

Watson Beat, parte de IBM Watson, utiliza inteligencia artificial para crear música en función de diversos parámetros, como el estado de ánimo y la emoción. Puede generar composiciones originales adaptadas a las necesidades del usuario.

- **Endel**

Endel utiliza algoritmos de IA para generar música personalizada en tiempo real, adaptándose a factores como la ubicación, el clima y el momento del día. Está diseñada para mejorar la concentración, el sueño y la relajación.

- **WaveAI - Alysia**

Alysia es una aplicación que utiliza IA para asistir en la composición musical. Puede generar acompañamientos musicales basados en la voz del usuario y ofrecer sugerencias creativas.

Capítulo 3

Fundamentos

3.1. Teoría Musical

Comprender la teoría musical es comparado con una alfabetización. Sin este conocimiento, interpretar música sería como intentar leer un libro únicamente a través de sus imágenes: se podría captar la esencia de la historia, pero se perderían las sutilezas y complejidades que le dan profundidad. Aunque la música es un arte y una forma poderosa de expresión, esto no implica que no pueda abordarse con formalidad técnica. De hecho, la teoría musical proporciona un marco técnico que permite analizar, interpretar y crear música de manera estructurada. Adquirir una base sólida de teoría musical no solo resulta esencial para alcanzar la alfabetización musical, sino también para aprovechar al máximo las herramientas técnicas que este arte ofrece. (Tellier, 2024)

En este capítulo, se explicarán las partes esenciales de la teoría musical que permitirán comprender lo que es una armonía y su papel fundamental en la composición musical.

3.1.1. Notas musicales

En la música occidental, un concepto clave es que existen únicamente 12 notas distintas que constituyen la base de toda creación musical. Estas notas son el fundamento de cualquier sonido que se escuche en una canción, independientemente del género al que pertenezca. Ya sea hip-hop, jazz, pop, música clásica u otro estilo, todos los sonidos y melodías provienen

exclusivamente de estas 12 notas.

En el teclado de un piano es fácil observar este patrón de 12 notas repetirse de manera uniforme a lo largo de todo el teclado, al conjunto de estas doce notas se le conoce comúnmente como una **octava**. Este diseño permite identificar de forma visual y práctica cada octava, de otra manera sería mucho más complejo tocar el piano. En la Figura 3.1 se puede apreciar claramente este patrón distintivo.

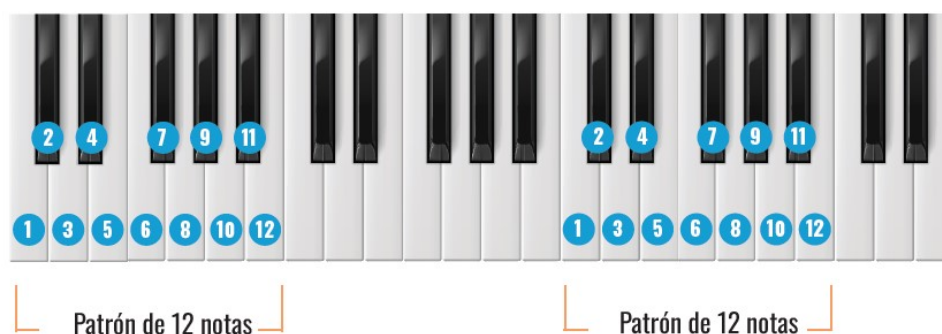


Figura 3.1: Disposición de las notas en un piano. Se puede observar cómo el teclado de un piano sigue un patrón cíclico de 12 notas, que se repite a lo largo de su extensión.

Nombre de las notas

Ahora que hemos identificado el patrón de las 12 notas en un piano, es momento de asignarles nombres para facilitar su memorización. Comenzaremos con las teclas blancas, que representan las **notas naturales**: Do, Re, Mi, Fa, Sol, La y Si. Figura 3.2

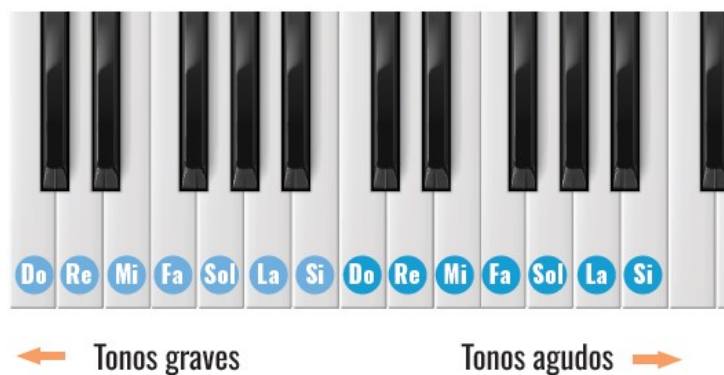


Figura 3.2: A medida que se desciende, las notas suenan más graves, y a medida que se asciende, los tonos se vuelven más agudos, pero siguen siendo las mismas notas.

Ahora vamos a explorar los nombres de las teclas negras en el teclado, las cuales pueden considerarse como modificaciones o variaciones de las teclas blancas adyacentes, representando versiones de tono más alto o más bajo de las notas naturales (teclas blancas) (Tellier, 2024). Moverse hacia la derecha desde una tecla blanca hacia la tecla negra siguiente agudiza la nota, por ejemplo, de Do a Do# (Do sostenido), elevando su tono. Moverse hacia la izquierda desde una tecla blanca hacia la siguiente tecla negra hace mas grave la nota, por ejemplo, de Re a Reb (Re bemol), disminuyendo su tono. La tecla negra entre Do y Re puede ser llamada Do# o Reb, las cuales son llamadas *equivalentes enarmónicos* que representan la misma nota.

1



Figura 3.3: Las teclas negras pueden llamarse Do#, Re#, Fa#, Sol# y La# cuando la tecla negra está situada a la derecha de la tecla blanca o pueden denominarse Sib, Lab, Solb, Mib y Reb cuando están a la izquierda de la tecla blanca más cercana.

En la Figura 3.3 también puede apreciarse que no hay una tecla negra entre las notas Mi y Fa ni tampoco la hay entre las notas Si y Do. Por lo que un Mi# es igual a un Fa y un Si# es igual a un Do, análogamente un Dob es igual a un Si y un Fab es igual a un Mi.

Intervalos Musicales

Se define como la distancia (musicalmente hablando) que existe entre una nota y otra. Hay diversas maneras de expresar esta distancia, una de las más usadas es el método de tonos y semitonos. Un semitono es la distancia musical que hay entre una tecla (blanca o negra)

¹El símbolo " #" se conoce como **sostenido** y representa un aumento en el tono de una nota. Por otro lado, el símbolo *b* se denomina **bemol** e indica una disminución en el tono de una nota. Estos símbolos se utilizan para nombrar las teclas negras en relación con las teclas blancas.

y la siguiente en el piano ya sea hacia lo agudo o lo grave, mientras que un tono equivale a dos semitonos. Por ejemplo, entre Do y Re hay dos semitonos de distancia (lo que equivale a un tono): de Do a Do# es un medio tono y de Do# a Re es otro medio tono. En la Figura 3.4, se muestra cómo de un Do al Do agudo siguiente hay una distancia de 12 semitonos (una octava) o 6 tonos, de un Re al siguiente Fa más agudo hay 3 semitonos o 1.5 tonos de distancia y, finalmente, de un Si al siguiente Sol más grave hay 4 semitonos o 2 tonos de distancia.

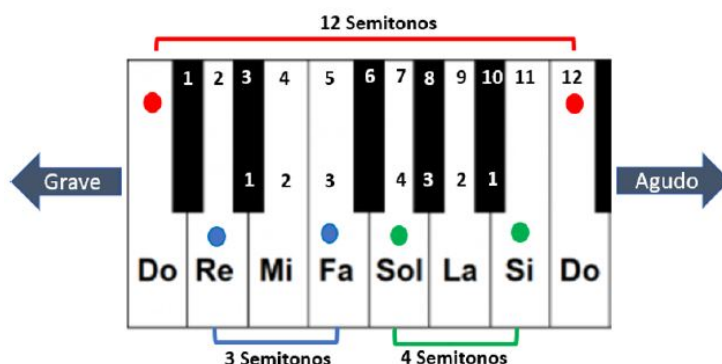


Figura 3.4: Intervalo Musical: Un tono es igual a 2 semitonos, por lo que de Do a Do hay 12 semitonos ó 6 tonos, de Re a Fa 3 semitonos ó 1.5 tonos y de Si a Sol 4 semitonos ó 2 tonos

Escalas Musicales

Para entender el concepto de escalas, podemos hacer una comparación con el lenguaje. Al hablar, seleccionamos palabras que están formadas por letras del alfabeto. De manera similar, en la música, las escalas actúan como un alfabeto, ya que establecen las notas disponibles para construir melodías y armonías, de la misma forma que se eligen letras para crear palabras. Por ejemplo, la escala de Do mayor está formada por las notas Do, Re, Mi, Fa, Sol, La y Si (que coinciden con las teclas blancas del piano), por lo tanto, al componer una canción en la tonalidad de Do Mayor, se utilizan exclusivamente estas notas para construir tanto melodías como armonías, asegurando que la música permanezca dentro de la misma tonalidad. De manera general, las notas que conforman una escala mayor están determinadas por una secuencia de intervalos específicos, compuesta por tonos (T) y semitonos (ST), la cual se repite cíclicamente: $T - T - ST - T - T - T - ST$.

De tal manera que para construir cualquier escala mayor partimos de la nota inicial (que es la que le dará el nombre a la escala) y las demás notas de la escala estarán determinadas por la

secuencia definida anteriormente. La figura 3.6 muestra como se construye la escala de Do mayor y Re mayor.

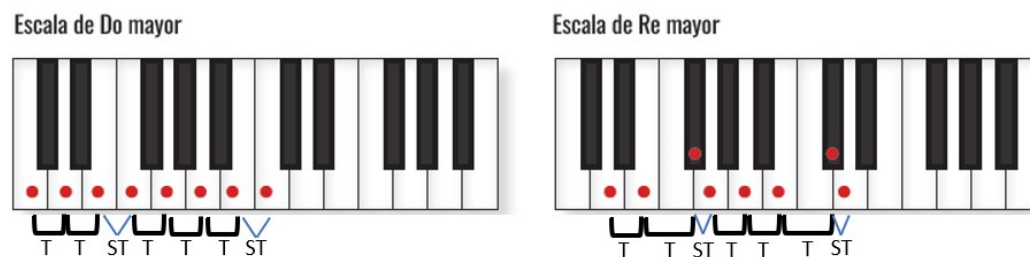


Figura 3.5: Construcción de la escala mayor por tonos y semitonos, es importante recordar que cada tono (T) representa un salto de dos teclas mientras que un semitono (ST) es un salto de una tecla en el piano

Análogamente, existe un patrón de intervalos que determina las notas en una escala menor ²:

$T - ST - T - T - ST - T - T$

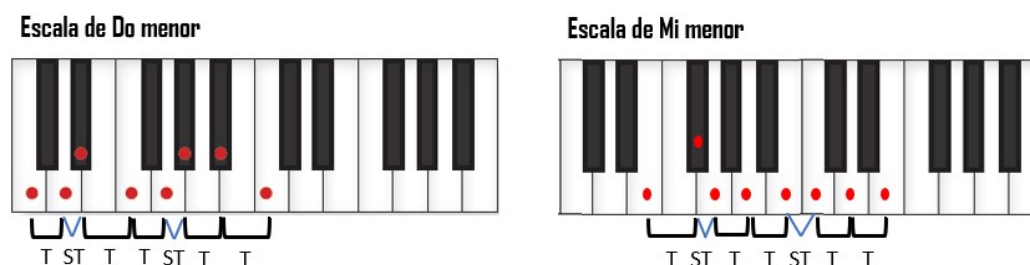


Figura 3.6: Construcción de la escala menor por tonos y semitonos, es importante recordar que cada tono (T) representa un salto de dos teclas mientras que un semitono (ST) es un salto de una tecla en el piano

Existe una inmensa variedad de escalas, cada una con su propio carácter y color tonal. Aunque las escalas mayores y menores son la base de la música occidental y predominan en géneros como el rock y el pop, existen muchas otras escalas que enriquecen el panorama musical, como las modales, pentatónicas, armónicas y exóticas de diferentes culturas. Explorar estas escalas no solo aporta diversidad estilística, sino que también amplía las posibilidades creativas. Para los compositores musicales, comprender y dominar tanto las escalas tradicionales como las menos convencionales es esencial, ya que brinda una estructura sólida para componer y producir música.

²Existen tres tipos de escalas menores: menor natural, menor armónica y menor melódica. La escala menor natural es la más común en la música del género Rock-Pop, por lo que nos centraremos en ésta

Armonización de una escala musical

La armonización de una escala musical es el proceso de construir **acordes** con las notas disponibles en una escala dada. Por definición, un acorde es un conjunto de al menos tres notas tocadas al unísono.

Para construir un acorde, tomamos cualquier nota de la escala, la cuál será la primer nota del acorde y se le conoce como *tónica*, después le agregamos dos notas más siguiendo un patrón específico:

- Primer nota: Es la nota tónica y es la que le dará el nombre al acorde.
- Segunda nota: Contando desde la tónica, es la tercer nota subsecuente de la escala.
- Tercer nota: Contando desde la tónica, es la quinta nota subsecuente de la escala.

Por ejemplo, considérese la escala de Do mayor que contiene las notas Do, Re, Mi, Fa, Sol, La y Si. Para armonizar la escala, se toma cada una de las 7 notas de la escala y se construyen los acordes sobre cada nota como se describió anteriormente, la Figura 3.7 ejemplifica la construcción de los acordes de Do, Re y Si.

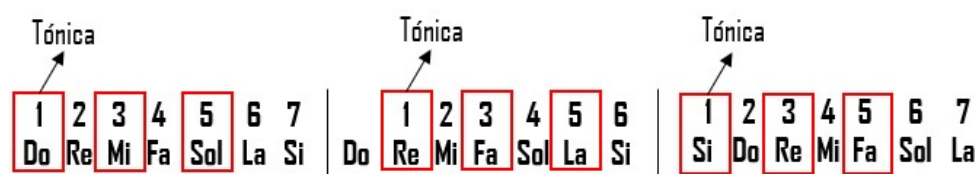


Figura 3.7: Proceso de armonización de la escala de Do mayor para las notas Do, Re y Si

Estos acordes resultantes se clasifican en 3 tipos, **Mayores, Menores y Disminuidos**, dependiendo de la distancia o intervalo que existe entre las notas que los conforman:

- Acorde Mayor: Se compone de la nota tónica, la segunda nota está ubicada a cuatro semitonos y la tercera nota se ubica a 7 semitonos.
- Acorde Menor: Se compone de la nota tónica, la segunda nota está ubicada a tres semitonos y la tercera nota se ubica a 7 semitonos.
- Acorde Disminuido: Se compone de la nota tónica, la segunda nota está ubicada a tres semitonos y la tercera nota se ubica a 6 semitonos.

La figura 3.8 muestra los acordes ya clasificados como Do Mayor, Re Menor y Si Disminuido, de acuerdo con lo explicado en el párrafo anterior.

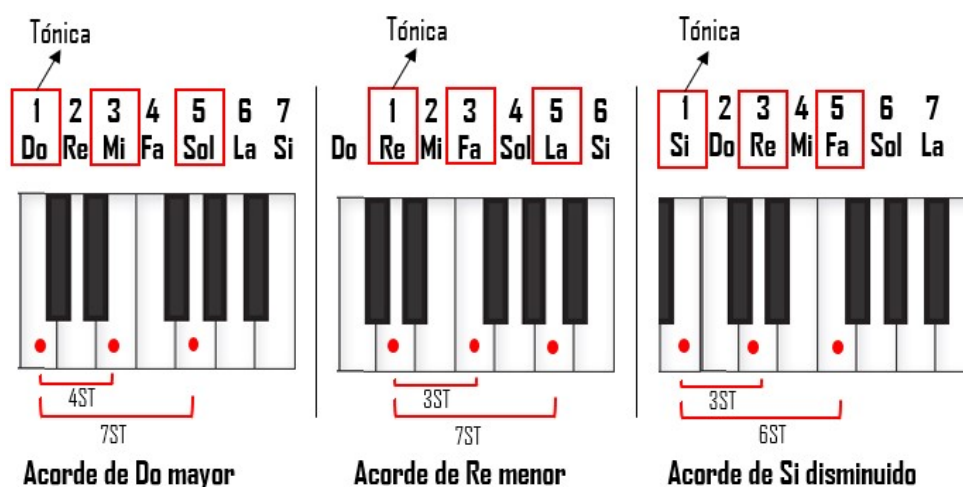


Figura 3.8: Do Mayor: De Do a Mi hay 4ST y de DO a Sol 7ST; Re menor: De Re a Fa hay 3ST y de Re a La 7ST; Si Disminuido: De Si a Re hay 3ST y de Si a Fa 6ST

De manera general, cuando se armoniza una escala mayor siempre se van a obtener 7 acordes (un acorde por cada nota de la escala), comúnmente en la teoría musical, estos acordes se enumeran con números romanos (ejemplo: Grado I, denota el acorde de la primera nota de la escala) y quedan ordenados de la siguiente manera:

Grado	Ejemplo en Do mayor	Tipo de acorde
I	C - E - G	Mayor
ii	D - F - A	Menor
iii	E - G - B	Menor
IV	F - A - C	Mayor
V	G - B - D	Mayor
vi	A - C - E	Menor
vii°	B - D - F	Disminuido

Una vez explicado el proceso de armonización para escalas mayores, se puede replicar en las escalas menores naturales y se obtendrán los siguientes acordes:

Grado	Ejemplo en La menor	Tipo de acorde
i	A - C - E	Menor
ii°	B - D - F	Disminuido
III	C - E - G	Mayor
iv	D - F - A	Menor
v	E - G - B	Menor
VI	F - A - C	Mayor
VII	G - B - D	Mayor

Canción

Una canción es una composición musical que combina melodía, armonía y letra para transmitir emociones, contar una historia o expresar ideas. Las canciones pueden variar ampliamente en estilo, género y estructura, pero generalmente comparten ciertas partes esenciales que contribuyen a su cohesión y efectividad. Estas partes pueden variar en presencia y orden dependiendo del género musical y la intención del compositor, pero juntas forman la estructura básica de muchas canciones populares:

- **Introducción:** La sección inicial de la canción que prepara al oyente para lo que vendrá. Establece el tono y el tempo, puede ser instrumental o incluir letras.
- **Verso:** Secciones narrativas que desarrollan la historia o tema de la canción. Proporciona detalles y contexto; cada verso suele tener letras diferentes pero una melodía similar.
- **Estribillo:** La parte más pegajosa y repetitiva de la canción. Resume el tema principal; es memorable y suele tener la misma letra y melodía cada vez que aparece.
- **Puente:** Una sección contrastante que introduce una nueva idea o emoción. Aporta variedad y evita la monotonía; a menudo conduce a un clímax
- **Pre-estribillo:** Una sección que precede al estribillo y aumenta la anticipación. Sirve como transición entre el verso y el estribillo, a menudo elevando la energía.
- **Coro:** Una frase o sección extremadamente pegajosa que captura la esencia de la canción. Es la parte más memorable; puede ser parte del estribillo o aparecer en otros lugares.

- Coda: La sección final de la canción que cierra la pieza. Proporciona una conclusión; puede ser una repetición del estribillo, una sección instrumental, o algo completamente nuevo.

Armonía

La armonía es el estudio de los acordes y como se ejecutan, por ejemplo, si tomamos los acordes de una canción y los ejecutamos en un orden diferente la sensación de la canción original cambiará, por ende, es importante comprender bien la armonía y las reglas que la teoría dicta, para saber que acordes debemos elegir y como los debemos ejecutar (en que orden) al momento de hacer una canción. La armonía, constituye la base de toda canción y sobre esta base se construye la melodía y demás elementos decorativos que pudiese llegar a tener la canción.

Frecuencia

En música la frecuencia se refiere a cuantas veces se repite el patrón de vibración de una onda sonora en un punto dado, a mayor frecuencia, más aguda y entre menor frecuencia más grave será la nota que se percibe. A continuación en la Figura 3.9, se muestra el valor en frecuencia de cada nota correspondiente a la cuarta octava de un piano y como se ven gráficamente tres sonidos de diferente frecuencia:

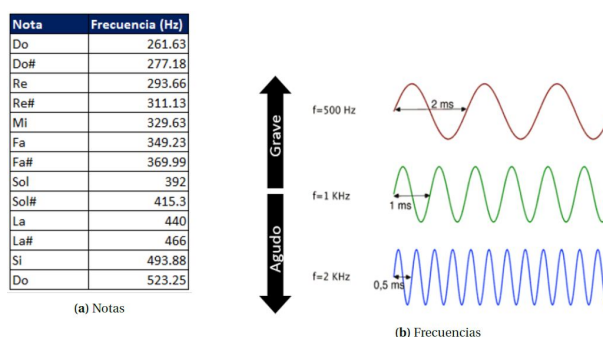


Figura 3.9: Notas y sus frecuencias

Armonía de una canción

La armonía, constituye la base de toda canción y sobre esta base se construye la melodía y demás elementos decorativos que pudiese llegar a tener la canción. En armonía es muy común utilizar letras para referirnos a acordes, a saber: C → Do, D → Re, E → Mi, F → Fa, G → Sol, A → La y B → Si. Así mismo, cuando tenemos una “m” delante de una letra, esto significa que el acorde es menor, por ejemplo Dm → Re menor, cuando encontramos un “#” delante de una letra, esto significa que el acorde es sostenido, por ejemplo F# → Fa sostenido, y finalmente cuando hay un “b”, significa bemol: Eb → Mi bemol. A manera de ejemplo, en una canción existe una serie de acordes los cuales conforman la armonía, a esta sucesión de acordes se le conoce como sucesión armónica y se representa con números romanos, donde el I corresponde al acorde de la tonalidad en la que se encuentra la canción y los demás acordes en la sucesión armónica recibirán el número dependiendo a que distancia este dicho acorde del acorde I.

Como puede verse en la tabla de la Figura 3.10, cada acorde tiene 7 grados que se expresan en números romanos, así el primer grado corresponde al número I, el segundo grado corresponde al número II y así sucesivamente hasta llegar al VII. El grado VIII vuelve a ser el Grado I pero una octava más arriba, por ende se dice que un acorde solo tiene 7 grados. Existen sucesiones armónicas que han sido muy utilizadas desde la música clásica hasta la actualidad y que generan la misma sensación auditiva desde el punto de vista armónico, por ejemplo, la sucesión armónica I-IV-VII-III-VI-II-V se ha utilizado en las siguientes composiciones:

- Sonata No 9 en LA de Mozart
- Segundo concierto de Brandemburgo de Bach
- L'Inverno de Vivaldi
- I Will Survive de Gloria Gaynor
- Fly to the moon de Frank Sinatra

Existen muchos ejemplos similares con otras sucesiones armónicas que han sido utilizadas a través de la historia en diversas composiciones, las cuales se van a analizar y se utilizarán para la fase de entrenamiento de nuestro modelo.

