



UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR



**TFM del Máster en Ingeniería
Informática**

**Aplicación Web para el
reconocimiento de Melodías
mediante el uso de
Inteligencia Artificial**



Presentado por Eduardo Mora González
en la Universidad de Burgos
a 8 de junio de 2021
Tutor: Álvar Arnaiz González



UNIVERSIDAD DE BURGOS

ESCUELA POLITÉCNICA SUPERIOR



D. Álvar Arnaiz González, profesor del departamento de Ingeniería Informática, área de Lenguajes y Sistemas Informáticos.

Expone:

Que el alumno D. Eduardo Mora González, con DNI 05982411-L, ha realizado el Trabajo final de Máster en Ingeniería Informática titulado título de «Reconocimiento y clasificación de melodías usando Inteligencia Artificial».

Y que dicho trabajo ha sido realizado por el alumno bajo la dirección del que suscribe, en virtud de lo cual se autoriza su presentación y defensa.

En Burgos, 8 de junio de 2021

Vº. Bº. del Tutor:

D. Álvar Arnaiz González

*Interfaz del capricho con la indecisión,
Mensaje de error.
Sistema en off. Error, Sistema en off.*

FANGORIA

Resumen

Está claro que la aparición y desarrollo de las Tecnologías de la Información y Comunicación en los últimos tiempos ha revolucionado la manera en la que percibimos e interactuamos con el mundo. La música tampoco se ha quedado atrás en el uso de estas nuevas tecnologías y actualmente se ha lanzado diversas líneas de investigación para analizar y comprender este arte. El presente documento supone un estudio en una de estas investigaciones.

En el artículo «A Convolutional Approach To Melody Line Identification In Symbolic Scores» se propone una Red Neuronal que, tras ser entrenada, permite separar la línea melódica del acompañamiento.

Una vez estudiada y probada dicha Red Neuronal y mediante el uso de Python y el framework Flask se ha desarrollado una Aplicación Web para poder utilizarla.

Finalmente, se ha creado una estructura de contenedores Dockers para separar cada una de las partes: Red Neuronal, Aplicación Web y BBDD. Estos contenedores se han comunicado mediante Sockets y Post de Python.

Descriptores

Inteligencia Artificial, Red Neuronal, Música, Flask, Melodía y Acompañamiento.

Abstract

The emergence and development of Information and Communication Technologies in recent times has revolutionized our perception and interaction with the world. Music has not kept out of new technologies and their uses. Several research areas have been launched to analyze and understand this art. This research is one more contribution to this field of knowledge.

«A convolutional approach to the identification of melodic lines in symbolic scores » propounds a Neural Network capable of separating the melodic line from the accompaniment.

When the Neural Network has been trained and tested through Python and the Flask framework, a Web Application has been created to exploit it.

Finally, a structure of Dockers containers has been created to separate each of the parts: Neural Network, Web Application and DB. These containers have been communicated using Python Sockets and Post.

Keywords

Artificial Intelligence, Neural Network, Music, Flask, Melody and Accompaniment.

Agradecimientos

Quisiera agradecer a todas las personas y entidades la ayuda que me han prestado en la realización de este Trabajo Fin de Máster.

En primer lugar, a mis padres, hermano y abuelos, que siempre han estado a mi lado apoyándome para seguir hacia adelante.

Quisiera hacer extensiva mi gratitud a *Álvar Arnaiz González* tutor de este T.F.M., por todo lo que me ha aportado y ayudado a juntar mis dos grandes pasiones: la música y la informática.

Finalmente, y no por ello menos importante, quiero agradecer a *Lucia Rosa Alonso* todos los consejos y ayuda que me ha prestado desde que nos conocemos, sin ella todo esto no hubiera sido lo mismo.

A todos ellos, muchas gracias.

Eduardo Mora González

Índice general

Índice general	VIII
Índice de figuras	XI
Índice de tablas	XIII
1 Introducción	1
1.1. Contexto	1
1.2. Motivación	2
1.3. Estructura del documento	2
2 Objetivos del proyecto	4
2.1. Objetivo General	4
2.2. Objetivos Parciales	4
3 Conceptos Teóricos	6
3.1. Fundamentos Musicales	6
3.2. Fundamentos de la Inteligencia Artificial	10
3.3. A Convolutional Approach To Melody Line Identification In Symbolic Scores	12
4 Técnicas y herramientas	20
4.1. Metodologías Ágiles	20
4.2. Docker	23
4.3. Flask	24
4.4. Otras Herramientas	24
5 Aspectos relevantes del desarrollo del proyecto	26