TÓNICA	I	II	III	IV	V	VI	VII
C (Do)	C	Dm	Em	F	G	Am	Bb
D (Re)	D	Em	F#m	G	A	Bm	C
E (Mi)	E	F#m	G#m	A	B	C#m	D
F (Fa)	F	Gm	Am	Bb	C	Dm	Eb
G (Sol)	G	Am	Bm	C	D	Em	F
A (La)	A	Bm	C#m	D	E	F#m	G
B (Si)	B	C#m	D#m	E	F#	G#m	A

Figura 3.10: Grados Armonicos

Ritmo

En música el ritmo es una secuencia de sonidos y silencios que tiene un patrón o secuencia repetitiva y que se expresa en un lenguaje escrito, el cual se muestra en la Figura 3.11:

<u>FIGURA RÍTMICA</u>	<u>SILENCIO</u>	<u>NOMBRE</u>	<u>DURACIÓN</u>
		Redonda	Cuatro pulsos
		Blanca	Dos pulsos
		Negra	Un pulso
		Corchea (Un corchete).	Medio Pulso
		Semicorchea (Dos corchetes).	Un cuarto de pulso

Figura 3.11: Valores Rítmicos

Mediante estas figuras se puede determinar el ritmo de una canción, ya en uso el patrón rítmico en una partitura luce como en la Figura 3.12:



Figura 3.12: Oda A La Alegría

Melodía

La melodía es la ejecución de un conjunto de notas que se tocan una después de la otra con un cierto ritmo definido. Y es la idea principal de la canción, por ejemplo, en la música rock pop contemporánea generalmente la melodía es la que se interpreta con la voz. Suele también decirse que cuando se tararea una canción, la melodía es aquello que se está tarareando. También puede decirse que la melodía es lo que recordamos cuando tratamos de reproducir una canción en nuestra mente.

3.2. Representación de Audio

La representación es el tipo de datos que se utilizará para entrenar y generar contenido musical, por ejemplo; señales de audio, señales transformadas (espectogramas, transformada de Fourier), piano roll, MIDI, tablaturas o incluso texto. La elección del tipo de representación y su codificación está estrechamente ligado a la configuración de la entrada y la salida de la arquitectura de la red neuronal, además tiene una gran influencia en la exactitud del proceso de aprendizaje y de la calidad del contenido generado.

3.2.1. Forma de Onda (Waveform)

La representación más directa de una señal de audio es la forma de onda. La visualización de una señal de onda original se muestra en el inciso a) de la Figura 3.13, otra manera de visualizar la onda con una resolución más fina se muestra en el inciso b). En ambas figuras, el eje "X" representa el tiempo y el eje "Y" representa la amplitud de la señal.

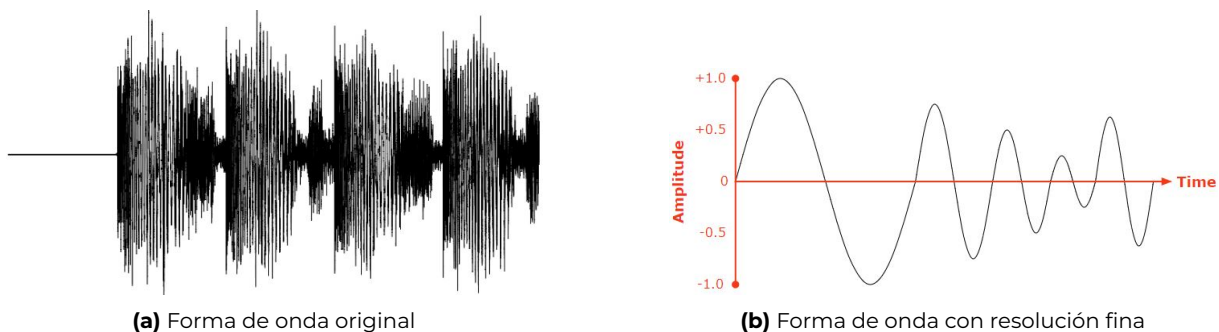


Figura 3.13: Formas de onda

La ventaja de usar la forma de onda en la representación de audio es que no tiene ninguna modificación o pérdida de transformación, por lo que se trabaja con la resolución original. La desventaja es que el tratamiento de este tipo de formato demanda mucha memoria y procesamiento computacional.

3.2.2. Espectrograma

El espectrograma es una representación de audio que se obtiene a través de la transformada de Fourier, la Figura 3.14 muestra una imagen del espectro de audio. El eje "X" representa el tiempo en segundos, el eje "Y" representa la frecuencia in KHz y la escala de color a la derecha representa la intensidad del sonido (volumen) medida en dBFS.

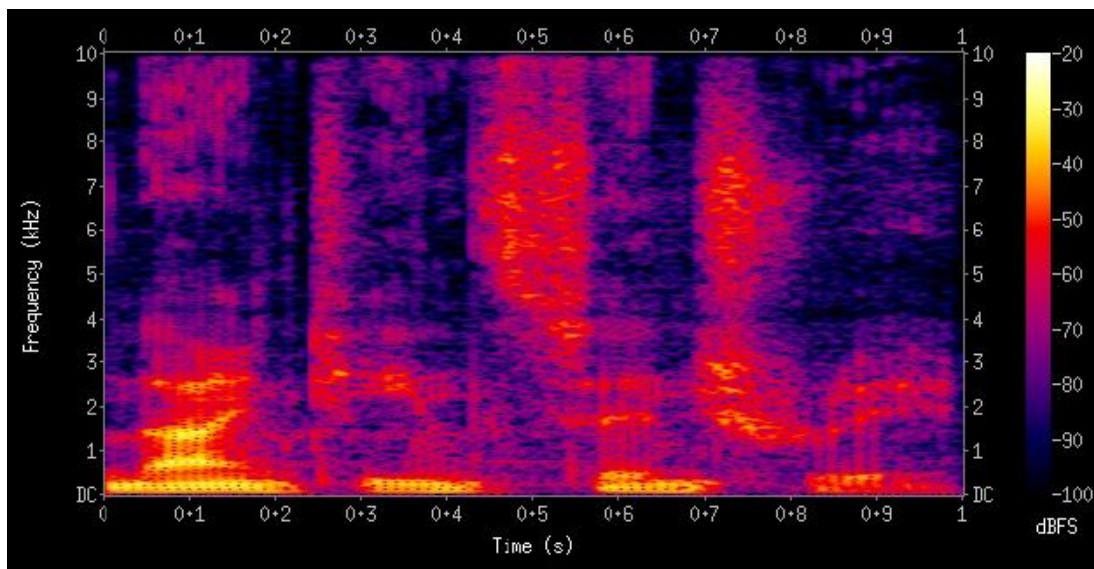


Figura 3.14: Espectrograma

3.2.3. Cromagrama

El término *croma* se refiere a las notas musicales cromáticas, es decir, todas las notas presentes en una octava. Un cromagrama es útil para analizar la tonalidad, los acordes y otros aspectos relacionados con la distribución de las frecuencias en la música. El cromagrama suele representarse como un gráfico en el que el eje horizontal representa el tiempo y el eje vertical representa las diferentes notas musicales. Cada nota se asocia con un color o un valor numérico, y la presencia de esa nota en un momento específico se indica mediante la intensidad del color o el valor numérico en el cromagrama, Figura 3.15

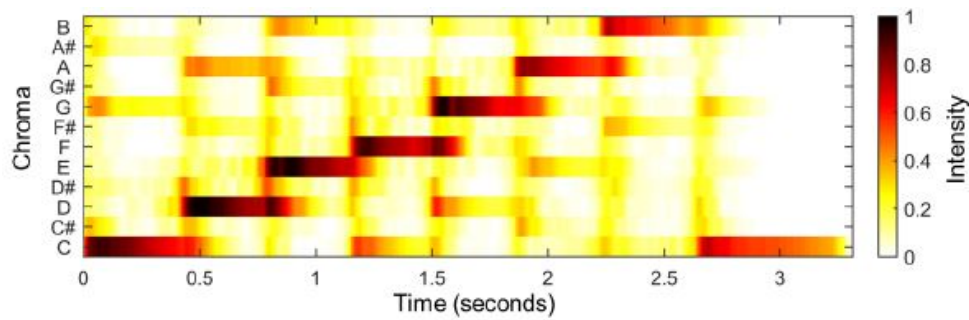


Figura 3.15: Cromagrama

3.2.4. Partituras musicales

Una partitura es una representación gráfica de una composición musical que utiliza símbolos para indicar la altura tonal, la duración y otras características. Las notas, las claves, las líneas y otros elementos en la partitura son símbolos que representan la información musical. Figura 3.16



Figura 3.16: Partitura

Tablaturas de guitarra

En lugar de utilizar notación musical tradicional, las tablaturas de guitarra representan la disposición de las cuerdas y los trastes de una guitarra, indicando dónde colocar los dedos para producir ciertos sonidos. Figura 3.17

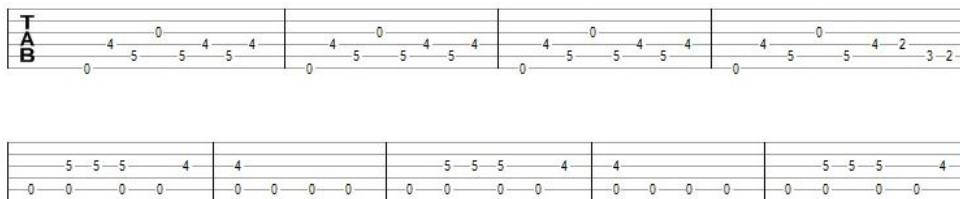


Figura 3.17: Tablatura de guitarra

Cifrado Americano

Es un sistema de notación musical que utiliza letras para referirse a las notas. Figura 3.18

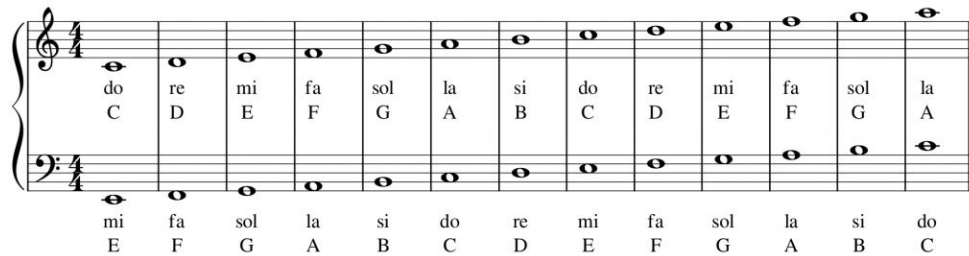


Figura 3.18: Cifrado Americano

3.2.5. Piano Roll

Es una representación visual de la duración y altura de las notas musicales a lo largo del tiempo. Las notas se muestran como rectángulos horizontales que representan su duración, y su posición vertical indica la altura tonal (la nota musical). La disposición de estas notas en el tiempo forma la secuencia musical. Figura 3.19



Figura 3.19: Piano roll

3.2.6. Representación digital de audio - Lenguaje MIDI

MIDI (Musical Instrument Digital Interface) es un protocolo de comunicación binario/hexadecimal que sirve como la interface entre instrumentos musicales y la computadora. Este protocolo permite visualizar en tiempo real el audio proveniente de los instrumentos musicales, así como tener una grabación digital del audio generado y su posterior modificación. Para poder entender cómo funciona el lenguaje MIDI, primero se tienen que definir los siguientes

parámetros:

Tono

El tono (*pitch*) es la nota que se está ejecutando. MIDI asigna valores desde 0 hasta 127 para cada nota que comprende el rango que va desde el Do de la escala -1 (C-1) hasta el sol de la octava 9 (G9). La asignación de números para cada nota se muestra en la Figura 3.20

	Octava										
Note	-1	0	1	2	3	4	5	6	7	8	9
C	0	12	24	36	48	60	72	84	96	108	120
C#	1	13	25	37	49	61	73	85	97	109	121
D	2	14	26	38	50	62	74	86	98	110	122
D#	3	15	27	39	51	63	75	87	99	111	123
E	4	16	28	40	52	64	76	88	100	112	124
F	5	17	29	41	53	65	77	89	101	113	125
F#	6	18	30	42	54	66	78	90	102	114	126
G	7	19	31	43	55	67	79	91	103	115	127
G#	8	20	32	44	56	68	80	92	104	116	
A	9	21	33	45	57	69	81	93	105	117	
A#	10	22	34	46	58	70	82	94	106	118	
B	11	23	35	47	59	71	83	95	107	119	

Figura 3.20: Asignación numérica de notas en MIDI

Volumen

En MIDI al volumen se le conoce como *velocity* y está cuantificado con valores que van desde el 0 para el volumen más bajo (silencio) hasta el 127 para el más alto, este valor dependerá de la fuerza con que se toque la nota en cuestión. Figura 3.21

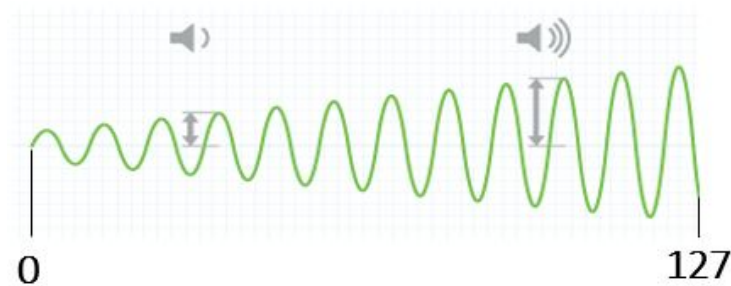


Figura 3.21: Representación del volumen en MIDI

Timbre

MIDI puede asignar el sonido de un instrumento en específico (también conocido como

programa) a la nota que se está reproduciendo. Figura 3.22

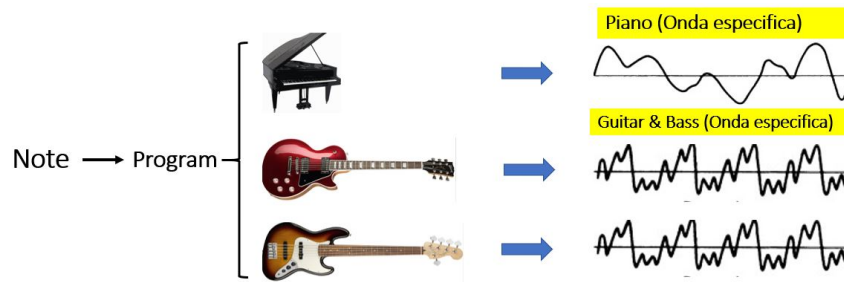


Figura 3.22: Asignación de timbre en MIDI

Canal

MIDI tiene un máximo de 16 canales, cada uno puede enviar y recibir información musical y tener un instrumento diferente asignado. Además de que los parámetros de audio pueden ser modificados independientemente en cada canal. La Figura 3.23 muestra a manera de ejemplo como se ven los primeros tres canales de un archivo MIDI, en el canal 1 se encuentra el audio de la batería, en el canal 2 el bajo y el canal 3 la guitarra. Nótese que no hay un orden específico o estandarizado para la asignación de instrumentos en un canal, es decir el contenido de cada canal podrá ser diferente entre un archivo MIDI y otro.



Figura 3.23: Asignación de canales en MIDI

Mensajes MIDI

Los mensajes MIDI transmiten información musical, estos mensajes llevan datos sobre notas, cambios de volumen, modulación y otros parámetros, permitiendo la comunicación entre instrumentos musicales electrónicos y dispositivos MIDI. Un mensaje MIDI es la unidad básica

de información en el protocolo y los hay de dos tipos, de estado y de datos. Los mensajes MIDI están compuestos principalmente por bytes de 8 bits. El primer bit (el más significativo) para los bytes de estado es siempre 1, mientras que el primer bit de los bytes de datos es siempre 0. Esto permite que los dispositivos MIDI distingan entre bytes de estado y de datos.

Estos mensajes se utilizan para controlar eventos relacionados con un canal específico (instrumento específico). Cada mensaje de canal consta de al menos dos bytes, un byte de estado y uno o más bytes de datos.

Bytes de Estado

Indica el tipo de mensaje y el canal al que se destina, hay un total de 7 tipos de bytes de estado:

0. Note off (Para apagar una nota)
1. Note on (Para activar una nota)
2. Key pressure (Presion de notas polifonicas)
3. Control Change (Efectos y distorsiones)
4. Program change (Cambio de instrumento)
5. Channel pressure (Presion de notas monofonicas)
6. Pitch bend (Para variar la frecuencia de una nota)

En el byte de estado los primeros 4 bits son para el bit más significativo seguido de 3 bits para designar el tipo de mensaje de los 7 posibles descritos anteriormente, y los 4 bits restantes son para designar el canal al cual será aplicado el mensaje como se muestra en la Figura 3.24

En la Figura 3.25 se muestran los 7 tipos de bytes de estado y su respectiva configuración binaria y hexadecimal.

Bytes de datos

Proporciona información adicional sobre el tipo de mensaje de estado y empieza siempre con el primer bit igual a 0 (identificador de byte de tipo datos), en la Figura ?? se pueden observar 3 bytes de un mensaje MIDI, el primer byte es de estado y activa la función Note On para el canal "cccc", el segundo byte es de tipo datos y define la nota que se activará, recuérdese que "nnnnnnn" puede tomar los valores de 0 a 127 dependiendo de la nota que se haya tocado, y el tercer byte define el volumen, los bits "vvvvvv" pueden representar valores desde 0 hasta 127 para definir el nivel de volumen del canal.

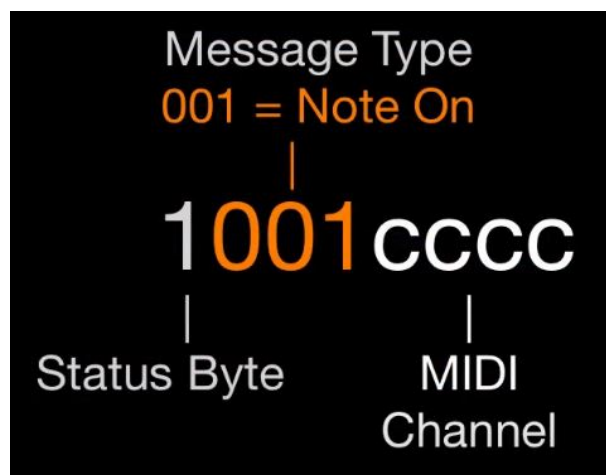


Figura 3.24: Byte de Estado

3.3. Inteligencia Artificial

A lo largo de la historia, la humanidad siempre ha imaginado máquinas con habilidades humanas y hasta seres artificiales con conciencia propia. Un ejemplo de ello data del año 400 a. C. en la antigua mitología griega, donde Talos era un robot gigante de bronce creado para proteger a Europa de los enemigos invasores. Su tarea consistía en patrullar las costas recorriendo toda Creta varias veces al día. Cuando algún enemigo intentaba acercarse, Talos los recibía con enormes pedruscos y en otras ocasiones se calentaba hasta el rojo vivo con grandes fogatas para después abrazar al intruso y calcinarlo. La creación de Talos estuvo a cargo de Hefesto, dios del fuego, la metalurgia y los herreros en Grecia. Según la leyenda, Talos fue hechizado por Medea, quien lo volvió loco con ayuda de espíritus de la muerte orillándolo hasta el suicidio. La Figura 3.27 muestra una representación de Talos extraída de *Historias de dioses y héroes* (1920) de Thomas Bulfinch (PICKOVER, 2021).

Siglos más tarde en el año de 1898 Nikola Tesla (1845 – 1943) realizó la demostración del prototipo de un barco, el cual era manipulado por radio como se ilustra en la Figura 3.28. Cuando un reportero del New York Times sugirió que podría ser utilizado como un arma de guerra, Tesla le respondió que pensara más allá de los torpedos y se diera cuenta que el barco representaba el primer autómatas, entes mecánicos que realizarían el trabajo pesado de la raza humana. Tesla en su ensayo (Tesla, 2022) afirmó que “... *los autómatas contruidos hasta ese momento tenían mentes prestadas [...], pues formaban parte del operador distante*”. Tesla fue más allá al sugerir, según (PICKOVER, 2021), “... *se podría idear un autómatas que tuviera su propia mente y fuera capaz de llevar a cabo por sí solo (en respuesta a influencias*

STATUS		DATA BYTES	DESCRIPTION
Hex	Binary		
8nH	1000nnnn	0kkkkkkk 0vvvvvvv	Note Off vvvvvvv: note off velocity
9nH	1001nnnn	0kkkkkkk 0vvvvvvv	Note On vvvvvvv $\neq$ 0: velocity vvvvvvv = 0: note off
AnH	1010nnnn	0kkkkkkk 0vvvvvvv	Polyphonic Key Pressure (Aftertouch) vvvvvvv: pressure value
BnH	1011nnnn	0ccccccc 0vvvvvvv	Control Change (See Table III) ccccccc: control # (0-119) vvvvvvv: control value ccccccc = 120 thru 127: Reserved. (See Table IV)
CnH	1100nnnn	0ppppppp	Program Change ppppppp: program number (0-127)
DnH	1101nnnn	0vvvvvvv	Channel Pressure (Aftertouch) vvvvvvv: pressure value
EnH	1110nnnn	0vvvvvvv 0vvvvvvv	Pitch Bend Change LSB Pitch Bend Change MSB

Figura 3.25: Codificación de bytes de estado

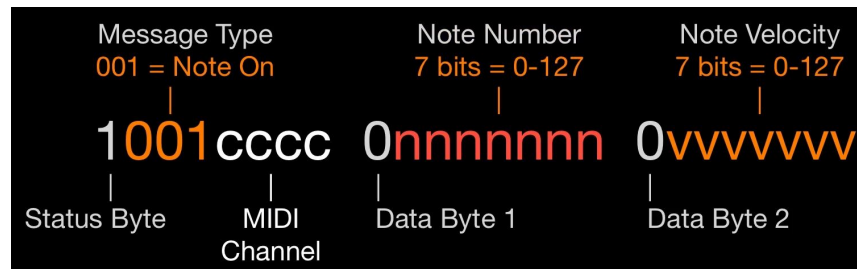


Figura 3.26: Mensaje MIDI

externas que afectaran sus órganos sensitivos) una gran variedad de actos y operaciones como si tuvieran inteligencia...".

En 2016, el sistema robótico autónomo inteligente para tejidos (STAR, *Smart Tissue Autonomous Robot*) demostró su capacidad al operar el intestino delgado de un cerdo utilizando su propia visión, inteligencia artificial (IA) y habilidades técnicas. En comparación con los puntos realizados por cirujanos humanos, las suturas realizadas por STAR fueron más firmes, lo que resultó en un intestino más resistente a las posibles fugas. STAR utilizó etiquetas fluorescentes colocadas en el tejido intestinal para que su cámara pudiera identificar los puntos de sutura más óptimos y realizó ajustes en tiempo real cuando el tejido se movía.



Figura 3.27: Talos, robot de la antigua mitología griega



Figura 3.28: Tesla mostrando el primer dron marino

A diferencia de la cirugía laparoscópica, la cirugía robótica puede eliminar los temblores de la mano del cirujano, y sus movimientos pueden reducirse para lograr una mayor precisión en movimientos más pequeños. En el emergente campo de la telecirugía, los instrumentos robóticos conectados a redes de comunicación de alta velocidad permiten realizar cirugías remotas de forma segura. La Figura 3.29 muestra el sistema para cirugía STAR (Nilsson, 2009).



Figura 3.29: STAR, robot inteligente para cirugía

Talos, el primer dron de Tesla y STAR son solo algunos ejemplos de como a lo largo de la historia la humanidad siempre se ha interesado por el desarrollo de máquinas autónomas e inteligentes. Un amplio tratado sobre el desarrollo histórico de la IA se puede encontrar en el libro *The quest for artificial intelligence* (Nilsson, 2009).