5.1.	Funcionamiento de la Red Neuronal	26
5.2.	Comunicación entre Dockers	26
5.3.	Desarrollo de una Aplicación Flask	27
5.4.	Comunicación entre la Aplicación Web y la Red Neuronal	27
6	Trabajos relacionados	29
6.1.	DeepBach	29
6.2.	MuseNet	30
6.3.	SaxEx	31
7	Conclusiones y Líneas de trabajo futuras	33
7.1.	Consecución de objetivos	33
7.2.	Competencias	34
7.3.	Trabajo futuro	35
Apéndice A	Plan de Proyecto Software	36
A.1.	Planificación temporal	36
A.2.	Viabilidad Legal	44
Apéndice B	Especificación de Requisitos	46
B.1.	Introducción	46
B.2.	Objetivo general	46
B.3.	Especificación de requisitos	47
Apéndice C	Especificación de diseño	49
C.1.	Diagrama de Casos de Uso	49
C.2.	Diseño Base de Datos	52
C.3.	Diagrama de Secuencia	54
C.4.	Diagrama de Despliegue	55
C.5.	Diseño de interfaz	55
Apéndice D	Documentación técnica de programación	59
D.1.	Introducción	59
D.2.	Estructura del Sistema	59
D.3.	Manual del programador	61
D.4.	Compilación, instalación y ejecución del proyecto	64
D.5.	Pruebas del sistema	65
Apéndice E	Documentación de usuario	69
E.1.	Introducción	69
E.2.	Requisitos de usuarios	69
E.3.	Manual del usuario	69

Índice de figuras

3.1. Ejemplo de Monofonía	8
3.2. Ejemplo de melodía acompañada	8
3.3. Ejemplo de Homofonía	9
3.4. Ejemplo de Polifonía	9
3.5. Esquema global del algoritmo del articulo.	13
3.6. Esquema de la Red Neuronal del algoritmo.	14
3.7. Extracto de un ejemplo del algoritmo.	15
3.8. Algoritmo para construir un Grafo Acíclico Dirigido.	16
3.9. Mapa de prominencia.	17
4.1. Eventos de <i>Scrum</i>	22
6.1. Esquema del algoritmo <i>DeepBach</i>	30
6.2. Diagrama de bloques de los componentes de <i>SaxEx</i>	32
C.1. Diagrama general de Casos de Uso	49
C.2. Diagrama de diseño	53
C.3. Diagrama Relacional	53
C.4. Diagrama de Secuencia	54
C.5. Diagrama de Despliegue	55
C.6. Prototipo de Login en el sistema	56
C.7. Prototipo de Registro en el sistema	56
C.8. Carga del fichero	57
C.9. Proceso de conversión	57
C.10. Descarga de la salida	58
C.11. Biblioteca del sistema	58
D.1. Registrar un Usuario existente	65
D.2. Salida registrar un Usuario existente	65

D.3. Registrar un Usuario con correo erróneo	66
D.4. Salida registrar un Usuario con correo erróneo	66
D.5. Identificarse con usuario incorrecto	67
D.6. Salida identificarse con usuario incorrecto	67
D.7. Identificarse con usuario correcto	68
D.8. Inicio del Sistema	68
D.9. Mensaje de No se puede procesar	68
E.1. Pantalla Login del sistema	70
E.2. Pantalla Registro del sistema	70
E.3. Inicio del Sistema	71
E.4. Mensaje de No se puede procesar	71
E.5. Subir Canción para Procesar	72
E.6. Biblioteca canción sin procesar	72
E.7. Biblioteca canción procesada	73

Índice de tablas

C.1. Tabla Caso de Uso Registrar	50
C.2. Tabla Caso de Uso Identificar	50
C.3. Tabla Caso de Uso Procesar Canción	51
C.4. Tabla Caso de Uso Subir Canción	51
C.5. Tabla Caso de Uso Separar Melodía/Acompañamiento	52
C.6. Tabla Caso de Uso Descargar Melodía/Acompañamiento	52

Capítulo 1

Introducción

En este capítulo inicial se pretende contextualizar todas aquellas innovaciones que este Trabajo Fin de Máster pretende resolver, señalando las motivaciones que nos han llevado a su realización y presentando la estructura del documento.

1.1. Contexto

La búsqueda y elección de un tema para realizar un Trabajo Fin de Máster, puede resultar una tarea bastante tediosa, pues serían tantas las opciones disponibles que costaría tomar una decisión.

Sabiendo que el abanico de posibilidades es bastante amplio y que la elección de alguno de esos temas puede servir para innovar o complementar conocimientos, no sólo a la hora de comprenderlos y analizarlos, sino también para usarlo en un espacio más práctico.