Por otro lado, la IA está revolucionando también el mundo del arte, ofreciendo nuevas herramientas y métodos de creación a artistas contemporáneos. Con algoritmos avanzados, la IA puede generar obras de arte originales, desde pinturas hasta música, analizando patrones y estilos de los grandes maestros. Además, la IA permite la personalización del arte, adaptándose a las preferencias estéticas individuales. Esta colaboración entre humanos y máquinas está desafiando las definiciones tradicionales de creatividad y autoría. A medida que la tecnología avanza, el diálogo entre IA y arte promete transformar el paisaje cultural. Un ejemplo de la conjunción entre IA y arte es *OpenArt*, una aplicación que genera imágenes mediante el procesamiento de texto descriptivo (en inglés) que el usuario da como entrada. La verdadera dificultad para la creación de una imagen ahora radica en la habilidad con la que se describe la ilustración que se desea crear, entre mas detallada y mejor estructurada sea la descripción, más compleja será la imagen. Es tal el alcance de la aplicación, que se ha utilizado para hacer logotipos y diseños de marca, códigos QR con diseños divertidos e incluso modelos arquitectónicos. La ilustración que se muestra en la Figura 3.30 fue creada mediante el texto descriptivo *Photo of a guitar player in the moon with the earth in the background*

(Open Art, 2024).



Figura 3.30: Guitar Player, imagen creada mediante IA

En el campo de la música también ha habido avances tecnológicos de IA, una muestra de ello es la aplicación AIVA (*AIVA - Your personal AI music generation assistant*, 2024), cuyo modelo fue entrenado con más de 30,000 pistas de audio y permite generar canciones nuevas mediante opciones de entradas predefinidas que el usuario va seleccionando hasta crear una pieza musical nueva. Las entradas disponibles incluyen instrumentación, tonalidad, tiempo (BPM), progresión de acordes, género, entre otras. Una vez seleccionadas las entradas, la aplicación se encarga de generar una pista de audio nueva, la cual puede ser editable por el usuario en caso de que se desee modificar una sección específica de la canción.

Para poder comprender mejor a la inteligencia artificial es necesario entender cómo funciona el aprendizaje en los seres humanos, principalmente aprendemos a través de la percepción sensorial (visión, olfato, audición, gusto y tacto), la experiencia directa y la observación. Utilizamos la cognición para procesar información y ajustar nuestros comportamientos dependiendo de las condiciones del entorno. Y además, las emociones y la motivación juegan un papel crucial en el aprendizaje humano, ya que factores como la curiosidad e interés impulsan la adquisición de nuevo conocimiento.

Formalmente, se define a la inteligencia artificial como una disciplina de la informática centrada en la *creación de sistemas capaces de realizar tareas que, en general, requieren intelligen-*

cia humana, por ejemplo, la toma de decisiones, la comprensión y generación de lenguaje escrito y hablado, la capacidad para resumir información o mostrar creatividad. (Nilsson, 2009)

Por otro lado, la IA aprende a partir del procesamiento de grandes volúmenes de datos que le permiten identificar patrones, entender contextos y tomar decisiones. En analogía con los seres humanos que estudian y aprenden de los ejemplos, la IA ajusta sus respuestas a medida que procesa más información, mejorando su precisión y capacidad para resolver problemas. En contraste la IA carece de emociones y su creatividad es una combinación de patrones aprendidos. Aunque la IA puede procesar y aprender rápidamente de grandes cantidades de datos, puede tener dificultades para comprender el contexto y generalizar conocimientos a situaciones nuevas. (Nagy, 2018)

3.3.1. Aprendizaje Automático (ML, Machine Learning)

La forma tradicional de hacer que una computadora realice una tarea (hasta hace algunos años) era tener un programador humano que escribiera las reglas (un programa de computadora) que se seguían para convertir datos de entrada en respuestas apropiadas ó salidas. El **aprendizaje automático** cambia este enfoque, ya que ahora la máquina analiza los datos de entrada y las respuestas correspondientes, y determina cuáles deberían de ser las reglas. En pocas palabras, un sistema de aprendizaje automático se entrena (en lugar de ser explícitamente programado) con muchos ejemplos relevantes para una tarea, y encuentra estructuras implícitas en estos ejemplos que eventualmente permiten al sistema generar reglas para automatizar la tarea (Chollet, 2020).

De manera formal el **Aprendizaje Automático** se define como *una rama de la IA que se centra en el desarrollo de algoritmos y modelos que permiten a las máquinas aprender a partir de datos; en lugar de ser programadas con reglas fijas para realizar una tarea.* (Harrington, 2012). La figura 3.31 muestra la diferencia entre la programación clásica y el aprendizaje profundo:



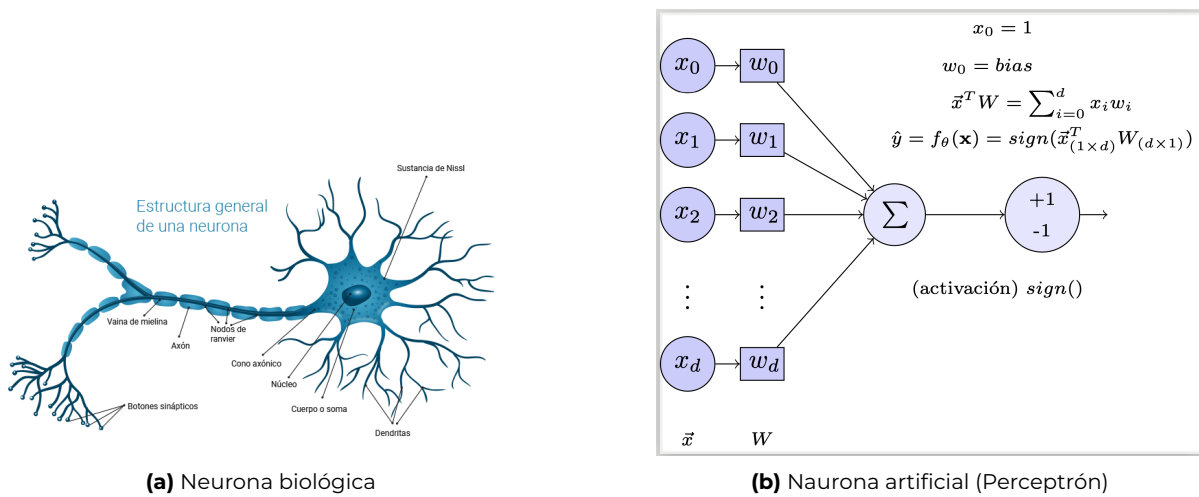
Figura 3.31: Comparación entre la programación clásica y el aprendizaje automático.

El aprendizaje a partir de datos se utiliza en situaciones donde no tenemos una solución analítica, pero sí contamos con datos que podemos emplear para construir una solución empírica. El aprendizaje a partir de datos es actualmente una de las técnicas más ampliamente utilizadas en ciencia, ingeniería y economía, entre otros campos (Abu-Mostafa, 2012).

Considérese que se está diseñando un sistema para clasificar correos electrónicos como *spam* o *no spam*. En lugar de programar reglas explícitas como "si el correo contiene la palabra 'gratis', marcar como spam", un sistema de aprendizaje automático aprende de los datos. Primero, se le dan miles de ejemplos de correos electrónicos ya clasificados (etiquetados como *spam* o *no spam*). El sistema analiza esos ejemplos y encuentra patrones, como la frecuencia de ciertas palabras, la estructura del mensaje o incluso el remitente. A partir de esos patrones, el sistema genera sus propias reglas para clasificar nuevos correos. De este modo, si un correo no contiene la palabra "gratis", pero tiene características similares a otros correos etiquetados como spam, el sistema podría identificarlo correctamente.

El Perceptrón

El concepto de Perceptrón fue desarrollado en los años 1950's y 1960's por Frank Rosenblatt y sentó las bases para la teoría de lo que hoy se conoce como *Sigmoid Neuron* (Haykin, 2005). El Perceptrón se inspiró en una neurona biológica, la cual consta de un cuerpo celular, múltiples dendritas y un solo axón. Las conexiones entre las neuronas se conocen como *sinapsis*. La neurona recibe estímulos a través de las dendritas y, en caso de que los estímulos sean suficientes, la neurona se *activa* (excita o dispara) y envía un estímulo a través de su axón, el cual se transmite a otras neuronas que tienen conexiones sinápticas con la neurona activada. Las señales sinápticas pueden ser excitadoras o inhibidoras; es decir, algunas señales pueden impedir que una neurona se active en lugar de hacer que se active.

**Figura 3.32:** Inspiración para crear el Perceptrón

Por otra parte, un Perceptrón es una función que tiene d cantidad de *entradas* x y produce una sola *salida* binaria y . La representación gráfica de un perceptrón se muestra en la Figura 3.32b. Rosenblatt (Haykin, 2005) introdujo también el término de peso (*weight*) asociado a cada entrada y denotado por w , que se interpreta como una manera de medir la importancia que tiene una entrada en una determinada salida. Cada entrada tiene un peso asociado que determina el nivel de influencia a la hora de determinar el valor de la salida.

El parámetro que determina qué valor va a tomar la salida se conoce como **umbral** (*threshold*). Si la sumatoria de las entradas multiplicadas por sus respectivos pesos es menor o igual que el valor del umbral, entonces la salida del perceptrón es -1 . Si la sumatoria de las entradas multiplicadas por sus respectivos pesos es mayor que el valor del umbral, entonces la salida del perceptrón será igual a $+1$. A este proceso de decisión se le conoce como **función de activación**, y en el caso del perceptrón clásico es una **función escalón**. La función de activación transforma la sumatoria ponderada de las entradas en la salida final del perceptrón, decidiendo si esta será -1 o $+1$ en función del umbral. De esta manera, la salida del Perceptrón es binaria, ya que solo puede tomar dos valores, $+1$ ó -1 .

Sea $\mathcal{X} \in \mathbb{R}^d$ el espacio de entradas, donde $\mathbb{R}^d$ es el espacio euclidiano d -dimensional, y sea $\mathcal{Y} = \{+1, -1\}$ el espacio de salida, que denota una decisión binaria. En nuestro ejemplo de clasificación de correos electrónicos, las diferentes coordenadas del vector de entrada $\vec{x} \in \mathbb{R}^d$ corresponden a características como la frecuencia de ciertas palabras, la estructura del mensaje, el remitente y otros elementos relevantes de un correo electrónico, así mismo, la salida binaria y indica si el correo se clasifica como *spam* o *no spam*.

La forma funcional $f_{\theta}(\vec{x})$ que elegimos del conjunto de hipótesis $\mathcal{H}$ asigna diferentes pesos a

las características de $\vec{x}$, reflejando su importancia relativa en la decisión de clasificación. Estas características ponderadas se combinan para generar un "puntaje de spam", que se compara con un valor umbral. Si el puntaje supera el umbral, el correo se clasifica como *spam*; de lo contrario, se clasifica como *no spam*, pudiendo representar estas condiciones por la Ecuación

$$f_{\theta}(\vec{x}) = \text{sign}\left(\sum_{i=1}^d w_i x_i + \text{bias}\right) \quad (\text{Ecuación 3.1})$$

Donde:

- $x_1, \dots, x_d$ son los componentes del vector $\vec{x}$ que representan las características del correo electrónico.
- $f_{\theta}(\vec{x}) = +1$ significa *spam*.
- $f_{\theta}(\vec{x}) = -1$ significa *no spam*.
- $\text{sign}(s) = +1$; si $s > 0$ y $\text{sign}(s) = -1$ si $s < 0$.
- Los pesos $w_1, \dots, w_d$ representan la importancia de cada característica en la clasificación.
- El umbral está determinado por el término de sesgo *bias*.

Algunos de los pesos $w_1, \dots, w_d$ podrían ser negativos, indicando que ciertas características disminuyen la probabilidad de que un correo sea spam. Por ejemplo, el peso asociado a la ausencia de palabras sospechosas podría ser negativo, ya que esto indica que el correo probablemente no es spam. El valor del sesgo *bias* podría ser grande o pequeño, reflejando qué tan estricto debería ser el modelo al clasificar un correo como spam.

Para simplificar la notación de la ecuación Ecuación 3.1 del perceptrón, trataremos el sesgo *bias* como un peso $w_0 = \text{bias}$ y lo combinaremos con los otros pesos en un único vector $W = [w_0, w_1, \dots, w_d]^T$, donde T denota la transposición de un vector, por lo que W es un vector columna. También tratamos $\vec{x}$ como un vector columna y lo modificamos para que sea $\vec{x} = [x_0, x_1, \dots, x_d]^T$, donde la coordenada adicional x_0 está fija en $x_0 = 1$. Finalmente, el espacio de entradas ahora está determinado por $\mathcal{X} \in \mathbb{R}^d = \{[x_0, x_1, \dots, x_d]^T \mid x_0 = 1, x_1 \in \mathbb{R}, \dots, x_d \in \mathbb{R}\}$

Con esta convención, $\vec{x}^T W = \sum_{i=0}^d x_i w_i$, y por lo tanto la Ecuación 3.1 puede reescribirse en forma vectorial como:

$$\hat{y} = f_{\theta}(\vec{x}) = \text{sign}(\vec{x}^T W) \quad (\text{Ecuación 3.2})$$

Funciones de activación

Como se explicó anteriormente, el perceptrón emplea una función de activación $sign(s)$, que activa o desactiva la salida en función de un umbral fijo. Aunque útil, esta función limita la capacidad del perceptrón para abordar problemas complejos y no lineales. En los modelos de aprendizaje profundo, en cambio, se utilizan funciones de activación más avanzadas, que permiten transformar la entrada de cada neurona en una salida no lineal. Estas funciones son esenciales para modelar relaciones no lineales, mejorando la capacidad de aprendizaje del modelo y permitiendo el desarrollo de redes profundas que capturan patrones complejos en los datos.

Los diferentes tipos de funciones de activación enumeradas aquí, son básicamente diferentes funciones matemáticas no lineales. Algunas de las funciones de activación más populares son las siguientes:

- **Sigmoide:** Una función sigmoide o también conocida como función logística es expresada mediante la ecuación Ecuación 3.3

$$\sigma(x) = \frac{1}{1 + e^{-x}} \quad (\text{Ecuación 3.3})$$

La gráfica de la función sigmoide se muestra en la figura 3.33:

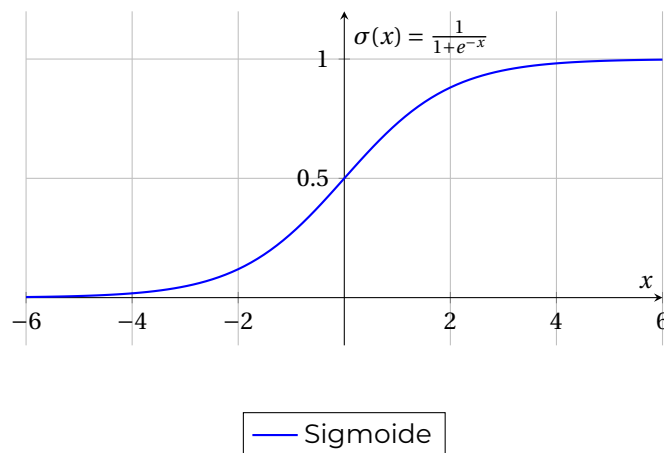


Figura 3.33: Gráfica de la función sigmoide

Como se puede observar en la gráfica de la función sigmoide, el valor de y tiene un rango de $(0, 1)$.

- **tanh (Tangencial Hiperbólica):** Es una función que transforma los valores de entrada a un rango entre $(-1, 1)$, por ende, esta activación es útil en casos en los que se requieren salidas negativas y positivas. Esta función se expresa mediante la ecuación Ecuación 3.4:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (\text{Ecuación 3.4})$$

La gráfica de esta función se muestra en la Figura 3.34:

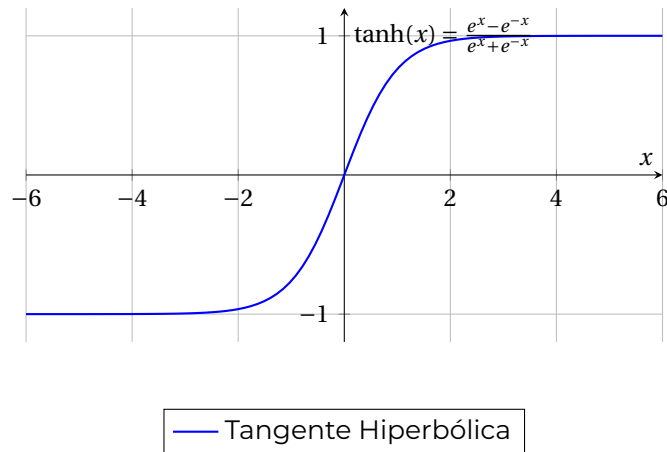


Figura 3.34: Gráfica de la función Tangente Hiperbólica (TanH)

- **ReLU (Rectified Linear Units):** La función ReLU se define matemáticamente como en la ecuación Ecuación 3.5:

$$ReLU(x) = \max(0, x) \quad (\text{Ecuación 3.5})$$

Esto significa que si el valor de entrada de x es positivo, la salida es $y = x$; mientras que si el valor de entrada de x es negativo o cero, la salida es $y = 0$.

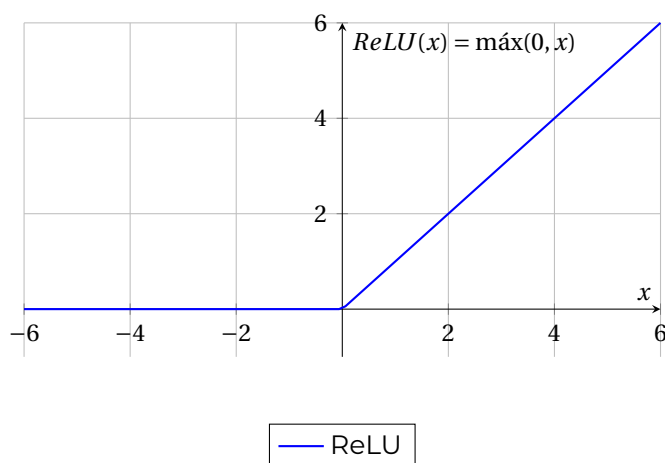


Figura 3.35: Gráfica de la función ReLU

- Leaky ReLU:** Es una variante de la función de activación ReLU, diseñada para solucionar uno de sus principales problemas: la "muerte de neuronas". Este problema ocurre cuando las neuronas quedan inactivas y producen siempre un valor de salida de 0 para cualquier entrada negativa, impidiendo que la red aprenda de esas neuronas. La función Leaky ReLU introduce una pequeña pendiente en el lado negativo, de modo que, en lugar de ser 0, los valores negativos se multiplican por un pequeño valor α . Esto se expresa matemáticamente mediante la ecuación Ecuación 3.6:

$$LeakyReLU(x) = \begin{cases} x & \text{si } x \geq 0 \\ \alpha x & \text{si } x < 0 \end{cases} \quad (\text{Ecuación 3.6})$$

La gráfica de la función Leaky ReLU se muestra en la Figura 3.36

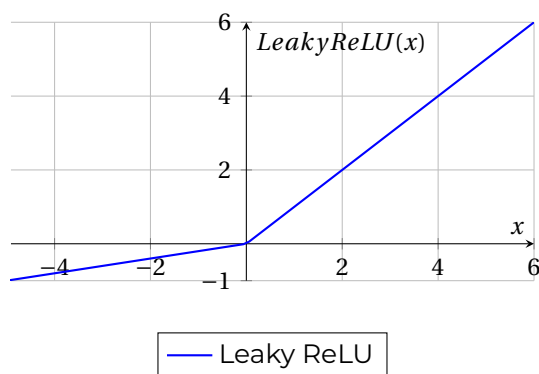


Figura 3.36: Gráfica de la función Leaky ReLU

Las funciones de activación **no lineales** permiten que las redes neuronales aprendan relaciones complejas entre las entradas y las salidas. Sin funciones no lineales, una red neuronal sería equivalente a una simple transformación lineal — es decir, como una única multiplicación de matrices. Considerando la Ecuación 3.7, los pesos de las capas ocultas h^1, h^2 y h^3 pueden ser reemplazadas por una matrix W' ya que la multiplicación de matrices es asociativa.

$$\hat{y} = f_{\theta}(\vec{x}) = \left(\left(\left(\vec{x}_{d \times n}^T W_{n \times m}^{h^1} \right) W_{m \times n}^{h^2} \right) W_{n \times c}^{h^3} \right) = \vec{x}_{d \times n}^T W'_{n \times c}$$

Las funciones de activación tales como: *ReLU()*, *tanh()*, *sigmoid()*, etc.) introducen no linealidades en el modelo, rompiendo la asociatividad de las transformaciones lineales y permitiendo que una red neuronal pueda aproximar cualquier función compleja, es decir le da capacidad a una red de: a) Detectar patrones no triviales, b) Separar datos que no son linealmente separables, c) Aproximar funciones complicadas.

Perceptrón multicapa

Una vez definido el perceptrón, se puede extender el concepto para poder formar redes de perceptrones que pueden resolver problemas mas complejos. En la Figura 3.37 se muestra una red con tres capas ocultas, en la primer capa de perceptrones se evalúan d condiciones simples considerando los pesos y las entradas. En la segunda capa cada perceptrón esta evaluando los resultados de la primer capa y sus respectivos pesos, pero esta vez la evaluación de entradas y pesos es mas compleja y abstracta que la primera. Y en consecuencia en la tercer capa se evalúan condiciones mas complejas que en las dos capas anteriores:

$$\hat{y} = f_{\theta}(\vec{x}) = \left(\left(\left(\vec{x}_{d \times n}^T W_{n \times m}^{h^1} \right) W_{m \times n}^{h^2} \right) W_{n \times c}^{h^3} \right) \quad (\text{Ecuación 3.7})$$

Es importante recordar que un perceptrón siempre tiene una única salida, las flechas múltiples que se aprecian en la imagen anterior son solo para denotar que la misma salida de un perceptrón es utilizada como entrada para la siguiente capa de perceptrones.

Función de pérdida y optimización de un modelo de red neuronal

Antes de continuar, recordemos que usamos $\vec{x}$ para denotar los datos de entrada y $f_{\theta}()$ para representar un modelo de red neuronal. La **etiqueta** y o salida asociada con $\vec{x}$, representa el valor real o la *respuesta correcta* que queremos que el modelo aprenda a **predecir**. En un

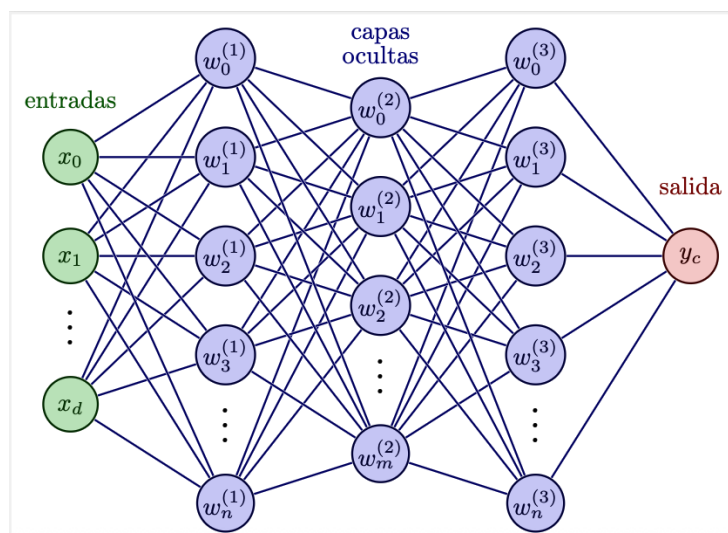


Figura 3.37: Multi Perceptrón

problema de **clasificación** sin embargo tendremos una salida multivaluada, representado por un vector de posibilidades $\vec{y}$ que podría representar la clase a la que pertenece una imagen. Nuestro modelo toma $\vec{x}$ y produce una predicción $\hat{y}$ expresado como $\hat{y} = f_{\theta}(\vec{x})$.

En un modelo de aprendizaje profundo la capa de salida final de la red neuronal genera una **pérdida**. Esta pérdida se calcula con la ayuda de una **función de pérdida**, denotada por $\ell(y, \hat{y})$, la cual es una medida que indica qué tan lejos están las predicciones de un modelo $\hat{y}$, con respecto a los valores reales esperados y . Un *valor alto de la función de pérdida indica que las predicciones del modelo están lejos* de los valores reales, lo que sugiere que el modelo no está aprendiendo correctamente. Un *valor bajo significa que las predicciones están cerca* de los valores reales, lo que indica un mejor desempeño (Raff, 2022).

Basado en lo anterior, se podría decir que el objetivo es que la función de pérdida sea igual a cero, ya que esto implicaría que las predicciones del modelo coinciden perfectamente con los valores reales y que las predicciones son infalibles en su totalidad. Sin embargo, en la práctica, alcanzar una pérdida de cero es poco común y, en muchos casos, no deseable por alguna de las siguientes razones.

- **Sobreajuste (overfitting):** Si la pérdida es exactamente cero en los datos de entrenamiento, existe el riesgo de que el modelo haya memorizado esos datos en lugar de generalizar patrones útiles. Esto puede llevar a que el modelo funcione mal con datos nuevos o no vistos.
- **Datos ruidosos:** En la mayoría de los conjuntos de datos del mundo real, hay ruido

o variabilidad. Si el modelo intenta reducir la pérdida a cero, podría estar intentando aprender patrones del ruido, lo que también conduce a un mal rendimiento en datos nuevos.

- **Objetivo práctico:** En lugar de buscar una pérdida de cero, el objetivo es minimizar la pérdida tanto como sea posible sin comprometer la capacidad del modelo para generalizar. Esto significa encontrar un punto en el que el modelo haga buenas predicciones en datos nuevos, aunque haya una pequeña pérdida en los datos de entrenamiento.

En conclusión, aunque teóricamente se podría aspirar a una función de pérdida igual a cero, en la práctica se busca una pérdida baja que permita un buen equilibrio entre precisión y generalización. (Chollet, 2020)

La función de pérdida se calcula comparando las predicciones de un modelo con los valores reales esperados y aplicando una fórmula que cuantifique el error o la discrepancia. El cálculo exacto depende del tipo de tarea y del tipo de función de pérdida elegida.

En el caso de que la salida de una red neuronal sea un valor cuantitativo (escalar) $\hat{y}$ que debe compararse con una etiqueta y también escalar, las funciones de pérdida más comunes son:

- **Error Absoluto Medio (MAE):** También usado en regresión, mide el promedio de las diferencias absolutas entre las predicciones y los valores reales.

$$\text{MAE} = \frac{1}{N} \sum_{i=1}^N |y_i - \hat{y}_i| \quad (\text{Ecuación 3.8})$$

- **Error Cuadrático Medio (MSE):** Utilizada en problemas de regresión, donde se mide la distancia entre valores reales y predicciones continuas.

$$\text{MSE} = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2 \quad (\text{Ecuación 3.9})$$

Donde y_i es el valor real, $\hat{y}_i$ es la predicción del modelo, y N es el número de ejemplos o muestras en la base de datos $\mathcal{D}$.

En el caso de que la salida de una red neuronal sea un valor cualitativo (vector) $\hat{\mathbf{y}}$ que debe compararse con conjunto de clases $\mathbf{\tilde{y}}$ (también conocido como *One Hot Vector*), las funciones de pérdida más común es:

- **Entropía Cruzada (Cross-Entropy):** Común en problemas de clasificación, mide la diferencia entre las distribuciones de probabilidad de las predicciones y las etiquetas

reales.

$$\ell(\vec{y}, \hat{\vec{y}}) = -\log(sm(\hat{\vec{y}})) \quad (\text{Ecuación 3.10})$$

Donde $sm()$ es la función **softmax**, definida de la siguiente manera. Dado un vector de valores o puntuaciones (*logits*) $z = [z_1, z_2, \dots, z_C]$, donde C es el número de clases, la probabilidad estimada para la clase i es:

$$sm(\hat{y}_i) = \frac{e^{z_i}}{\sum_{j=1}^C e^{z_j}} \quad (\text{Ecuación 3.11})$$

Softmax asigna a cada puntuación $0, 1, \dots, C-1$ un valor en el rango de $[0, 1]$, tal que la suma de todos ellos sea igual a 1.

Para minimizar la función de pérdida en una red neuronal, los parámetros o pesos W en cada capa del modelo se modifican. El grado de modificación está determinado por un coeficiente, conocido también como **tasa de aprendizaje** (*Learning rate*) representado por η . El procedimiento de actualización de los parámetros en una red neuronal, es a lo que llamamos **optimización de un modelo de red neuronal** y tiene un impacto significativo en la efectividad del entrenamiento.

En resumen, las redes neuronales reciben datos y hacen predicciones o clasificaciones. Todos los ajustes que controlan cómo la red realiza estas predicciones se llaman parámetros. La pérdida nos indicará que tan mal está prediciendo o clasificando el modelo. Los parámetros pueden ser modificados n cantidad de veces hasta alcanzar un valor óptimo de la función de pérdida. La Figura 3.38 muestra el proceso de optimización de un modelo de red neuronal.

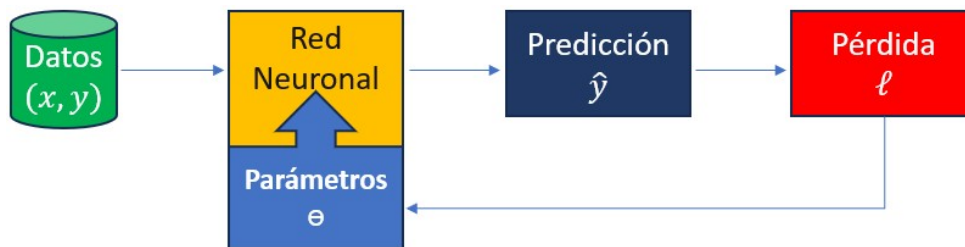


Figura 3.38: Proceso de optimización de un modelo de red neuronal: los parámetros se pueden modificar n cantidad de veces hasta encontrar un valor óptimo para la función de pérdida

Para realizar esto, se ajusta la **tasa de aprendizaje**, que controla la magnitud de los cambios que se aplican a los parámetros en cada paso del entrenamiento. Específicamente, determina el tamaño de los pasos que el algoritmo de optimización da en la dirección opuesta al gradiente de la función de pérdida, con el objetivo de minimizarla.

Supóngase que θ_k representa el conjunto de parámetros actuales del modelo, los cuales se desea optimizar. El objetivo es determinar los nuevos parámetros θ_{k+1} que permitan reducir la función de pérdida del modelo. La actualización de parámetros se realiza según la siguiente regla: los nuevos parámetros θ_{k+1} se obtienen restando a los parámetros actuales θ_k el gradiente de la función de pérdida con respecto a θ_k , calculado sobre el error de predicción de la red neuronal frente a los valores verdaderos, promediado sobre el conjunto de datos y escalado por la tasa de aprendizaje.

$$\theta_{k+1} = \theta_k - \eta \cdot \frac{1}{N} \sum_{i=1}^N \nabla_{\theta_k} \ell(\hat{y}, y_i) \quad (\text{Ecuación 3.12})$$

3.3.2. Modelo de Aprendizaje Automático

Partiendo de los conceptos expuestos en la sección anterior, es necesario considerar tres elementos clave para que construir un modelos de aprendizaje automático, los cuales trabajan en conjunto para permitir que el modelo aprenda y generalice reglas útiles que se aplican a nuevos datos. (Chollet, 2020) :

- **Datos de entrada:** Como imágenes, audio, texto o cualquier otro formato de información.
- **Ejemplos de las salidas/respuestas esperadas:** Proporcionando etiquetas, transcripciones o clasificaciones que guíen el aprendizaje.
- **Una métrica de evaluación:** Permite medir qué tan bien el modelo está logrando predecir las respuestas esperadas.

Construir un modelo de aprendizaje automático es un proceso iterativo. Comienza con la comprensión del problema y luego con la recolección de datos relacionados con el dominio del problema. Posteriormente, estos datos son procesados, limpiados y preparados para el modelado, esto se muestra en la Figura 3.39 :

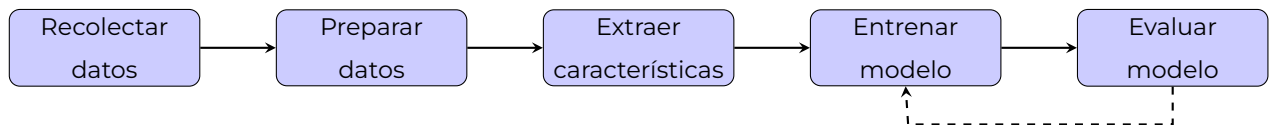


Figura 3.39: Flujo del proceso de modelado de aprendizaje automático.

Cada fase de este proceso se define a continuación (Ghosh, 2023):

- **Recolección de datos:** La recolección de datos es el primer y fundamental paso en la construcción de un modelo de aprendizaje automático. Consiste en reunir información relevante, precisa y suficiente que represente el problema que se desea resolver. La calidad de los datos recolectados tiene un impacto directo en la efectividad del modelo, ya que estos son la base sobre la cual se entrenará y evaluará. En este proceso, es crucial definir las características (atributos) que mejor describen el fenómeno de interés y garantizar que los datos sean variados, equilibrados y representativos del dominio.
- **Preparación de datos:** Este paso implica examinar los datos recopilados para identificar posibles problemas, como valores ausentes o anómalos que puedan haber surgido durante la recolección. Si se detectan estos errores, es necesario corregirlos o eliminarlos para garantizar la calidad del conjunto de datos.
- **Extraer características:** Las características de los datos limpios se analizan más a fondo para identificar intercorrelaciones evidentes. Es posible que algunas características estén altamente correlacionadas, y el uso de una sola de estas características relacionadas sea suficiente para resolver el problema. También puede haber características que no sean relevantes en absoluto para el problema. Después de este paso, se seleccionan algunas características. A veces, es necesario derivar una nueva característica a partir de las características recolectadas. Por ejemplo, se puede usar una función logarítmica para transformar el valor de una característica y utilizar este valor logarítmico como una nueva característica. Este paso también se conoce como ingeniería de características.
- **Entrenar el modelo:** Es el proceso de ajustar los parámetros de una función matemática (o modelo) para que esta pueda aprender a realizar predicciones o clasificaciones basadas en las características seleccionadas de los datos de entrada. El modelo se construye con el objetivo de minimizar la diferencia entre sus predicciones y los resultados reales esperados, a través de un proceso iterativo que optimiza una métrica de error definida. Generalmente, los modelos utilizados en aprendizaje automático son paramétricos, lo que significa que su forma funcional está definida por un conjunto de parámetros que deben ajustarse para obtener el mejor desempeño. En problemas más complejos, como redes neuronales, el modelo puede tener miles o incluso millones de parámetros.
- **Evaluación del modelo:** Se diseñan diversas métricas para evaluar la calidad del modelo. Estas métricas varían según el tipo de algoritmo de aprendizaje automático utilizado. Si se encuentra que la calidad del modelo no es aceptable, se prueban diversas formas de mejorarlo. Esto puede implicar elegir un algoritmo diferente y sus parámetros se estiman

nuevamente. Si aún no se logra encontrar un buen modelo, puede ser necesario regresar a la etapa de preparación de datos para diseñar más características o incluso retroceder más y recolectar datos y características más relevantes para resolver el problema.

Para evaluar un modelo de manera adecuada, es fundamental dividir los datos recolectados en dos conjuntos principales: **entrenamiento** y **validación**. Como su nombre lo indica, el conjunto de entrenamiento se utiliza durante la etapa de entrenamiento para ajustar los parámetros del modelo, permitiéndole aprender patrones y relaciones en los datos. Por otro lado, el conjunto de validación se emplea para evaluar el desempeño del modelo con datos no vistos durante el entrenamiento, lo que ayuda a identificar problemas o sobreajustes (overfit). Figura 3.40.

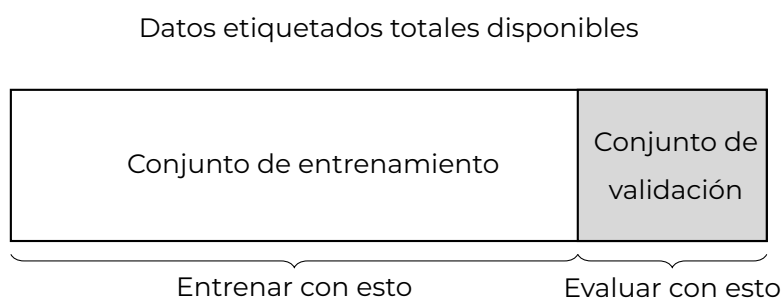


Figura 3.40: División de los datos en dos conjuntos, uno servirá para entrenar al modelo y el otro para evaluar su desempeño.

Esta separación asegura que el modelo no solo aprenda a memorizar los datos de entrenamiento, sino que también sea capaz de generalizar el conocimiento a datos nuevos.

Durante la fase de validación se debe considerar la **fuga de información**, cada vez que se ajusta un parámetro del modelo durante la etapa de validación, una fracción de la información de los datos de validación se filtra hacia el modelo. Si esto ocurre una sola vez, entonces se filtrará muy poca información y el conjunto de datos de validación seguirá siendo confiable para evaluar el modelo.

Sin embargo, si este proceso se repite muchas veces, se filtrará una cantidad cada vez más significativa de información del conjunto de datos de validación hacia el modelo. Esto provocará que el modelo funcione artificialmente bien con los datos de validación, porque es precisamente para eso que fue optimizado.

No obstante, lo importante es el desempeño del modelo con datos completamente nuevos. Se requiere utilizar un conjunto de datos nunca antes visto por el modelo, ni siquiera de manera

indirecta, si algún parámetro(s) del modelo ha sido ajustado basándose en el desempeño del conjunto de validación en su totalidad, la medida de generalización quedará comprometida. (Chollet, 2020)

Existen muchos métodos de cómo dividir los datos entre entrenamiento y validación, algunas técnicas consideran, por ejemplo, procesos estocásticos para dividir los datos de manera mas eficiente. Para mayor detalle sobre el tema, se recomiendan las siguientes referencias: (Ghosh, 2023; Harrington, 2012; Chollet, 2020; Abu-Mostafa, 2012).

3.3.3. Arquitecturas de redes neuronales artificiales

Un modelo de aprendizaje puede construirse en base a varios tipos de arquitecturas de redes neuronales que se han desarrollado a lo largo del tiempo. Un factor distintivo importante entre las diferentes arquitecturas es el tipo y la combinación de capas que se utilizan en la red neuronal. Algunas de las arquitecturas más conocidas son las siguientes (Ghosh, 2023):

Red completamente conectada o lineal (FCN, Fully Connected Network)

En una red completamente conectada, como se muestra en la Figura ??, todas las neuronas que preceden a esta capa están conectadas a todas las neuronas que le siguen:

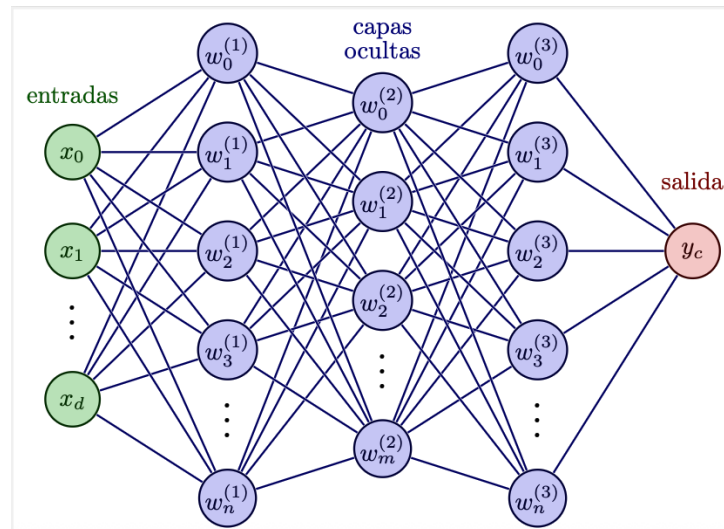


Figura 3.41: Red de capas completamente conectadas (FCN, Fully Connected Network)

La representación matemática de la red completamente conectada de la Figura 3.41 precedente, se encuentra determinada por la Ecuación 3.13, en la que se considera una función de

activación $\tanh$ en cada una de las neuronas de las capas ocultas (1), (2) y (3)

$$\hat{y} = f_{\theta}(\vec{x}) = \left(\tanh \left(\tanh \left(\tanh \left(\vec{x}_{n \times d}^T W_{n \times m}^{(1)} \right) W_{m \times n}^{(2)} \right) W_{n \times c}^{(3)} \right) \right) \quad (\text{Ecuación 3.13})$$

Red Neuronal Convolucional (CNN, Convolutional Neural Networks)

La arquitectura de red neuronal CNN se usa cuando tenemos que procesar datos que tienen una estructura espacial, por ejemplo en imágenes (con sus píxeles organizados en dos dimensiones). El objetivo de las redes convolucionales es capturar patrones locales (bordes, texturas, formas) que luego se combinan para reconocer estructuras más complejas. Estas redes también son aptas para ser aplicadas en señales unidimensionales como el sonido en donde cada valor se registra en función del tiempo.

Formalmente la convolución se define de la siguiente manera. Sea $I \in \mathbb{R}^{W \times H}$ una imagen de tamaño $W \times H$, y $K \in \mathbb{R}^{m \times n}$ un kernel (filtro) de tamaño $m \times n$. La **convolución** de I con el kernel K , denotada como

$$S = I \otimes K \quad (\text{Ecuación 3.14})$$

produce una imagen de salida $S \in \mathbb{R}^{(W-m+1) \times (H-n+1)}$, cuyos valores se definen como:

$$S(i, j) = \sum_{u=0}^{m-1} \sum_{v=0}^{n-1} I(i+u, j+v) \cdot K(u, v) \quad (\text{Ecuación 3.15})$$

para cada $i = 0, 1, \dots, W-m$ y $j = 0, 1, \dots, H-n$.

Una vez definido la operación de convolución, es posible describir matemáticamente una red CNN que emplea una capa oculta convolucional, de la siguiente forma:

$$f_{\theta}(\vec{x}) = \text{vec} \left(\tanh \left(\vec{x} \otimes W^{(h_1)} + b^{(h_1)} \right) \right) \cdot W^{\text{out}} + b^{\text{out}} \quad (\text{Ecuación 3.16})$$

donde:

- $\vec{x} \in \mathbb{R}^{C_{\text{in}} \times W \times H}$ es la imagen de entrada con $C_{\text{in}} = 1$ si la imagen es en niveles de gris o $C_{\text{in}} = 3$ si la imagen es a color. W y H representan el ancho y la altura de la imagen.
- $W^{(h_1)} \in \mathbb{R}^{C_{\text{out}} \times C_{\text{in}} \times K \times K}$ representa C_{out} filtros convolucionales o kernels de tamaño $(K \times K)$ en la capa oculta h_1

- $b^{(h_1)} \in \mathbb{R}^{C_{out}}$ es el sesgo de la convolución
- $W^{out} \in \mathbb{R}^{n_{out} \times (C_{out} \cdot W' \cdot H')}$ son los pesos de la capa *fully connected*
- $b^{out} \in \mathbb{R}^{n_{out}}$ es el sesgo de la capa *fully connected*
- *vec* es una función que sirve para vectorizar o aplanar un tensor en un vector columna.

La Figura 3.42 muestra en un diagrama de bloques la red CNN formalizada en la Ecuación 3.16.

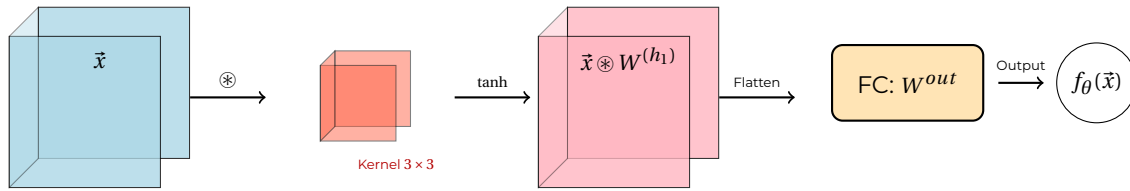


Figura 3.42: Diagrama de una capa convolucional.

A diferencia de una red totalmente conectada (FCN), en una red convolucional (CNN) no es necesario actualizar un gran número de pesos. En una FCN, cada neurona está conectada a todas las entradas, lo que genera una matriz de pesos muy grande que debe ser aprendida y actualizada. En cambio, en una CNN, los *kernels* (filtros) se aplican localmente y comparten los mismos pesos a lo largo de la entrada. Este mecanismo de **peso compartido** reduce drásticamente la cantidad de parámetros que se deben aprender, lo que permite una mayor eficiencia computacional y un mejor aprovechamiento de las correlaciones espaciales de los datos (e.g. en imágenes).

Red Neuronal Recurrentes (RNN, Recurrent Neural Networks)

Una Red Neuronal Recurrente (RNN) es un tipo de red neuronal con una arquitectura diseñada para procesar secuencias de datos, modelando dependencias temporales y contextuales entre elementos de la secuencia. Típicamente en una FCN o en una CNN la entrada es considerada como un vector de características $\vec{x}$, sin embargo en una RNN la entrada es una **secuencia ordenada de vectores** $X = (\vec{x}_1, \vec{x}_2, \dots, \vec{x}_T)$, con cada $\vec{x}_t \in \mathbb{R}^n$ y $X \in \mathbb{R}^{T \times n}$ donde n es la dimensión de cada vector y T es la longitud de la secuencia de vectores. Por ejemplo si consideramos el análisis de texto en una frase. Las palabras pueden ser representadas por cada vector x_t y la frase completa, por la matriz X . Es importante resaltar que el orden de los vectores (*palabras*) es importante para conservar la semántica de la frase. En este modelo de red, los vectores de

entrada no son procesados de forma independiente, ya que la red debe recordar la información procesada en etapas anteriores (histórico) mediante el uso de un *estado oculto* h_t calculado con la entrada actual x_t y el estado oculto anterior h_{t-1} .

$$h_t = \phi(W_{xh}x_t) + W_{hh}h_{t-1} + b_h \quad (\text{Ecuación 3.17})$$

donde:

- h_t es el estado oculto en un tiempo t
- x_t es una entrada en un tiempo t
- W_{xh} son los pesos de la entrada
- W_{hh} son los pesos recurrentes
- b_h es un sesgo de la capa oculta h
- ϕ Es una función de activación (e.g. $\tanh$, $ReLU$, σ)

En una RNN, se puede calcular una salida $\hat{y}$ en cada paso de la secuencia o solo al final, dependiendo de la tarea. Cuando la tarea requiere una predicción o etiqueta en cada instante (como en la generación de texto o la clasificación palabra por palabra), la red produce una salida $\hat{y}_t$ en cada estado oculto h_t . Sin embargo, en tareas donde se necesita una única predicción global para toda la secuencia (como la clasificación del sentimiento de una frase completa), la salida se calcula solo al final usando el último estado oculto h_T .

$$\hat{y}_t = \psi(W_{hy}h_t) + b_y \quad (\text{Ecuación 3.18})$$

donde:

- $\hat{y}_t$ es la salida en un tiempo t
- x_t es una entrada en un tiempo t
- W_{hy} son los pesos en la capa de salida
- b_y es un sesgo en la salida
- ψ Activación de la salida (e.g. softmax)

La Figura 3.43 muestra el cálculo recurrente de los estados ocultos formalizados en la Ecuación 3.17, y también muestra la salida definida por la Ecuación 3.18.