Hablar de innovación es decir que se da comienzo, desde sus primeros pasos, a cualquier nueva idea que aparezca en un momento determinado y que al día siguiente de su aparición ya se está mejorando. En un proceso en el que este nuevo proyecto después de alcanzar su punto más álgido vislumbrará su fin, por haber sido superada por otra idea mejor y más innovadora, así sucesivamente, y que, aunque se desconoce cuándo y cómo, esto sucederá [1].

La música acompaña al hombre desde su existencia. En cualquier estudio antropológico de la humanidad esta siempre ha estado presente y asociada directamente a su evolución [2].

Para muchos un arte, para otros una dimensión intangible; en todo caso, al igual que se puede leer y sentir una poesía, se puede leer y sentir una partitura sin necesidad de escucharla.

Actualmente el uso de las T.I.C. está presente en todos y cada uno de los diferentes ámbitos de la vida del ser humano. Y como se puede comprender, la música tampoco se ha quedado atrás en este sentido.

1.2. Motivación

No es la primera vez que intento mezclar dos de mis pasiones: La Música y La Informática. El primer intento fue en mi Trabajo de Fin de Grado titulado: «*Propuesta tecnológica para apoyar el aprendizaje de un instrumento monódico mediante comparación de pasajes musicales*» [1], donde descubrí que la informática se puede usar en todos los campos que uno se plantee.

Por esto, en este Trabajo de Fin de Máster, y con la ayuda de mi tutor D. Álvar Arnaiz González, nos hemos embarcado en otra aventura donde queríamos plasmar todos los conocimientos adquiridos durante la realización de este Máster.

En este T.F.M. hemos analizado, probado, modificado y usado una Red Neuronal Convolutacional descrita en el artículo «*A Convolutional Approach To Melody Line Identification In Symbolic Scores*» [3] y hemos creado una aplicación web donde se pueda usar dicha Red.

1.3. Estructura del documento

A continuación, se detalla la estructura del documento mencionando y explicando de una manera breve y concisa todos los capítulos y anexos que lo componen.

Capítulo 2: Objetivos del proyecto

En este segundo capítulo se muestran el objetivo principal del proyecto y los objetivos parciales de este.

Capítulo 3: Conceptos Teóricos

En este capítulo, se hace un despliegue de los conceptos necesarios para poder comprender la totalidad del proyecto.

Además, se realiza un análisis en profundidad del artículo en el que se basa todo el proyecto.

Capítulo 4: Técnicas y herramientas

En este capítulo, se presenta la metodología de desarrollo seguida y se da una visión de las distintas herramientas utilizadas para llevar a cabo el desarrollo e implementación del sistema.

Capítulo 5: Aspectos relevantes del desarrollo del proyecto

En este quinto capítulo se recogen todos los aspectos más interesantes que han ocurrido durante el desarrollo del proyecto.

Capítulo 6: Trabajos relacionados

En este capítulo, se hace un análisis de los sistemas existentes que relacionan la Inteligencia Artificial y la Música. Se muestran los que se han considerado más relevantes.

Capítulo 7: Conclusiones y Líneas de trabajo futuras

En este último capítulo y a modo de conclusión del proyecto, se analiza uno a uno la consecución de los objetivos marcados. Además, se analizan las competencias adquiridas durante el desarrollo de este Trabajo Fin de Máster y se plantean algunas líneas de trabajo futuras.

Anexo A: Plan de Proyecto Software

En este primer anexo se muestra todo lo relativo a la planificación del Trabajo Fin de Máster, así como la viabilidad legal que va a tener dicho trabajo.

Anexo B: Especificación de Requisitos

En este segundo anexo, se enumeran los objetivos que se pretenden alcanzar en este Trabajo Fin de Máster, así como todos los requisitos funcionales y no funcionales de este.

Anexo C: Especificación de Diseño

En este anexo, se ilustra todo lo relativo al diseño del sistema. En cada una de sus partes, se detallan de una manera precisa y a través de bocetos y diagramas todo el diseño y la estructura del sistema.

Anexo D: Documentación técnica de programación

En este cuarto anexo se pautaliza como está montado el sistema desde un punto más técnico.

Anexo E: Documentación de usuario

Finalmente, este anexo explica de una forma clara y concisa, los pasos necesarios para que cualquier usuario pueda usar el sistema.

Capítulo 2

Objetivos del proyecto

En este segundo capítulo se muestran el objetivo principal del proyecto y los objetivos parciales de este.

2.1. Objetivo General

El objetivo principal de este Trabajo Fin de Máster es estudiar la implementación y funcionamiento de la Red Neuronal propuesta en el artículo «A Convolutional Approach To Melody Line Identification In Symbolic Scores» [3] y ser capaz de usar dicha Red Neuronal a través de una Aplicación Web.