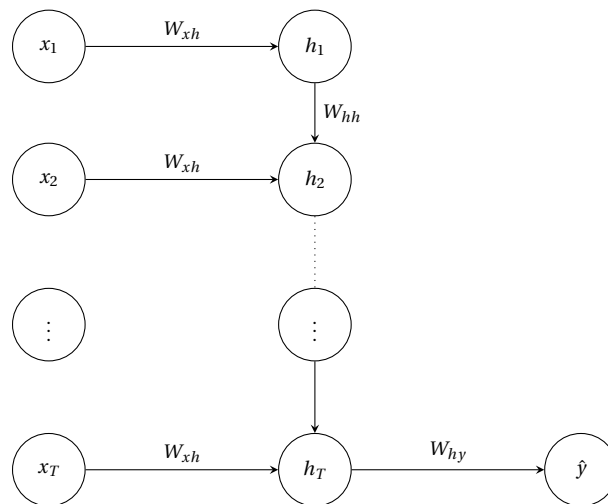


Figura 3.43: Cálculo de los estados ocultos en una RNN.

Capítulo 4

Metodología

En este capítulo se presenta un enfoque detallado sobre la recolección, preparación y utilización de los datos de entrada para el entrenamiento de una red neuronal recurrente tipo LSTM con el propósito de crear *armonías musicales del género Rock-Pop*.

4.1. Fases de desarrollo para el modelo de Red Neuronal (LSTM) para la Generación de armonías de Guitarra y Bajo

La Figura 4.1 muestra el diagrama de bloques del modelo de red neuronal desarrollado en esta tesis, proporcionando una representación visual de la arquitectura y las etapas clave del proceso de generación armónica. Esta estructura permite comprender la interacción entre los distintos componentes de la red y cómo estos contribuyen a la producción de secuencias musicales. Las fases se enlistan en el siguiente orden y serán explicadas en detalle en las secciones posteriores:

- **Preparación del *Dataset* (Datos de entrada)**
- **Definición de la arquitectura**
- **Fase de entrenamiento**
- **Generación**

■ Evaluación

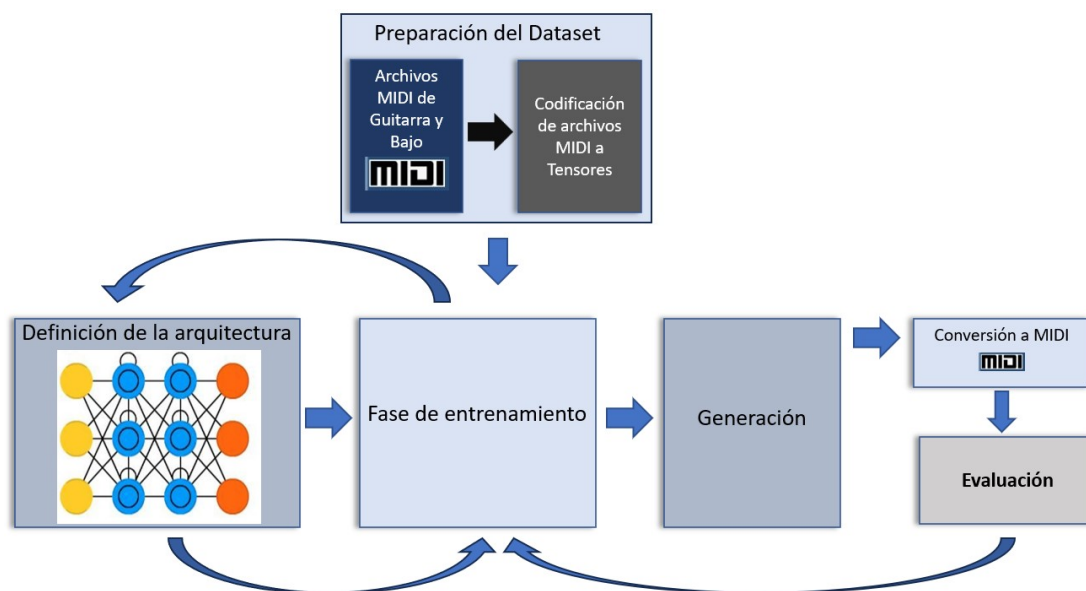


Figura 4.1: Las principales fases del modelo son: Preparación del Dataset, Definición de la arquitectura, Fase de entrenamiento, Generación y Evaluación

4.1.1. Preparación del *Dataset* (Datos de entrada)

En cualquier modelo de red neuronal, la representación de los datos de entrada y su codificación está estrechamente vinculada con las salidas (resultados) del modelo. Si bien, las redes neuronales están diseñadas para extraer automáticamente características relevantes de los datos, la elección de la representación de los datos de entrada es crucial para la calidad del contenido generado.

Basado en lo anterior, se llevó a cabo una evaluación de los diferentes tipos de datos musicales disponibles para determinar el formato más adecuado que permitiera aprovechar al máximo las capacidades de la red neuronal. Durante la evaluación se estudiaron distintos formatos de representación musical para los datos de entrada, por ejemplo, audio (formato WAV y MP3), imagen (tablaturas y partituras) y archivos MIDI. Cada formato presenta sus propias ventajas y desventajas en términos de contenido informativo, complejidad de procesamiento y adecuación para el entrenamiento de modelos generativos. Tras esta comparación, se concluyó que el formato MIDI era el más conveniente para entrenar al modelo por las siguientes razones clave:

- **Almacenamiento y Procesamiento:** Los archivos MIDI son **considerablemente** de menor tamaño que los formatos de audio e imagen, lo que permite manejar grandes volúmenes de datos sin requerir una capacidad de almacenamiento excesiva ni un procesamiento intensivo.
- **Compatibilidad:** El formato MIDI al ser un protocolo de comunicación desarrollado para computadora, proporciona una representación estructurada y clara de la música, describiendo eventos musicales discretos como notas y sus propiedades (por ejemplo, duración e intensidad), lo que es altamente **compatible** con las necesidades computacionales para el entrenamiento de una red neuronal.
- **Manipulación:** La disponibilidad de librerías en Python como `music21`, `mido` y `pretty_midi`, entre otras, permiten la manipulación de archivos MIDI y la transformación directa de los elementos musicales a un archivo .CSV, otorgando un control más preciso para el procesamiento de datos musicales. Esto es especialmente útil para convertir la información musical del archivo MIDI a formatos más adecuados para el entrenamiento de una red neuronal, en este proyecto en específico se utilizaron **Tensores**.

En contra parte, en el formato MIDI se identifican las siguientes desventajas:

- **Pérdida de información expresiva:** Los archivos MIDI solo contienen *notas, duración y velocidad (volumen)*, pero no incluyen detalles como *timbre, vibrato o articulaciones*, lo que limita la interpretación realista.
- **Dependencia del sintetizador:** Un archivo MIDI no tiene sonido propio, depende del *motor de síntesis o banco de sonidos* que lo reproduzca.
- **Interpretación humana:** MIDI suele sonar *mecánico*, ya que no captura las *microvariaciones en el tiempo y dinámica* que aportan realismo a una grabación de audio.

Recolección y Manipulación de archivos MIDI

Las canciones en formato MIDI fueron descargadas del sitio **mideworld** (<https://www.mideworld.com>) dedicado a la distribución de recursos musicales. La selección de esta fuente de datos se basó en criterios de calidad y accesibilidad, asegurando que el conjunto de datos recopilado sea representativo para el género Rock-pop de los años 1960s-2000's.

Para la preparación de los datos fue necesario modificar los archivos MIDI descargados, los cuales contienen múltiples canales, cada uno asignado a un instrumento musical en específico

en la canción. La modificación consistió en extraer los canales de guitarra y bajo que son los instrumentos que utilizaremos para crear las secuencias armónicas, todos los demás canales fueron eliminados del archivo MIDI.

Para llevar a cabo la separación de canales y extraer únicamente los canales de guitarra y bajo, se utilizó **Reason 13**, un software de audio digital tipo DAW (*Digital Audio Workstation*), ampliamente utilizado en la industria musical. Reason ofrece una amplia gama de herramientas para la edición y manipulación de archivos MIDI, lo que lo convierte en una opción ideal para nuestro propósito. A continuación, se describe el proceso detalladamente:

- **Visualización de archivos MIDI en Reason:** El primer paso consistió en abrir los archivos MIDI en Reason y visualizar todos los canales/instrumentos contenidos en cada archivo.



Figura 4.2: Visualización de instrumentos/canales en Reason, para esta imagen en específico se puede apreciar que el Canal 1 contiene Piano Melody, el Canal 2 Piano Chords, y los Canales 3 y 4 instrumentos de percusión

- **Identificación de los Canales de Guitarra y Bajo:** Una vez abiertos los archivos MIDI en Reason, el siguiente paso fue identificar los canales de guitarra y bajo. Algunos archivos MIDI en la base de datos tenían los canales identificados con el nombre del instrumento como se muestra en la Figura 4.2; sin embargo la mayoría de los canales en los archivos MIDI carecen de una identificación adecuada, esto es debido a que el nombre o etiqueta del canal es asignado deliberadamente por el creador del archivo MIDI, debido a esto, fue necesario escuchar y analizar cada canal de cada archivo MIDI para identificar la guitarra y el bajo.
- **Extracción de Canales de Bajo y Guitarra:** Después de identificar los canales de guitarra y bajo. Utilizando las herramientas de edición en Reason, se crean dos archivos MIDI nuevos, uno exclusivamente para guitarra y el otro para bajo.
- **Archivos MIDI con más de un canal de guitarra:** Desde un punto de vista musical, es común que las agrupaciones de Rock y Pop cuenten con dos o más guitarristas, cuyas

interpretaciones se complementan a lo largo de una canción. Al analizar los archivos MIDI, se observó esta misma dinámica, ya que varios MIDIs contenían más de un canal de guitarra, véase la Figura 4.3. Para capturar adecuadamente esa interacción, fue necesario combinar manualmente los distintos canales de guitarra en un único canal consolidado, Figura 4.4. Este proceso resultó imprescindible para eficientar la cantidad de archivos MIDI de guitarra que se iban a procesar, logrando una representación coherente y completa de la armonía de guitarra presente en la canción y contenida en un solo canal.

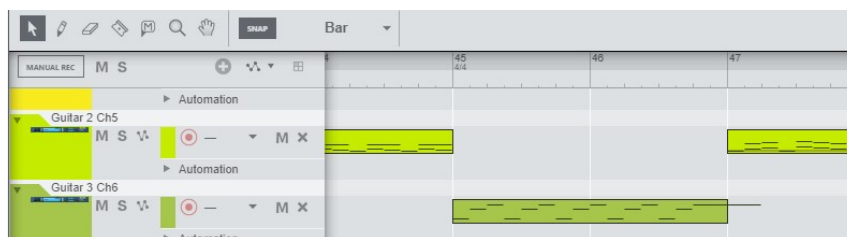


Figura 4.3: Dos canales de guitarra en un mismo Archivo MIDI

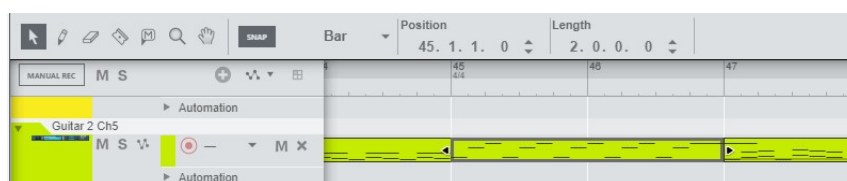


Figura 4.4: Combinación de 2 canales de guitarra en un mismo canal

Una vez creados los archivos MIDI de guitarra y bajo individualmente, fue necesario transformar esta información en un formato adecuado para el procesamiento con redes neuronales.

Codificación de archivos MIDI en Tensores

La Figura 4.5 muestra de manera visual como los archivos MIDI se codificaron en un Tensor (matriz), las filas representan notas musicales y las columnas representan el tiempo. Cada archivo MIDI se transforma en una matriz que captura cuándo se toca una nota a una razón de muestreo de 5 *frames* por segundo (esto define cuántas veces por segundo se captura la información de las notas, en este caso 5 veces por segundo) de modo que solo conservamos información sobre si una nota está siendo tocada (1) o no (0) a lo largo del tiempo.

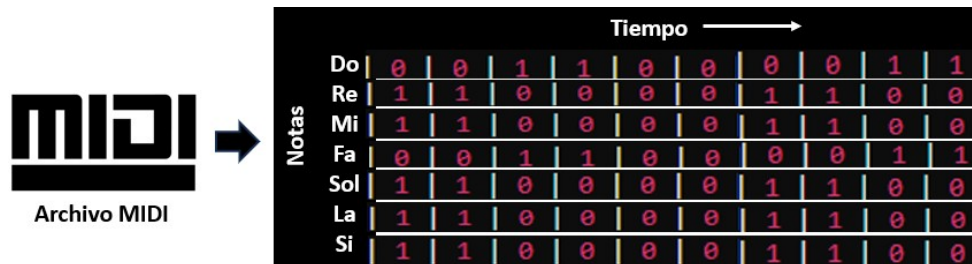


Figura 4.5: Cada archivo MIDI se transforma en un Tensor que captura cuándo una nota está siendo tocada (1) o no (0) a lo largo del tiempo, similar a una representación del tipo Piano roll

Visto de manera más sencilla, esta matriz/tensor es una representación musical del tipo **Piano Roll**, explicada en la sección **Representación de Audio** en la figura 3.19, pero adaptada para su procesamiento en redes neuronales de manera numérica.

El código en Pytorch para la transformación de archivos MIDI a Tensores se muestra en el **Apéndice A** sección .1 (Codificación de los archivos MIDI en Tensores - Código).

Proceso de padding para los archivos MIDI transformados a Tensores:

Una vez que se crean los tensores, el siguiente paso es asegurarnos de que todos los arreglos tengan el mismo tamaño. Como los archivos MIDI tienen diferentes duraciones, aplicamos una técnica llamada **padding**, donde las secuencias más cortas se rellenan con valores (0) hasta alcanzar la longitud máxima del MIDI mas grande, véase la Figura 4.6. Esta operación asegura que todas las secuencias tengan las mismas dimensiones y sean aptas para ser procesadas por el modelo.

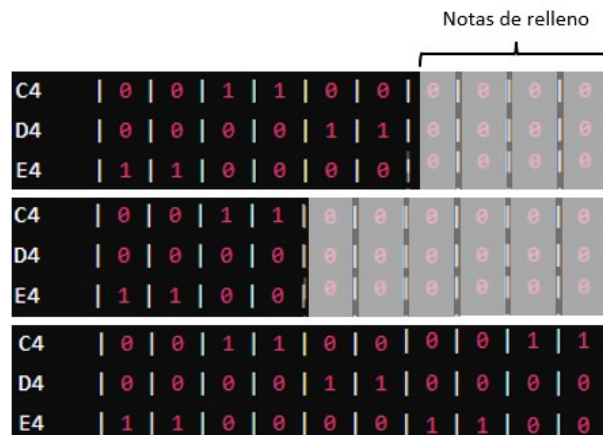


Figura 4.6: Proceso de padding: Para que los Tensores tengan las mismas dimensiones y puedan ser procesadas por el modelo

De esta manera obtenemos nuestra base de datos (*Dataset*) en un formato listo para ser utilizado durante la fase de entrenamiento.

El código para el proceso de *padding* de los Tensores resultantes de los archivos MIDI se encuentra descrito en la sección .2 (Proceso de padding para los archivos MIDI transformados a Tensores:) en el **Apéndice A**.

División del *Dataset* en lotes (*batches*) de entrenamiento y validación

Es importante dividir los datos para poder entrenar y evaluar el modelo de manera efectiva, como se muestra en la Figura 4.7 . Utilizando el 80% de los datos para entrenamiento y el 20% restante para validación. Esto garantiza que el modelo se entrene con la mayoría de los datos, pero también se evalúe con un conjunto separado, para verificar que no está "memorizando" las secuencias sino generalizando su aprendizaje.

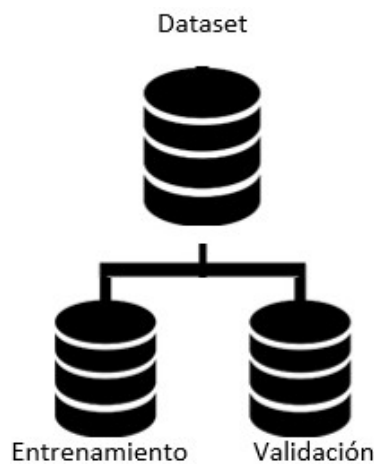


Figura 4.7: División del dataset en entrenamiento y validación

En esta primera fase del modelo, lo que hemos hecho es preparar cuidadosamente los datos para que puedan ser utilizados en una red neuronal tipo LSTM. Comenzamos transformando archivos MIDI en matrices/Tensores, ajustando las secuencias para que todas tengan la misma longitud (proceso de *padding*), y organizando los datos en un formato que facilite el aprendizaje. Después, aseguramos que el modelo será entrenado y evaluado correctamente al dividir los datos en conjuntos de entrenamiento y validación. Este proceso garantiza que los datos estén listos para ser utilizados de manera eficiente durante el entrenamiento de la red.

El código desarrollado en Python para la división de los datos de entrada en lotes (*batches*) de entrenamiento y validación se muestra en la sección .3 (División del *Dataset* en lotes

(*batches*) de entrenamiento y validación) en el **Apendice A**.

4.1.2. Definición de la arquitectura del modelo LSTM

La clase `MusicLSTM` fue diseñada como un modelo secuencial basado en redes neuronales recurrentes del tipo LSTM, con el objetivo de generar secuencias musicales armónicas. La arquitectura fue implementada en PyTorch mediante la subclase `nn.Module`, y en su constructor se definen los parámetros fundamentales y la configuración de las capas del modelo.

En conjunto, estas capas conforman una arquitectura capaz de aprender secuencias musicales con continuidad temporal. El flujo inicia con una secuencia de notas MIDI (77 dimensiones), la cual se transforma mediante una capa lineal a un espacio de 512 dimensiones. Posteriormente, dos capas LSTM procesan la información considerando dependencias temporales. Finalmente, el espacio oculto se proyecta nuevamente a 77 dimensiones para reconstruir la secuencia musical, representando activaciones binarias de notas MIDI.

A continuación, se definen las capas que conforman la arquitectura:

- **Capa densa de entrada** (`nn.Linear`): Transforma el vector binario de 77 dimensiones (que representa el conjunto de notas activadas en un instante de tiempo) en un vector denso de dimensión `hidden_size = 512`. Esta transformación permite que el modelo trabaje con una representación continua y de mayor dimensionalidad.
- **Capa de normalización por lotes** (`BatchNorm1d`): estabiliza la activación de las neuronas al reducir la covariancia interna entre capas. Esta técnica acelera la convergencia del entrenamiento y permite utilizar tasas de aprendizaje más altas sin riesgo de inestabilidad.
- **Capa LSTM** (`nn.LSTM`): Red neuronal recurrente de tipo LSTM con 2 capas y 512 unidades por capa.
- **Capa densa de salida** (`nn.Linear`): Proyecta la salida de la LSTM nuevamente a una dimensión de 77, correspondiente a las posibles notas musicales a predecir. Esta capa permite generar una probabilidad de activación para cada nota en cada instante de tiempo.

La Figura 4.8 es una representación del modelo desarrollado:

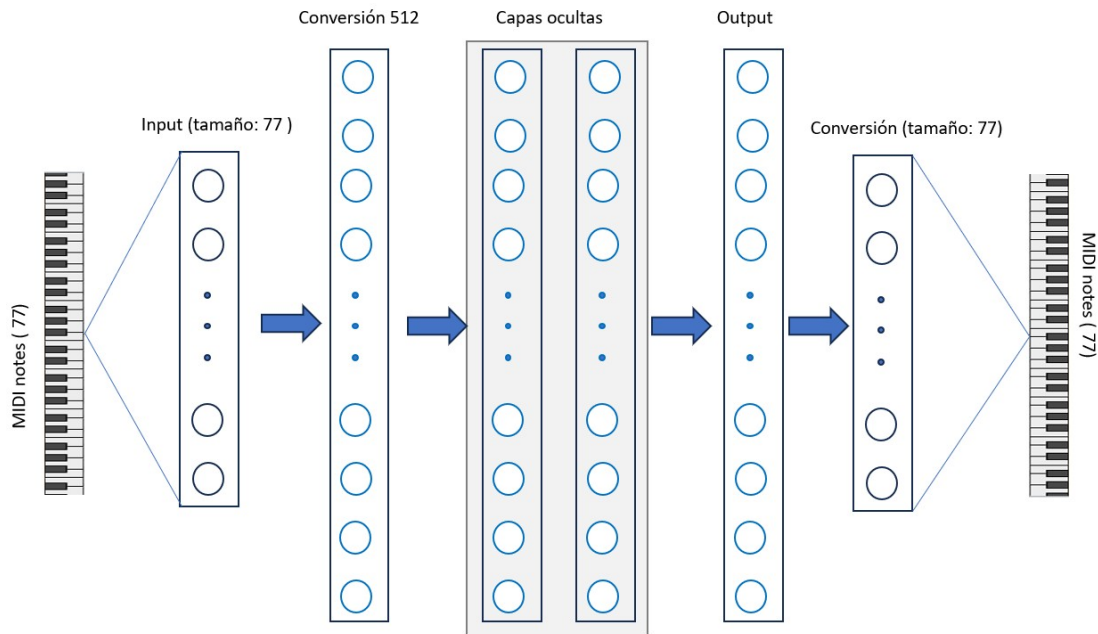


Figura 4.8: Representación esquemática de la arquitectura del modelo.

El método `forward` define el recorrido de los datos a través de las distintas capas del modelo `MusicLSTM`, desde la entrada hasta la generación de una predicción binaria por nota.

El código en `pytorch` que define la arquitectura del modelo se muestra en la sección .4 (Definición de la arquitectura del modelo LSTM para generación musical) en el **Apendice A**.

4.1.3. Fase de entrenamiento del modelo

La fase de entrenamiento del modelo LSTM consiste en la optimización de sus parámetros a partir de un conjunto de datos etiquetado, mediante la retropropagación del error y el ajuste de pesos. A continuación se explica el proceso de entrenamiento:

- **Configuración de hiperparámetros:** Se establecen los tamaños de entrada y salida del modelo (77 clases posibles), el tamaño del estado oculto (512) y el número de capas LSTM (2).
- **Inicialización del modelo y funciones auxiliares:** El modelo `MusicLSTM` se inicializa y se transfiere a GPU utilizando `.cuda()`.
- **Función de Pérdida:** La función de pérdida utilizada es `CrossEntropyLoss`, adecuada para tareas de clasificación multiclase. El optimizador seleccionado es `Adam`, conocido por su eficiencia y estabilidad en tareas de aprendizaje profundo.

- **Bucle de entrenamiento:** Durante cada época, el modelo se pone en modo de entrenamiento mediante `model.train()`. Para cada lote del conjunto de entrenamiento:
 1. Se preprocesan las secuencias con la función `orderTensors`, que ordena y alinea los tensores según la longitud de las secuencias.
 2. Se realiza una pasada hacia adelante (*forward pass*) para obtener las predicciones del modelo.
 3. Se calcula la pérdida entre la salida del modelo y las etiquetas reales.
 4. Se realiza la retropropagación del error con `loss.backward()` y se actualizan los pesos mediante `optimizer.step()`.
 5. Se aplica `clip_grad_norm_()` para prevenir la explosión del gradiente, limitando la magnitud del mismo a un valor máximo de 1.0.
- **Bucle de validación:** Tras cada época, el modelo se evalúa utilizando el conjunto de validación. En modo `eval()`, se desactiva el comportamiento estocástico de capas como `dropout`. Se calcula la pérdida total sobre las secuencias de validación y se normaliza por el tamaño total de la secuencia, almacenando el resultado en una lista para su posterior análisis.

Tras realizar diversas modificaciones a los hiperparámetros del modelo (ajustes en el número de épocas de entrenamiento, el tamaño del espacio oculto y técnicas de regularización) se obtuvieron resultados consistentes en la generación de secuencias musicales.

La Figura 4.9 muestra una comparación visual entre una canción real y una secuencia generada por el modelo. Las notas en color gris representan una canción conocida del corpus de entrenamiento, mientras que las notas en color naranja corresponden a la producción generada por la red neuronal.

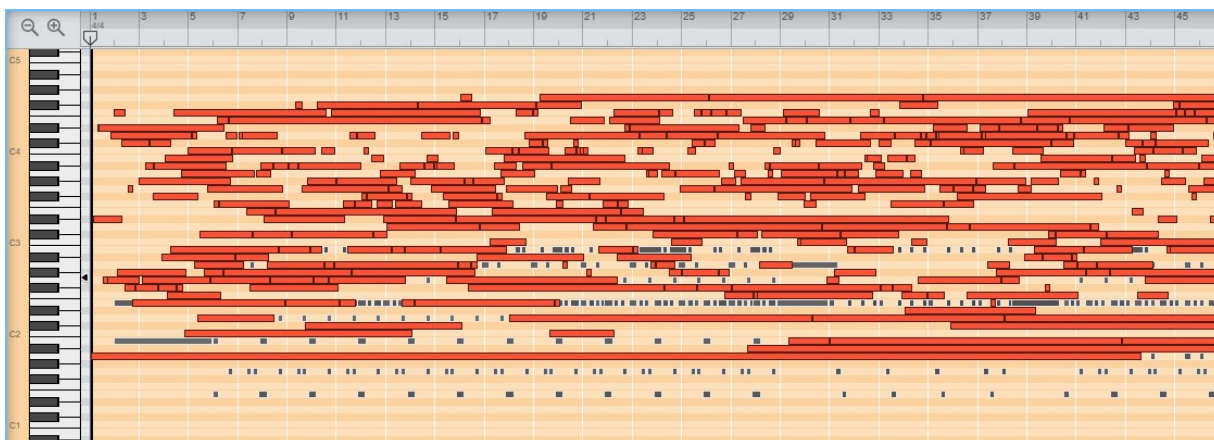


Figura 4.9: Comparativa visual entre una canción real (notas en color gris) y una secuencia generada por el modelo (notas en color naranja). Se observa que la densidad de notas es superior debido a la ausencia de restricciones tonales durante la generación

A pesar de los ajustes realizados, no se observó una variación significativa en la calidad de los resultados obtenidos. Las secuencias generadas capturan patrones rítmicos y armónicos, sin embargo, presentan diferencias notables respecto a las composiciones originales del corpus. Esta falta de coincidencia se atribuye principalmente a que el modelo genera secuencias musicales en cualquier tonalidad sin restricción, mientras que las canciones del corpus estaban limitadas a tonalidades específicas. Esta libertad tonal introduce una mayor densidad de notas en las composiciones generadas, comparada con la estructura más controlada y tonalmente coherente de las canciones reales.

No obstante, es importante señalar que esta alta densidad de notas es corregida durante la fase de generación final, donde se aplican técnicas de filtrado y selección de eventos musicales relevantes, logrando así obtener fragmentos musicales más claros, definidos y apropiados para su uso en procesos de composición asistida.

La Figura 4.10 muestra la evolución de la función de pérdida durante el entrenamiento y la validación del modelo LSTM. Esta evaluación permite visualizar el proceso de aprendizaje del modelo y su capacidad de generalización.

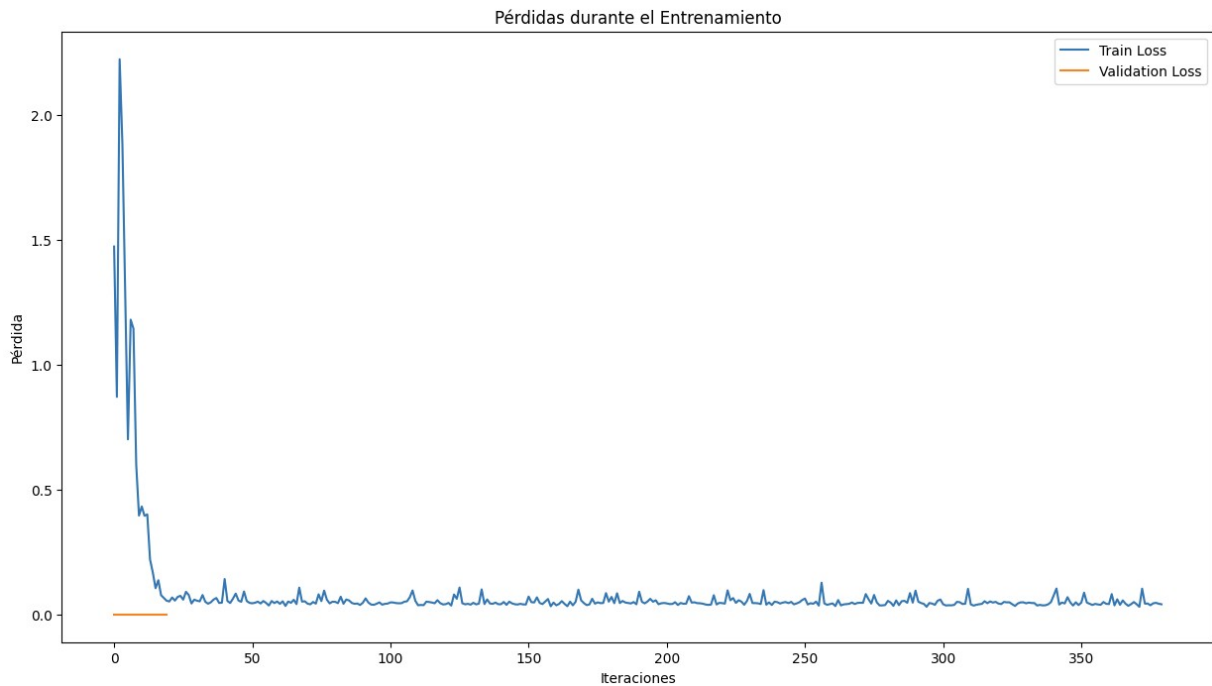


Figura 4.10: Gráfica de pérdida durante el entrenamiento y validación del modelo. Se observa una rápida disminución de la pérdida de entrenamiento, seguida de una estabilización, sin indicios claros de sobreajuste.

Se pueden destacar los siguientes aspectos clave del comportamiento observado:

- **Convergencia inicial rápida:** Durante las primeras iteraciones, la pérdida del conjunto de entrenamiento disminuye bruscamente, lo que indica que el modelo logra capturar patrones relevantes desde las primeras épocas.
- **Estabilización de la pérdida:** Tras la fase inicial de aprendizaje, la pérdida de entrenamiento se estabiliza en valores bajos y constantes, lo que sugiere que el modelo ha alcanzado una región de convergencia en el espacio de parámetros.
- **Comportamiento de la pérdida de validación:** La pérdida de validación se mantiene baja y sin incrementos significativos, lo que indica una adecuada capacidad de generalización del modelo. No se presentan síntomas evidentes de sobreajuste.

Conclusión: El comportamiento de las curvas de pérdida sugiere que el modelo ha sido entrenado de manera estable y eficiente, logrando un buen ajuste sobre los datos de entrenamiento sin comprometer su rendimiento en los datos de validación. Esto respalda la idoneidad de la arquitectura propuesta y de los hiperparámetros seleccionados.

El código detallado de la fase de entrenamiento se encuentra en la sección .5 (Fase de Entrenamiento) en el **Apendice A**.

4.1.4. Fase de Generación

Una vez completada la fase de entrenamiento, el modelo LSTM entrenado se utiliza para generar nuevas secuencias musicales. Esta etapa comienza con la carga del modelo previamente guardado (`lstm_guitar.pth`) mediante la clase `MusicLSTM`, en modo evaluación (`eval()`), asegurando que no se apliquen mecanismos como el *dropout* durante la inferencia.

Selección de tonalidad

Previo a la generación, se define una serie de escalas musicales tanto mayores como menores, codificadas como listas de valores enteros correspondientes a números MIDI. Estas listas actúan como filtros armónicos durante el proceso de generación y edición de las secuencias. Cada escala contiene las notas repetidas a lo largo del registro útil del modelo (notas entre 21 y 98), con el objetivo de garantizar que las notas generadas se mantengan dentro de un marco tonal definido. Este paso resulta fundamental para corregir uno de los principales problemas detectados durante la fase de entrenamiento: la alta densidad de notas producida por el modelo. Al restringir las notas generadas a una escala concreta, se eliminan activaciones disonantes o aleatorias fuera de contexto, logrando así secuencias más claras, musicales y tonalmente coherentes. Figura 4.11

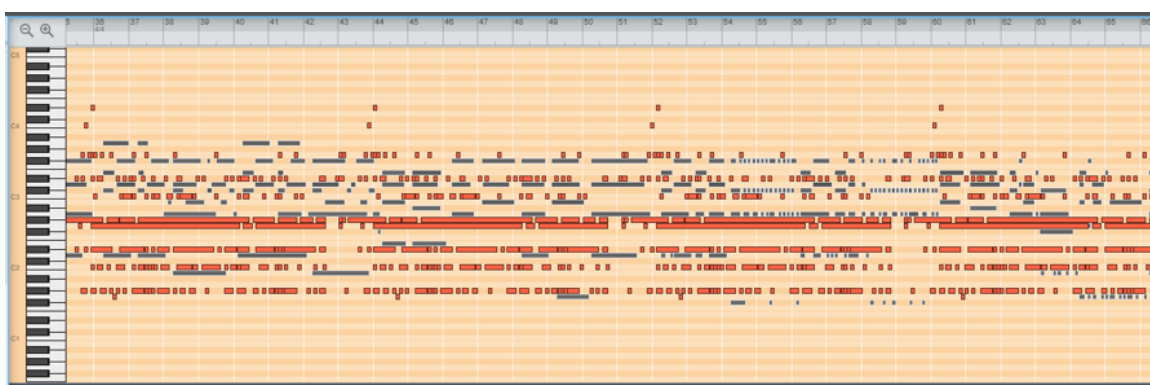


Figura 4.11: Visualización de una secuencia generada después de aplicar el filtrado por escala tonal. A diferencia de las versiones previas, se observa una reducción significativa en la densidad de notas, resultando en una disposición más clara, coherente y estructurada. Las notas naranjas representan la salida del modelo corregida armónicamente, mientras que las grises corresponden a una canción real del corpus de entrenamiento.

Composición estructurada mediante bloques

La función `generate_music()` permite controlar la estructura de la canción generada, estableciendo parámetros como:

- Número de versos iniciales (`initialVerses`)
- Número de versos intermedios (`middleVerses`)
- Cantidad de coros (`choruses`)
- Repeticiones del bloque intermedio (`middleRepeats`)
- Versos finales (`finalVerses`)
- Longitud de cada sección (en notas)
- Nivel de creatividad para verso y coro (factor de temperatura)

Internamente, esta función invoca a `generate_sample()`, que genera una secuencia de activaciones nota a nota. En cada paso, el modelo produce una distribución de probabilidad sobre las 77 notas MIDI posibles. Se aplica la función *softmax* con temperatura controlada por el parámetro `creativity`, lo que permite ajustar el nivel de aleatoriedad.

En la Figura 4.12 se observa una comparación entre dos secuencias generadas por el modelo. La primera (a) muestra una estructura desorganizada, con notas distribuidas de forma aleatoria y sin patrones evidentes. En contraste, la segunda secuencia (b) refleja una organización cíclica clara, resultado de incorporar una lógica estructural en el proceso de generación. Esta estructura, compuesta por partes repetitivas similares a las secciones de una canción (versos, estribillos, etc.), emula el formato común de una pieza musical en el género Rock-Pop, lo cual representa un avance significativo en la calidad compositiva del modelo.

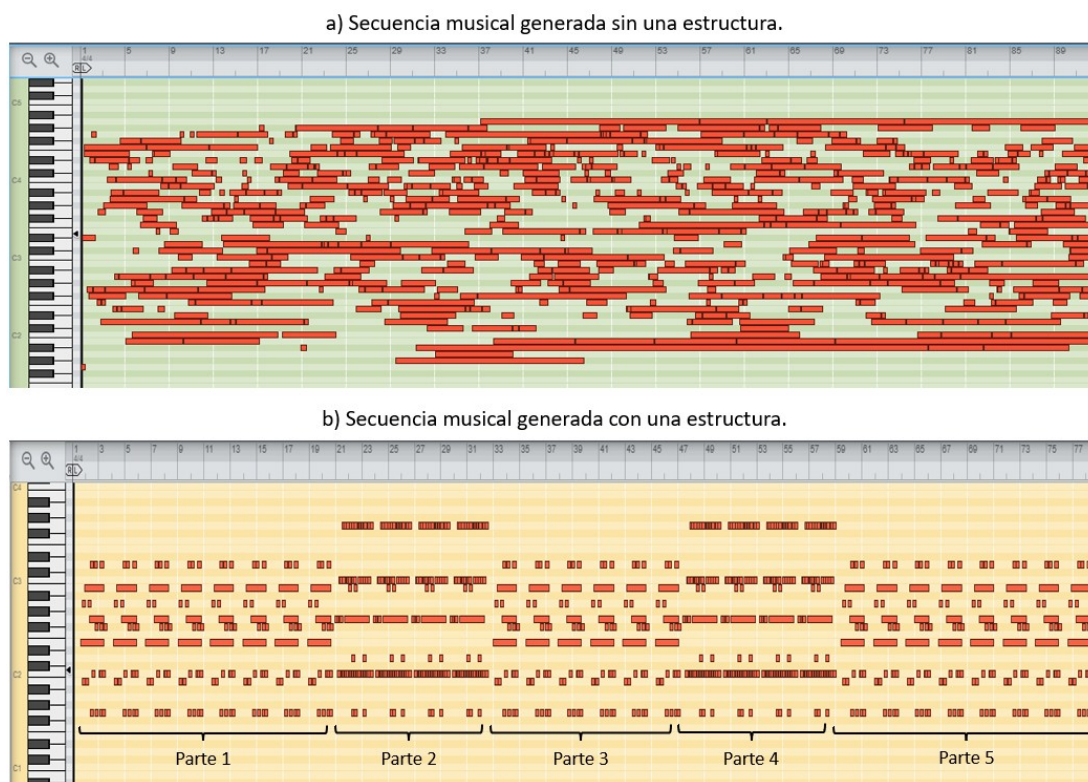


Figura 4.12: Comparación entre secuencias musicales generadas por el modelo. (a) Secuencia generada sin una estructura definida, con distribución aleatoria de notas. (b) Secuencia generada con estructura musical cíclica, compuesta por repeticiones organizadas que simulan partes típicas de una canción como versos o estribillos.

Conversión a formato MIDI

Una vez generada la secuencia, se utiliza la función `get_notesArray()` para traducirla a un arreglo de eventos MIDI con información sobre tipo de evento (`note_on`/`note_off`), nota, tiempo relativo y canal. Posteriormente, se aplica la función `get_notesArrayEdited()`, que ajusta las notas generadas a la escala seleccionada. Esto se realiza buscando, para cada nota fuera de la escala, la nota más cercana dentro de la escala objetivo.

Evaluación de similitud musical

Para evaluar qué tan parecida es la composición generada a las canciones reales del corpus, se aplica una métrica de similitud tolerante. Este análisis compara la secuencia MIDI generada con una muestra aleatoria de archivos MIDI reales. La métrica toma en cuenta coincidencias

en tipo de evento, nota (con tolerancia de hasta 4 semitonos) y tiempo (con tolerancia de hasta 7680 ticks, aproximadamente un compás), calculando un puntaje promedio de coincidencia.

Se comparan tanto la secuencia sin editar (`notes_array`) como la secuencia editada conforme a la escala (`notes_arrayEdited`). Los resultados muestran una mejora significativa tras la edición armónica, con un aumento en el puntaje de similitud promedio. Figura 4.13

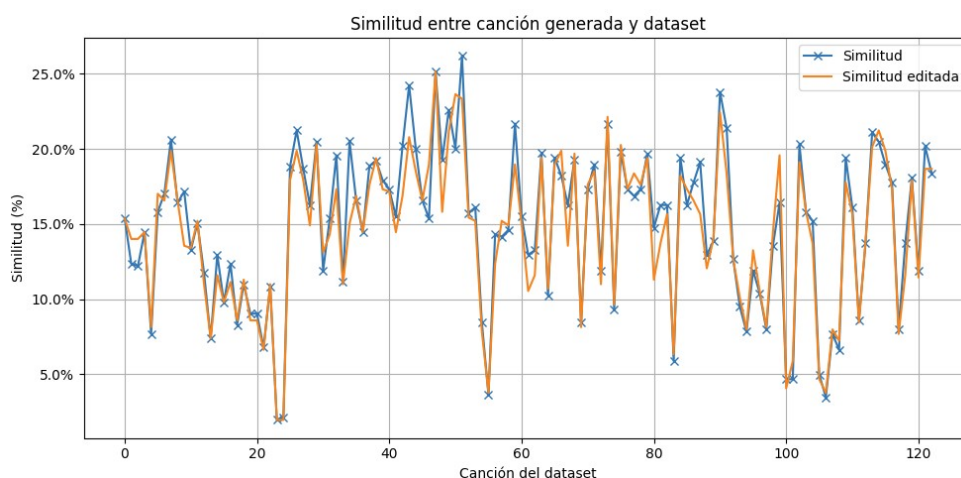


Figura 4.13: Comparación de similitud entre una canción generada por el modelo y las canciones reales del dataset. La línea azul representa la similitud directa, mientras que la línea naranja indica la similitud tras ajustar las notas generadas a una escala musical. Se observa un incremento general en la similitud al aplicar esta corrección, lo que indica que la edición tonal mejora la coherencia respecto al corpus original.

Capítulo 5

Resultados

En este capítulo se presentan los resultados obtenidos durante la fase de generación de la red neuronal LSTM diseñada para crear armonías musicales en el género Rock-Pop. Los resultados obtenidos se analizan en términos de su calidad musical, coherencia armónica y creatividad. La evaluación cualitativa del modelo no se limitó únicamente a la escucha pasiva de las secuencias generadas, sino que incluyó diversas estrategias prácticas orientadas a valorar su utilidad creativa en un contexto musical real.

5.1. Análisis musical mediante un DAW (*Digital Audio Work Station*) - Reason 13

Durante el análisis de las secuencias en el software musical **Reason 13**, uno de los aspectos más notables fue la presencia de una estructura cíclica. Al igual que en las canciones del conjunto de entrenamiento, las composiciones producidas por el modelo muestran secciones que se repiten de forma reconocible, lo cual remite a formas comunes como verso, precoro, coro o puente. Esta repetición no es aleatoria, sino que parece obedecer a una lógica interna que organiza el material musical en bloques funcionales. Esto sugiere que el modelo logró aprender no solo relaciones entre eventos musicales inmediatos, sino también patrones de organización formal presentes en el corpus. Figura 5.1

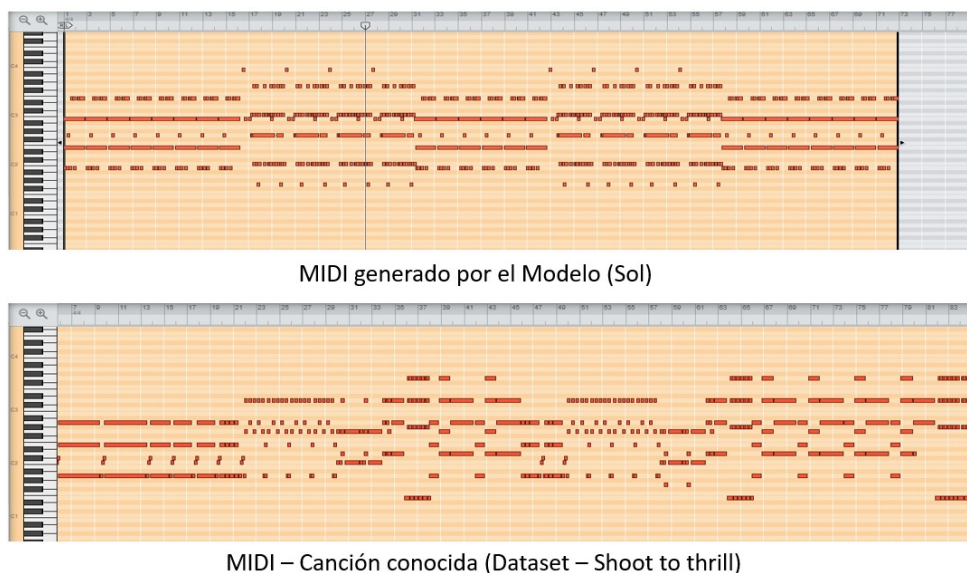


Figura 5.1: Las secuencias musicales generadas por el modelo presentan una estructura cíclica al igual que en las canciones del conjunto de entrenamiento

A pesar de que el contenido melódico o rítmico pueda variar entre cada generación, la existencia de estas macroestructuras refuerza la idea de que el modelo es capaz de generar material con una lógica formal coherente.

En este sentido, el modelo funciona no solo como un generador de notas o frases aisladas, sino como una herramienta compositiva que ofrece bloques musicales estructurados. El usuario puede seleccionar, reorganizar o reinterpretar estos bloques como punto de partida para desarrollar composiciones completas, aprovechando la repetición funcional como base estructural.

5.2. Comparación de tonalidad entre el modelo y la plataforma AIVA

Con el objetivo de evaluar la coherencia tonal de las secuencias generadas por el modelo LSTM, se diseñó una prueba de comparación entre la tonalidad definida por el modelo y la tonalidad detectada por AIVA, una plataforma de inteligencia artificial especializada en composición musical. Esta prueba buscó verificar si las piezas generadas mantenían una estructura tonal clara y coherente.

Se generaron múltiples archivos MIDI con diferentes tonalidades. Posteriormente, estos ar-

chivos se analizaron mediante la herramienta de detección tonal de AIVA. Se compararon ambas tonalidades (la del modelo y la detectada por AIVA) y se evaluaron bajo tres criterios: coincidencia directa, equivalencia enharmónica o parentesco funcional (como tonalidades relativas o paralelas).

Se observó que en numerosos casos AIVA detectó tonalidades que no coincidían literalmente con las esperadas, pero que sí eran *equivalentes enharmónicas* o *relativas modales*. Esto refleja que el modelo, aunque no siempre produce una coincidencia directa, genera material tonalmente coherente.

Cabe señalar que la propia herramienta AIVA reporta una precisión de aproximadamente un **80%** en la detección automática de tonalidades. Esto implica que algunas discrepancias podrían deberse no a un fallo del modelo, sino a *limitaciones en la capacidad de análisis tonal* de AIVA. Figura 5.2

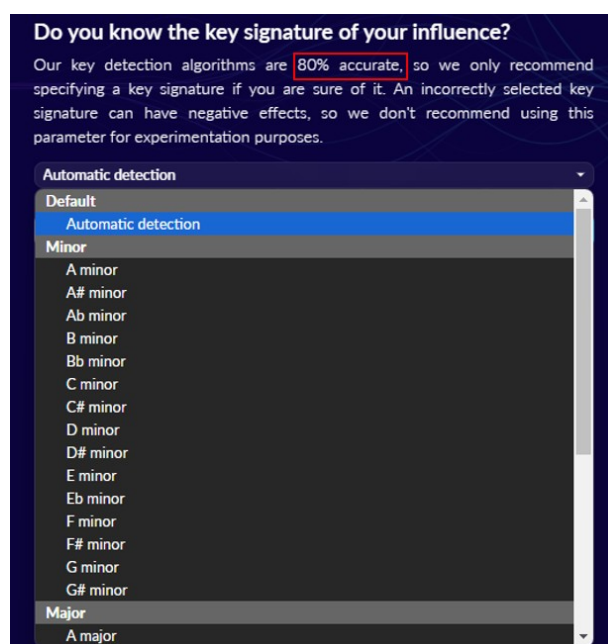


Figura 5.2: Interfaz de AIVA mostrando la opción de detección automática de tonalidad, la cual cuenta con una precisión aproximada del 80%.

La comparación entre las tonalidades generadas por el modelo y las detectadas por AIVA permitió validar que, en la mayoría de los casos, el modelo produce secuencias con sentido tonal claro. Aunque no siempre coinciden literalmente con la tonalidad identificada por AIVA, la relación entre ambas suele estar justificada desde un punto de vista musical. La Tabla 5.1 aporta evidencia adicional de que el modelo genera material armónicamente estructurado

5.2. COMPARACIÓN DE TONALIDAD ENTRE EL MODELO Y LA PLATAFORMA AIVA

y útil para procesos de composición asistida por IA con una certeza de al menos **67%** de precision tonal (puede ser mayor si se toma en cuenta el error de AIVA).

Tabla 5.1: Resumen porcentual de equivalencias tonales entre el modelo y AIVA

Tipo de equivalencia	Casos	Porcentaje (%)
No equivalente	33	33.0
Coincidencia directa	18	18.0
Relativa	17	17.0
Enarmónica	5	5.0
Casi equivalente	27	27.0

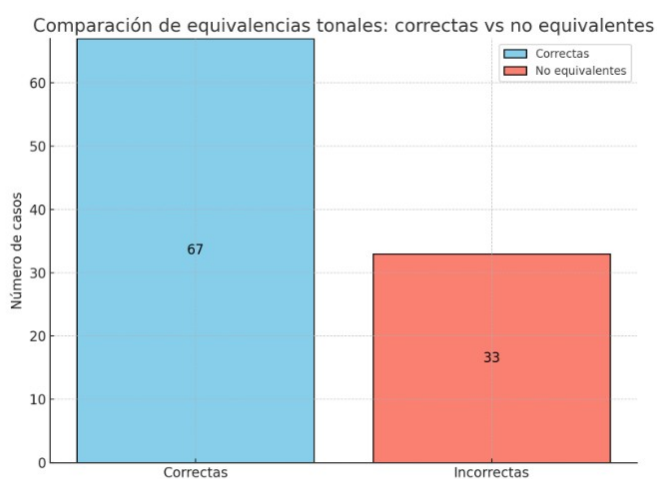


Figura 5.3: Distribución de tipos de equivalencia tonal entre el modelo y AIVA

Los resultados detallados de esta evaluación se pueden consultar en el **Apéndice B. Comparación completa de tonalidades entre el modelo y AIVA**. .5 (Apéndice B. Comparación de tonalidades entre el modelo y AIVA)

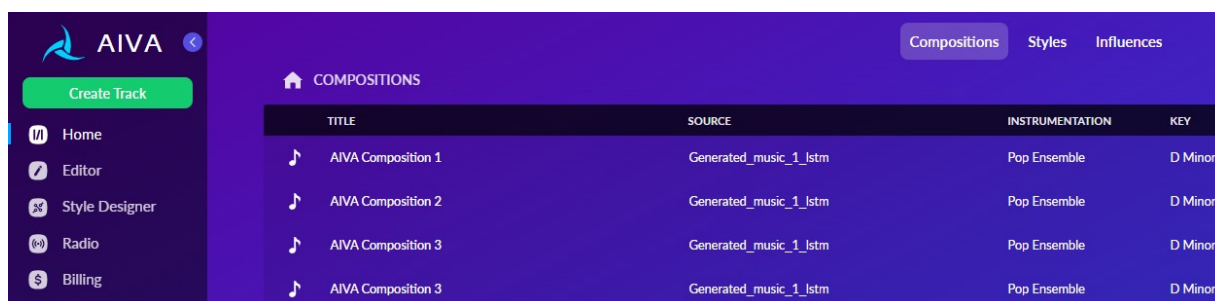
5.3. Transformación con la aplicación AIVA

Como parte del análisis cualitativo, se exploró la posibilidad de utilizar las secuencias generadas por el modelo como entradas para otras herramientas de composición asistida. En particular, se realizó una prueba utilizando la plataforma AIVA, una inteligencia artificial especializada en la creación automática de música.

AIVA (*Artificial Intelligence Virtual Artist*) permite generar piezas musicales de diversas maneras: a partir de estilos predefinidos, parámetros personalizados o incluso tomando como referencia una canción cargada por el usuario. Esta última modalidad fue la utilizada en el presente experimento, en el cual se le proporcionó a AIVA una de las secuencias generadas por el modelo LSTM como entrada para producir una nueva composición.

El resultado fue sorprendentemente positivo: la salida generada por AIVA no solo conservaba las ideas musicales originales, sino que también corregía varios de los aspectos que hacían difícil su utilización directa, como la sobrecarga de información o la falta de silencios. Además, AIVA no se limitó a reorganizar el material existente, sino que añadió nuevos instrumentos como parte de su proceso creativo, generando acompañamientos, líneas melódicas y armonizaciones que no estaban presentes en la entrada original. Este enriquecimiento automático contribuyó a que el resultado final sonara más cercano a una canción completa.

Este hallazgo sugiere que el modelo LSTM, puede actuar como una *fuentes generadora de ideas musicales primarias* que pueden ser refinadas posteriormente mediante otras herramientas creativas basadas en inteligencia artificial. En conjunto, esta prueba destaca el potencial del modelo no solo como un generador autónomo, sino que tiene el potencial de funcionar como una primera capa creativa dentro de un flujo de trabajo más amplio, en el cual la colaboración entre diferentes inteligencias artificiales y el criterio humano pueden converger para producir resultados musicales más elaborados y útiles.



The screenshot shows the AIVA web application interface. On the left is a dark sidebar with the AIVA logo and navigation links: Home, Editor, Style Designer, Radio, and Billing. The main area has a top navigation bar with 'Compositions', 'Styles', and 'Influences'. Below this is a table titled 'COMPOSITIONS' with four columns: TITLE, SOURCE, INSTRUMENTATION, and KEY. The table lists four compositions, all generated from 'Generated_music_1_lstm' and featuring 'Pop Ensemble' instrumentation in 'D Minor'.

TITLE	SOURCE	INSTRUMENTATION	KEY
AIVA Composition 1	Generated_music_1_lstm	Pop Ensemble	D Minor
AIVA Composition 2	Generated_music_1_lstm	Pop Ensemble	D Minor
AIVA Composition 3	Generated_music_1_lstm	Pop Ensemble	D Minor
AIVA Composition 3	Generated_music_1_lstm	Pop Ensemble	D Minor

Figura 5.4: Composiciones generadas en AIVA a partir de secuencias producidas por el modelo LSTM.

5.4. Generación de atmósferas musicales

Con el objetivo de evaluar la capacidad del modelo para generar música más allá de la estructura tradicional de una canción, se realizó un experimento de sonorización aplicado al ámbito cinematográfico. Para ello, se seleccionó la escena de una película de terror, de la cual se eliminó la música pero se conservaron los diálogos y efectos sonoros del audio original.

Posteriormente, se integró una de las secuencias generadas por el modelo como acompañamiento sonoro. Aunque estas secuencias fueron originalmente concebidas para ser ejecutadas con guitarra, en esta prueba se utilizó el software *Reason* para asignarles una instrumentación de piano, lo cual resultó en una sonoridad mucho más adecuada para la escena. La edición y sincronización final se realizaron utilizando el programa *CapCut*, lo que permitió alinear la música generada con los eventos clave de la escena sin alterar el ritmo narrativo.

El resultado fue notable: la nueva versión de la escena transmitía una atmósfera más oscura y opresiva que la original, generando mayor tensión y dramatismo. A pesar de no haber sido creada explícitamente para ese propósito, la pista generada por el modelo logró adaptarse de forma natural al contenido visual, acentuando los momentos de suspenso con texturas sonoras inquietantes y emocionalmente efectivas.

Este experimento demostró que el modelo no solo es capaz de generar secuencias musicales coherentes, sino también de contribuir significativamente a la creación de **atmósferas expresivas**, ampliando su potencial de aplicación hacia áreas como la música incidental, el cine y otros medios audiovisuales. Además, se comprobó que las secuencias generadas pueden ser interpretadas por distintos instrumentos, como el piano, lo cual abre la posibilidad de reutilización creativa en distintos contextos estilísticos.

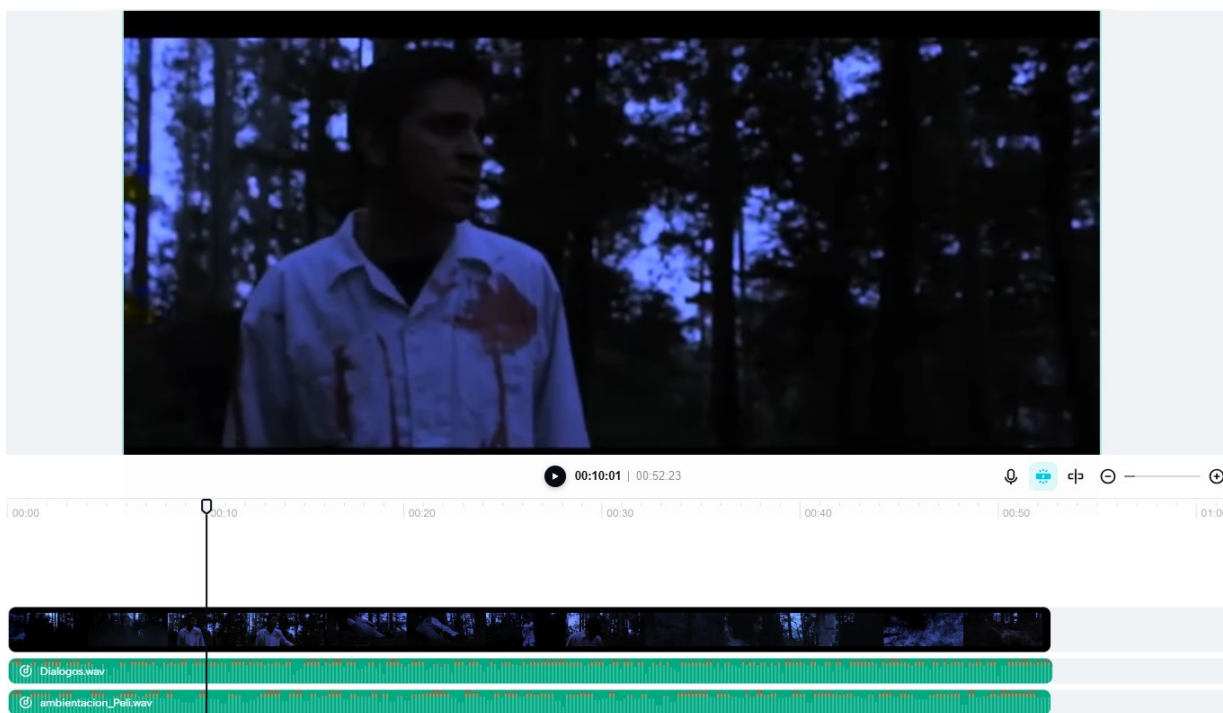


Figura 5.5: Escena de terror editada en el software CapCut. Se observan dos pistas de audio: los diálogos originales (superior) y la ambientación musical generada por el modelo LSTM (inferior).

Capítulo 6

Conclusiones y recomendaciones

Con base en los resultados obtenidos a lo largo del desarrollo de esta investigación, se puede concluir que los objetivos planteados fueron alcanzados de manera satisfactoria, y que la hipótesis propuesta ha sido validada mediante evidencias cualitativas y cuantitativas.

6.1. Conclusiones

- Se diseñó e implementó un modelo de red neuronal recurrente tipo LSTM para la generación de secuencias armónicas de guitarra y bajo en el estilo Rock-Pop. El modelo fue entrenado con una base de datos en formato MIDI previamente preparada, cumpliendo así con los primeros objetivos específicos.
- A lo largo del entrenamiento, se observó que el modelo es capaz de generar secuencias con una estructura tonal definida, lo cual fue confirmado mediante análisis comparativos con la herramienta AIVA. La evaluación mostró una alta tasa de coincidencia tonal (alrededor del 70% considerando coincidencias directas, relativas o enharmónicas), lo que indica que el modelo aprende patrones armónicos coherentes y estilísticamente adecuados al género musical de referencia.
- En la evaluación subjetiva, se identificaron múltiples frases musicales originales dentro de las composiciones generadas, que si bien requerían edición posterior, presentaban potencial para ser utilizadas como material creativo. Se concluye que el modelo puede ser una herramienta útil en el proceso compositivo, especialmente como generador de ideas armónicas y texturas musicales.

- Asimismo, se demostró que las secuencias generadas pueden ser utilizadas en contextos más allá del formato de canción tradicional. Uno de los hallazgos más interesantes fue su capacidad para ambientar una escena de terror, generando una atmósfera más densa y efectiva que el audio original, lo cual valida el objetivo relacionado con la exploración de nuevos contextos musicales.
- A pesar de que el modelo no genera composiciones completas listas para publicación, sí produce material aprovechable, confirmando así la hipótesis de que una red neuronal entrenada con canciones Rock-Pop puede generar armonías estilísticamente consistentes y útiles para la creación musical.

6.2. Recomendaciones

- Se recomienda ampliar el número de ejemplos en el conjunto de datos de entrenamiento, con el fin de mejorar la generalización del modelo. En este sentido, se contó con la colaboración de Xanthe de la Rosa, quien desarrolló una red neuronal clasificadora de instrumentos que permitió automatizar parcialmente la separación de canales MIDI y agilizar el proceso de preparación del dataset; sin embargo, debido a limitaciones de tiempo, dicha herramienta no fue implementada de manera completa en la presente versión del sistema (De la Rosa, 2025).
- Es aconsejable implementar una etapa de postprocesamiento automático para filtrar o refinar las secuencias generadas, a fin de reducir la densidad de notas o ajustar la métrica al compás deseado.
- Se sugiere explorar arquitecturas más modernas como Transformers, que podrían mejorar la capacidad del modelo para generar secuencias más estructuradas y coherentes a largo plazo.
- Desde el punto de vista práctico, se recomienda el uso del modelo como herramienta creativa para compositores, más que como generador automático de piezas completas. Su valor radica en ofrecer inspiración, motivos musicales y estructuras armónicas iniciales que pueden ser desarrolladas manualmente.
- Finalmente, se propone integrar esta solución en plataformas educativas para fomentar el aprendizaje de estructuras armónicas, secuenciación MIDI y aplicaciones de inteligencia artificial en la música.

Referencias

- Abu-Mostafa, Y. S. (2012). *Learning from data* (H.-T. Lin, Ed.). AML Book.
- Aiva - your personal ai music generation assistant. (2024). Descargado de <https://www.aiva.ai>
- Aäron van den Oord, S. D. (2016, diciembre). *Wavenet: A generative model for raw audio*. Descargado de <https://arxiv.org/abs/1609.03499v2>
- Chollet, F. (2020). *Deep learning with python* (F. Buontempo, Ed.). Manning Publications Co.
- Chuan C.-H., . H. D. (2018). Modeling temporal tonal relations in polyphonic music through deep networks with a novel image-based representation. <https://doi.org/10.1609/aaai.v32i1.11880>.
- De la Rosa, X. (2025). *Entrenamiento de una red neuronal con un repositorio (dataset) de archivos midi* (Inf. Téc.). Instituto Tecnológico de Toluca.
- Ghosh, T. (2023). *Practical mathematics for ai and deep learning* (K. Bhattacharyya, Ed.). BPB Publications.
- Harrington, P. (2012). *Machine learning in action* (J. Bleiel, Ed.). Manning Shelter Island.
- Haykin, S. (2005). *Neuronal networks: A comprehensive foundation*. Pearson education.
- <https://openai.com/index/jukebox/>. (2019, abril).
- Huang, C. A., Cooijmans, T., Roberts, A., Courville, A. C., y Eck, D. (2019). Counterpoint by convolution. *CoRR*, abs/1903.07227. Descargado de <http://arxiv.org/abs/1903.07227>
- Jean-Pierre Briot, F.-D. P. (2019, agosto). *Deep learning techniques for music generation – a survey*.
- Lim H., R. S. . L. K. (2017). Chord generation from symbolic melody using blstm networks. *preprint arXiv:1712.01011..*
- MORALES, A. F. (2025). *Manipulación de archivos midi para crear un dataset de música* (Inf. Téc.). Instituto Tecnológico de Toluca.
- Nagy, Z. (2018). *Artificial intelligence and machine learning fundamentals*. Packt Publishing. Descargado de <https://books.google.com.mx/books?id=htH8vQEACAAJ>
- Nilsson, N. J. (2009). *The quest for artificial intelligence* (1st ed.). USA: Cambridge University Press.
- Open art. (2024). Descargado de <https://openart.ai>
- PICKOVER, C. A. (2021). *Inteligencia artificial - una historia ilustrada*. Librero.
- Raff, E. (2022). *Inside deep learning* (F. Lefkowitz, Ed.). Manning Publications Co.
- Tellier, M. (2024). *Music theory simplified* (M. Tellier, Ed.). Musiciangoods.

Tesla, N. (2022). *El problema del aumento de la energía humana. con referencias especiales al aprovechamiento de la energía solar*. StreetLib.

ToDo : *Incrementar referencias bibliográficas*

Apéndice A. Modelo de Red Neuronal LSTM para la generación de armonías musicales - Código

El código fuente presentado en este apéndice fue desarrollado por Abraham Flores como parte de su proyecto de residencia profesional, llevado a cabo en el marco del presente trabajo de tesis. La implementación se basa en el algoritmo armónico propuesto en este documento y se encuentra documentada en (MORALES, 2025).

.1. Codificación de los archivos MIDI en Tensores - Código

El código para convertir los archivos MIDI a arreglos matriciales se muestra a continuación ¹:

```
1 def load_piano_rolls(directory_path):
2     piano_lists = []
3     max_len = 0
4     for filename in os.listdir(directory_path):
5         if filename.endswith(".mid"):
6             file_path = os.path.join(directory_path, filename)
7             mid = pretty_midi.PrettyMIDI(file_path)
8             piano_roll = mid.get_piano_roll(fs=5)
9             max_len = max(max_len, piano_roll.shape[1]) # max_len es usada para conocer
10                  la matriz mas grande
11             piano_roll = piano_roll[21:98] # 21 - 98 es el rango de notas audibles de la
12                  guitarra
13             piano_roll[piano_roll > 0] = 1
```

¹Nota: Explicaremos únicamente el código de la guitarra, que es equivalente al del bajo, con diferencias solo en los parámetros específicos de cada instrumento.

```
12         piano_lists.append(piano_roll)
13     return piano_lists, max_len
14
15 # Función para obtener la matriz de cada midi, dependiendo la carpeta
16 list_of_piano_rolls, max_len = load_piano_rolls("Guitarras y bajos//Guitarras//")
```

Propósito: Cargar los archivos MIDI desde un directorio, convertirlos en matrices y ajustar su rango de notas al de una guitarra. También se calcula la longitud máxima de las secuencias para usarlas más adelante.

Entrada:

directory_path: Ruta al directorio que contiene los archivos MIDI.

Salida:

piano_lists: Lista de las matrices correspondientes a los archivos MIDI

max_len: La longitud de la matriz más larga.

.2. Proceso de padding para los archivos MIDI transformados a Tensores:

```
1 def padNotes(sequence, maxlen, roll_len, pad_value):
2     paddedRoll = np.zeros((77, maxlen))
3     paddedRoll[:] = pad_value
4     paddedRoll[:, -roll_len:] = sequence
5     return paddedRoll.transpose()
6
7 class MusicDataset(Dataset):
8     def __init__(self, tensor_list, maxlen):
9         self.data = tensor_list
10        self.max_len = maxlen
11
12    def __len__(self):
13        return len(self.data)
14
15    def __getitem__(self, idx):
16        piano_roll = self.data[idx]
17        sequence_len = piano_roll.shape[1] - 1
18        input_sequence = piano_roll[:, :-1] # La secuencia de entrada es la matriz del
19                                           archivo MIDI menos la ultima nota
20        target_sequence = piano_roll[:, 1:] # La secuencia esperada de salida es toda la
21                                           matriz menos la primera nota
```

```
20     # Es decir, se espera que la salida de una nota sea la siguiente de la secuencia
    original
21
22     input_sequence = padNotes(input_sequence, self.max_len, sequence_len, 0)
23     target_sequence = padNotes(target_sequence, self.max_len, sequence_len, -100)
24
25     return (torch.FloatTensor(input_sequence), torch.LongTensor(target_sequence),
            torch.LongTensor([sequence_len]))
```

Propósito: padNotes: Asegura que todas las secuencias (matrices) tengan la misma longitud agregando padding (relleno) de ceros.

MusicDataset: Organiza los datos en forma de dataset para que sean fácilmente accesibles por PyTorch. Prepara las secuencias de entrada y las secuencias objetivo (target) con un padding adecuado. Para ello, usamos la función padNotes, que rellena las secuencias de entrada con ceros y las secuencias objetivo con un valor de -100 (esto es importante durante el cálculo de la pérdida, ya que PyTorch ignora los valores de -100 al evaluar el rendimiento del modelo).

Entrada:

sequence (en padNotes): Matriz que se quiere rellenar.

tensor_list (en MusicDataset): Lista de matrices a procesar.

maxlen: Longitud máxima de las matrices.

Salida:

padNotes: Matrices rellenas a la longitud máxima.

MusicDataset: Conjunto de tensores que incluye:

- input_sequence: Secuencia de entrada (matriz ajustada).
- target_sequence: Secuencia objetivo (la siguiente nota en la matriz).
- sequence_len: Longitud real de la matriz antes del padding.

.3. División del *Dataset* en lotes (*batches*) de entrenamiento y validación

```
1 indexes = list(range(len(list_of_piano_rolls))) # lista de 0 a n donde n representa la
    cantidad de archivos midi
2 random.shuffle(indexes) # se mezclan para que la muestra sea aleatoria
```

.4. DEFINICIÓN DE LA ARQUITECTURA DEL MODELO LSTM PARA GENERACIÓN MUSICAL

```
3 validation_len = int(len(list_of_piano_rolls)*0.8) # se toma el 80% para entrenamiento
   y el 20% para validaci'on
4 index_train = indexes[]
5 index_validation = indexes[validation_len:]
6 train_tensors = [list_of_piano_rolls[i] for i in index_train]
7 validation_tensors = [list_of_piano_rolls[i] for i in index_validation]
8
9 train_dataset = MusicDataset(train_tensors, max_len)
10 train_dataloader = DataLoader(train_dataset, batch_size=5, shuffle=True, drop_last=True)
11 # Se crea un lote de 5 tuplas que contienen (InputTensor, OutputTensor, SequenceLen) cada
   una
12 validation_dataset = MusicDataset(validation_tensors, max_len)
13 validation_dataloader = DataLoader(train_dataset, batch_size=3, shuffle=True, drop_last=
   True)
14 #Se crea un lote de 3 tuplas que contienen (InputTensor, OutputTensor, SequenceLen) cada
   una
```

Propósito:

Dividir el dataset en conjuntos de entrenamiento y validación, mezclándolos aleatoriamente. A continuación, se crean lotes (batches) de datos para su uso en el entrenamiento y la validación del modelo.

Entrada:

`list_of_piano_rolls`: Lista de secuencias matriciales de los archivos MIDI.

`max_len`: Longitud máxima de las secuencias.

Salida:

`train_dataloader`: Lotes de datos de entrenamiento, cada uno compuesto por 5 tuplas con secuencias de entrada, secuencias objetivo y longitudes de las secuencias.

`validation_dataloader`: Lotes de datos de validación, cada uno compuesto por 3 tuplas con secuencias de entrada, secuencias objetivo y longitudes de las secuencias.

.4. Definición de la arquitectura del modelo LSTM para generación musical

Esta parte del código implementa una red LSTM diseñada para predecir secuencias de notas a partir de las matrices de los archivos MIDI. La LSTM es un tipo de red neuronal recurrente (RNR)

.4. DEFINICIÓN DE LA ARQUITECTURA DEL MODELO LSTM PARA GENERACIÓN MUSICAL

que es ideal para capturar dependencias temporales en secuencias de datos. En este caso, se utiliza para aprender a predecir la siguiente nota en una secuencia de notas de guitarra, basándose en el contexto de las notas previas.

```
1 class MusicLSTM(nn.Module):
2     def __init__(self, input_size, hidden_size, number_cells, layers):
3         super(MusicLSTM, self).__init__()
4         self.input_size = input_size
5         self.hidden_size = hidden_size
6         self.number_cells = number_cells
7         self.layers = layers
8
9         self.linear = nn.Linear(input_size, hidden_size)
10        self.normal = nn.BatchNorm1d(hidden_size)
11        self.lstm = nn.LSTM(hidden_size, hidden_size, layers)
12        self.linear2 = nn.Linear(hidden_size, number_cells)
13
14    def forward(self, input_sequence, input_sequence_lens, hidden=None):
15
16        output = self.linear(input_sequence)
17        output = output.permute(1,2,0).contiguous()
18
19        output = self.normal(output)
20        output = nn.Dropout(0.25)(output)
21        output = output.permute(2,0,1)
22
23        output_packed = torch.nn.utils.rnn.pack_padded_sequence(output,
24            input_sequence_lens)
25        output, hidden = self.lstm(output_packed, hidden)
26
27        output, output_lens = torch.nn.utils.rnn.pad_packed_sequence(output)
28        output = output.permute(1,2,0).contiguous()
29        output = self.normal(output)
30        output = nn.Dropout(0.1)(output)
31
32        output = output.permute(2,0,1)
33        logits = self.linear2(output)
34        logits = logits.transpose(0,1).contiguous()
35
36        rev_logits = (1 - logits)
37
38        zero_one_logits = torch.stack((logits, rev_logits), dim=3).contiguous()
39        flatten_logits = zero_one_logits.view(-1,2)
```

```
return flatten_logits, hidden
```

Objetivos clave del código:

- **Procesar secuencias musicales (piano rolls):** Cada piano roll representa un conjunto de notas activas a lo largo del tiempo. Este modelo toma esas secuencias como entrada y, utilizando una LSTM, genera predicciones sobre qué notas deberían tocarse a continuación.
- **Ajustar las secuencias para la red:** El código también se encarga de aplicar transformaciones y normalizaciones necesarias para hacer que los datos sean compatibles con la red y mejorar la estabilidad del entrenamiento.
- **Hacer predicciones binarias:** El modelo genera predicciones para cada nota, indicando si debe estar activa o no en cada paso de tiempo de la secuencia.

Inicialización del modelo (`__init__`):

El constructor de la clase `MusicLSTM` define las capas y parámetros necesarios para la red:

- **Capa lineal (`self.linear`):** Transforma la entrada de las secuencias (piano rolls) a un tamaño compatible con las capas ocultas del modelo.
- **Batch Normalization (`self.normal`):** Estabiliza el entrenamiento normalizando la activación de las capas, mejorando la eficiencia y evitando problemas de sobreajuste.
- **LSTM (`self.lstm`):** La capa LSTM central, que captura dependencias temporales en las secuencias. Esta capa procesa la secuencia de notas para predecir las siguientes notas basándose en el contexto anterior.
- **Capa lineal final (`self.linear2`):** Transforma la salida de la LSTM en probabilidades para predecir las notas en el siguiente paso de la secuencia.

Propagación hacia adelante (`forward`):

El método `forward` realiza el paso de inferencia en el modelo, es decir, cómo la red procesa las secuencias de entrada para generar las predicciones. El flujo básico es el siguiente:

- **Transformación de la entrada:**

.4. DEFINICIÓN DE LA ARQUITECTURA DEL MODELO LSTM PARA GENERACIÓN MUSICAL

- La secuencia de entrada pasa primero por una capa lineal para ajustar su tamaño.
- Luego se aplican normalización y dropout para mejorar la eficiencia y prevenir el sobreajuste.

■ **Procesamiento con LSTM:**

- Las secuencias se "empaquetan" para ser procesadas de manera eficiente por la LSTM, ignorando los elementos de padding (relleno).
- La LSTM procesa la secuencia empaquetada, generando una nueva representación basada en las dependencias temporales entre las notas.

■ **Desempaquetado de la secuencia:**

- Después de que la LSTM procesa la secuencia, esta se desempaqueta y se normaliza nuevamente para devolverla a su forma original.

■ **Predicción de las notas:**

- La salida de la LSTM pasa por una última capa lineal que genera predicciones para cada nota de la secuencia. Se utiliza una transformación para calcular las probabilidades de si una nota debe estar activa o no (binaria).
- Las predicciones finales indican, para cada momento en el tiempo, si una nota debe tocarse (1) o no (0).

Salida del modelo

El método `forward` devuelve dos cosas:

- `flatten_logits`: Una versión aplanada de las predicciones del modelo, donde cada fila contiene la probabilidad de que una nota esté activa o inactiva.
- `hidden`: El estado oculto de la LSTM, que puede reutilizarse en secuencias futuras, permitiendo que la red continúe procesando secuencias largas de forma eficiente.

Propósito final del código:

Este modelo MusicLSTM tiene como objetivo procesar secuencias de notas de guitarra (extraídas de los piano rolls) y predecir qué notas deben tocarse a continuación, basándose en el contexto de las notas previas. Utiliza una red LSTM para captar dependencias temporales entre las notas, lo que es fundamental para tareas como la generación automática de música, donde la estructura y la continuidad en el tiempo son esenciales.

.4. DEFINICIÓN DE LA ARQUITECTURA DEL MODELO LSTM PARA GENERACIÓN MUSICAL

El modelo se entrena con datos de entrada y salida derivados de piano rolls, donde la secuencia de salida esperada es simplemente la versión desplazada en el tiempo de la secuencia de entrada. Durante el entrenamiento, la red ajusta sus parámetros para que, dado un fragmento de una secuencia de notas, pueda predecir el siguiente fragmento.

Preparación y reordenamiento de un lote de secuencias

Esta función tiene como objetivo preparar y reordenar un lote de secuencias (piano rolls) para que sean procesadas de manera eficiente por una red LSTM durante la fase de entrenamiento. Se enfoca en ordenar las secuencias por longitud, ajustarlas para que todas tengan la misma dimensión, y optimizar su procesamiento en GPU.

```
1 def orderTensors(batch):
2
3     input_sequences, output_sequences, lens = batch
4
5     input_sequences = input_sequences.split(split_size=1)
6     output_sequences = output_sequences.split(split_size=1)
7     lens = lens.split(split_size=1)
8
9     train_data_tuples = zip(input_sequences, output_sequences, lens)
10    train_data_tuples = sorted(train_data_tuples, key=lambda c: int(c[2]), reverse=True)
11
12    input_sequences, output_sequences, lens = zip(*train_data_tuples)
13
14    ordered_inputs = torch.cat(input_sequences)
15    ordered_outputs = torch.cat(output_sequences)
16    ordered_lens = torch.cat(lens)
17
18    ordered_inputs = ordered_inputs[:, -ordered_lens[0, 0]:, :]
19    ordered_outputs = ordered_outputs[:, -ordered_lens[0, 0]:, :]
20    ordered_inputs = ordered_inputs.transpose(0, 1)
21
22    ordered_lens_list = list(ordered_lens)
23    ordered_lens_list = map(lambda k: int(k), ordered_lens_list)
24
25    return ordered_inputs.cuda(), ordered_outputs.contiguous().view(-1).cuda(), list(
        ordered_lens_list)
```

Propósito:

- Reordenar las secuencias por longitud de mayor a menor. Esto es necesario para optimizar el uso de la función `pack_padded_sequence` de PyTorch, que permite que la LSTM

.4. DEFINICIÓN DE LA ARQUITECTURA DEL MODELO LSTM PARA GENERACIÓN MUSICAL

procese secuencias de diferentes longitudes de manera eficiente, ignorando el *padding*.

- Ajustar las secuencias a la longitud de la secuencia más larga del lote, eliminando cualquier *padding* innecesario y reduciendo el número de cálculos que el modelo debe realizar.
- Mover las secuencias a la GPU (`.cuda()`) para aprovechar las capacidades de procesamiento paralelo y acelerar el entrenamiento.
- Facilitar el cálculo de la pérdida al aplanar las secuencias de salida, lo que permite que las funciones de pérdida como la entropía cruzada trabajen con las predicciones y las etiquetas de manera eficiente.

Entradas:

La función recibe un parámetro, `batch`, que contiene:

- `input_sequences`: Un tensor que representa las secuencias de entrada, cada una de las cuales es un piano roll con dimensiones `[batch_size, time_steps, features]`.
- `output_sequences`: Un tensor que contiene las secuencias de salida (la siguiente nota a predecir) con la misma estructura que las secuencias de entrada.
- `lens`: Un tensor que contiene las longitudes reales de las secuencias de entrada (sin contar el *padding*).

Salidas:

La función devuelve tres elementos:

- `ordered_inputs`: Un tensor de secuencias de entrada que han sido:
 - Reordenadas en función de la longitud de mayor a menor.
 - Recortadas para eliminar *padding* innecesario.
 - Movidas a la GPU usando `.cuda()`.
- `ordered_outputs`: Un tensor de secuencias de salida que:
 - Ha sido reordenado y ajustado de la misma forma que las secuencias de entrada.
 - Aplanado con `view(-1)` para ser compatible con las funciones de pérdida de PyTorch.
 - También movido a la GPU con `.cuda()`.

- `ordered_lens_list`: Una lista de las longitudes de las secuencias reordenadas, necesaria para otras funciones de PyTorch, como `pack_padded_sequence`, que trabaja con longitudes variables.

.5. Fase de Entrenamiento

Este código representa un ciclo de entrenamiento y evaluación de un modelo LSTM, llamado `MusicLSTM`, que está diseñado para trabajar con secuencias de piano rolls (representaciones musicales).

```
1 input_size = 77
2 hidden_size = 512
3 number_cells = 77
4 layers = 2
5
6 model = MusicLSTM(input_size, hidden_size, number_cells, layers).cuda()
7 criterion = nn.CrossEntropyLoss().cuda()
8 optimizer = optim.Adam(model.parameters(), lr=0.01)
9
10 num_epochs = 10
11 train_losses = []
12 validation_losses = []
13
14 for epoch in range(1, num_epochs+1):
15
16     # Training Loop
17     model.train()
18     for batch in train_dataloader:
19
20         input_sequence, output_sequence, sequence_len = orderTensors(batch)
21         optimizer.zero_grad()
22
23         output, _ = model(input_sequence, sequence_len)
24         train_loss = criterion(output, output_sequence)
25
26         train_loss.backward()
27         torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)
28         optimizer.step()
29
30         train_losses.append(train_loss.item())
31
```

```

32     # Evaluation Loop
33     model.eval()
34     val_loss_full = 0.0
35     seq_len = 0.0
36     for batch in validation_dataloader:
37         val_inputs, val_outputs, seq_lens = orderTensors(batch)
38         output, _ = model(val_inputs, seq_lens)
39         loss = criterion(output, val_outputs)
40         val_loss_full += loss.item()
41         seq_len += sum(seq_lens)
42
43     validation_loss = val_loss_full/(seq_len*77)
44     validation_losses.append(validation_loss)
45
46     if ((epoch) % 1 == 0):
47         print(f'Epoch [{epoch}/{num_epochs}], Train Loss: {train_loss.item()}, Validation
              Loss: {validation_loss}')

```

Entradas:

Datos de entrenamiento y validación, divididos en lotes proporcionados por `train_dataloader` y `validation_dataloader`.

Salidas:

Pérdidas de entrenamiento y validación en cada época, que se almacenan en las listas `train_losses` y `validation_losses`, y se imprimen al final de cada época.

Propósito:

- **Fase de entrenamiento:** Ajustar los pesos del modelo utilizando los datos de entrenamiento, calculando los gradientes y actualizando los parámetros con *backpropagation*.
- **Fase de evaluación:** Validar el rendimiento del modelo en un conjunto de datos no visto (validación) para verificar que está generalizando y no sobreajustándose.

Apéndice B. Comparación de tonalidades entre el modelo y AIVA

.1. Comparación de tonalidades entre el modelo y AIVA - Guitarra

Resultados

Los resultados se resumen en la Tabla 1, en la que se observa el nombre del archivo, la tonalidad elegida al crear la pista, la tonalidad detectada por AIVA y una evaluación de su equivalencia armónica:

Tabla 1: Comparación completa de tonalidades entre el modelo y AIVA

Archivo	Tonalidad	Detectado por AIVA	¿Equivalente?
Mim_1_MIDI	E Minor	C Minor	✗
Mim_2_MIDI	E Minor	C Minor	✗
Mim_3_MIDI	E Minor	E Minor	✓
Mim_4_MIDI	E Minor	C Minor	✗
Mim_5_MIDI	E Minor	C Minor	✗
Rebm_1_MIDI	Db Minor	G# Minor	✓ (enharmónica)
Rebm_2_MIDI	Db Minor	G# Minor	✓ (enharmónica)
Rebm_3_MIDI	Db Minor	G# Minor	✓ (enharmónica)
Rebm_4_MIDI	Db Minor	G# Minor	✓ (enharmónica)
Rebm_5_MIDI	Db Minor	G# Minor	✓ (enharmónica)
Dom_1_MIDI	C Minor	C Minor	✓
Dom_2_MIDI	C Minor	C Minor	✓
Dom_3_MIDI	C Minor	C Minor	✓
Dom_4_MIDI	C Minor	C Minor	✓
Dom_5_MIDI	C Minor	Ab Major	✓ (modales)

Tabla 2: Comparación completa de tonalidades entre el modelo y AIVA (continuación)

Archivo	Tonalidad	Detectado por AIVA	¿Equivalente?
Fam_1_MIDI	F Minor	Ab Major	✓ (relativa)
Fam_2_MIDI	F Minor	Ab Major	✓ (relativa)
Fam_3_MIDI	F Minor	Ab Major	✓ (relativa)
Fam_4_MIDI	F Minor	G# Minor	✗
Fam_5_MIDI	F Minor	Eb Major	✓ (relativa)
Lab_1_MIDI	Ab Major	Ab Major	✓
Lab_2_MIDI	Ab Major	G# Minor	✗
Lab_3_MIDI	Ab Major	G# Minor	✗
Lab_4_MIDI	Ab Major	G# Minor	✗
Lab_5_MIDI	Ab Major	C Minor	✓ (modales)
Mib_1_MIDI	Eb Major	G# Minor	✗
Mib_2_MIDI	Eb Major	Ab Major	✓
Mib_3_MIDI	Eb Major	Ab Major	✓
Mib_4_MIDI	Eb Major	Ab Major	✓
Mib_5_MIDI	Eb Major	Ab Major	✓
Si_1_MIDI	B Major	G# Minor	✓ (relativa)
Si_2_MIDI	B Major	B Major	✓
Si_3_MIDI	B Major	G# Minor	✓ (relativa)
Si_4_MIDI	B Major	G# Minor	✓ (relativa)
Si_5_MIDI	B Major	G# Minor	✓ (relativa)
Sol_1_MIDI	G Major	E Minor	✓ (relativa)
Sol_2_MIDI	G Major	C Minor	✗
Sol_3_MIDI	G Major	E Minor	✓ (relativa)
Sol_4_MIDI	G Major	C Major	✓ (funcional)
Sol_5_MIDI	G Major	C Minor	✗
Re_1_MIDI	D Major	E Minor	✓ (escalas casi equivalentes)
Re_2_MIDI	D Major	E Minor	✓ (escalas casi equivalentes)
Re_3_MIDI	D Major	E Minor	✓ (escalas casi equivalentes)
Re_4_MIDI	D Major	E Minor	✓ (escalas casi equivalentes)
Re_5_MIDI	D Major	E Minor	✓ (escalas casi equivalentes)
Sim_1_MIDI	B Minor	E Minor	✓ (escalas casi equivalentes)
Sim_2_MIDI	B Minor	E Minor	✓ (escalas casi equivalentes)
Sim_3_MIDI	B Minor	G Major	✓ (relativa)
Sim_4_MIDI	B Minor	E Minor	✓ (escalas casi equivalentes)
Sim_5_MIDI	B Minor	E Minor	✓ (escalas casi equivalentes)

.2. Comparación de tonalidades entre el modelo y AIVA - Bajo

Se consideró como correcta la detección tonal realizada por AIVA en aquellos casos donde la tonalidad detectada y la tonalidad objetivo solo difieren en una nota dentro de sus respectivas escalas. A este tipo de relación se le denomina escalas casi equivalentes, ya que, aunque no son estrictamente iguales ni relativas entre sí, comparten 6 de las 7 notas diatónicas, lo cual implica una alta compatibilidad armónica.

Debido a esta cercanía tonal, se justifica aceptar dichas coincidencias como válidas dentro del análisis comparativo, ya que musicalmente pueden generar resultados muy similares o incluso indistinguibles en ciertos contextos de composición.

Tabla 3: Comparación completa de tonalidades entre el modelo y AIVA - Bajo

Archivo	Tonalidad	Detectado por AIVA	¿Equivalente?
A_1_MIDI	A Major	E Minor	X
A_2_MIDI	A Major	E Minor	X
A_3_MIDI	A Major	E Minor	X
A_4_MIDI	A Major	A Major	✓
A_5_MIDI	A Major	E Minor	X
E_1_MIDI	E Major	E Minor	X
E_2_MIDI	E Major	A Major	✓ (tonalidades cercanas)
E_3_MIDI	E Major	E Minor	X
E_4_MIDI	E Major	E Minor	X
E_5_MIDI	E Major	E Minor	X
D_1_MIDI	D Major	E Minor	✓ (escalas casi equivalentes)
D_2_MIDI	D Major	E Minor	✓ (escalas casi equivalentes)
D_3_MIDI	D Major	E Minor	✓ (escalas casi equivalentes)
D_4_MIDI	D Major	E Minor	✓ (escalas casi equivalentes)
D_5_MIDI	D Major	E Minor	✓ (escalas casi equivalentes)

Tabla 4: Comparación completa de tonalidades entre el modelo y AIVA - Bajo (continuación)

Archivo	Tonalidad	Detectado por AIVA	¿Equivalente?
Bb_1_MIDI	Bb Major	Eb Major	✓ (escalas casi equivalentes)
Bb_2_MIDI	Bb Major	D# Minor	✗
Bb_3_MIDI	Bb Major	D# Minor	✗
Bb_4_MIDI	Bb Major	D# Minor	✗
Bb_5_MIDI	Bb Major	D# Minor	✗
G_1_MIDI	G Major	E Minor	✓ (relativa)
G_2_MIDI	G Major	E Minor	✓ (relativa)
G_3_MIDI	G Major	E Minor	✓ (relativa)
G_4_MIDI	G Major	E Minor	✓ (relativa)
G_5_MIDI	G Major	E Minor	✓ (relativa)
Cm_1_MIDI	C Minor	D# Minor	✗
Cm_2_MIDI	C Minor	D# Minor	✗
Cm_3_MIDI	C Minor	Eb Major	✓ (relativa)
Cm_4_MIDI	C Minor	D# Minor	✗
Cm_5_MIDI	C Minor	D# Minor	✗
Fm_1_MIDI	F Minor	D# Minor	✗
Fm_2_MIDI	F Minor	D# Minor	✗
Fm_3_MIDI	F Minor	D# Minor	✗
Fm_4_MIDI	F Minor	D# Minor	✗
Fm_5_MIDI	F Minor	D# Minor	✗
Am_1_MIDI	A Minor	E Minor	✓ (escalas casi equivalentes)
Am_2_MIDI	A Minor	E Minor	✓ (escalas casi equivalentes)
Am_3_MIDI	A Minor	E Minor	✓ (escalas casi equivalentes)
Am_4_MIDI	A Minor	E Minor	✓ (escalas casi equivalentes)
Am_5_MIDI	A Minor	E Minor	✓ (escalas casi equivalentes)
Em_1_MIDI	E Minor	E Minor	✓
Em_2_MIDI	E Minor	E Minor	✓
Em_3_MIDI	E Minor	E Minor	✓
Em_4_MIDI	E Minor	E Minor	✓
Em_5_MIDI	E Minor	E Minor	✓
Bm_1_MIDI	B Minor	E Minor	✓ (escalas casi equivalentes)
Bm_2_MIDI	B Minor	E Minor	✓ (escalas casi equivalentes)
Bm_3_MIDI	B Minor	E Minor	✓ (escalas casi equivalentes)
Bm_4_MIDI	B Minor	E Minor	✓ (escalas casi equivalentes)
Bm_5_MIDI	B Minor	E Minor	✓ (escalas casi equivalentes)