2.2. Objetivos Parciales

Una vez detallado el objetivo principal, vamos a enumerar los objetivos parciales::

- Entender, analizar y ser capaz de usar la Red Neuronal propuesta en el artículo «A Convolutional Approach To Melody Line Identification In Symbolic Scores» [3].
- Crear una Aplicación Web conectada a una Base de Datos en donde se envíen peticiones a la Red Neuronal y se puedan almacenar la entrada y salida de dicha Red.

- Modularizar cada una de las partes del sistema (Red Neuronal, Aplicación Web y Base de Datos) en contenedores *Docker* y establecer una comunicación entre ambos contenedores.

Capítulo 3

Conceptos Teóricos

En este capítulo se introducen todos aquellos fundamentos básicos tanto musicales como conceptos de la Inteligencia Artificial, necesarios para comprender y abordar la solución al problema planteado.

3.1. Fundamentos Musicales

En este primer apartado se mostrarán los fundamentos teóricos musicales necesarios para comprender este proyecto.

El sonido

El fundamento natural de la música es el sonido. **El sonido** se produce cuando una fuente sonora vibra. Dicha vibración es un movimiento de vaivén que obligará a moverse de la misma manera a las moléculas de aire que están próximas a la fuente. Esa oscilación, juntará o separará las moléculas, creando zonas de compresión y de depresión que a su vez empujarán a otras moléculas contiguas produciéndose ondas sonoras. Las cualidades del sonido son las siguientes [4]:

- **Duración:** Se puede decir que la duración del sonido depende directamente de la duración del movimiento vibratorio que provoca el sonido.
- **Volumen:** El volumen es una sensación que aumenta a medida que aumenta la amplitud y disminuye a medida que aumenta la frecuencia.

- **Densidad:** Definimos densidad como la sensación que producen ciertos sonidos de ser más compactos, más llenos que otros.
- **Altura:** La altura de un sonido está determinada por la frecuencia de las vibraciones de un cuerpo sonoro, es decir, su carácter de Grave o Agudo. La altura depende del número de vibraciones por segundo que posee un sonido.
- **Timbre:** El Timbre define la diferencia en el color tonal de una nota tocada por diversos instrumentos o cantada por diferentes voces. El timbre depende de la forma de la onda sonora. El Sonido Puro (la vibración sinusoidal) sólo se puede conseguir electrónicamente o mediante el Diapasón.
- **Intensidad:** Es la cualidad del sonido que permite diferenciar entre un sonido fuerte de otro débil y viceversa.

Modos básicos de organización temporal

Para comprender los distintos conceptos relacionados con la organización temporal, se hará uso de poéticas palabras de *Mathis Lussy*:

“En el tiempo, el hombre...clava sus jalones a fin de interrumpir la continuidad, la uniformidad, establecer puntos de apoyo para satisfacer sus necesidades rítmicas. Así nace la música y la poesía. Estos puntos de apoyo, estos límites son intangibles; son instantes fisiológicos y es su organismo, es su respiración la que los suministra. El ritmo es para la música lo que la simetría es para la arquitectura; es la división, la ruptura regular en la continuidad del tiempo, como la simetría es la división regular del espacio.” [5]

Entonces podemos definir los distintos conceptos de organización temporal de la siguiente manera [6]:

- **Pulso:** estímulo idéntico e isócrono que se repite en una serie regular. Marca unidades iguales en el *continuum* temporal.
- **Compás:** es la medida del número de pulsos existentes entre los acentos que van apareciendo con una recurrencia más o menos regular.
- **Ritmo:** modo en que una o más partes no acentuadas son agrupadas en relación con otra parte que si lo está.

Textura

La textura, en general, es el modo en que se combinan la melodía, el ritmo y la armonía en una composición o fragmento musical [7]. Depende la combinación que se haga podemos obtener diferentes tipos de texturas:

- **Monodía o monofonía:** Consiste en una sola línea melódica, sin armonía ni acompañamiento. En la figura 3.1 se muestra un fragmento del *Preludio de la Suite 1 para cello J.S.Bach* en donde se puede ver un ejemplo bastante conocido de este tipo textura.

Suite I
BWV 1007

1. Prélude



Figura 3.1: Ejemplo de Monofonía

- **Melodía acompañada:** Destaca una melodía sobre un acompañamiento, dicha melodía puede aparecer a distintas alturas. En la figura 3.2 se muestra un fragmento de la *Elégie para cello de G.Fauré* donde se puede ver como el cello tiene la melodía y el piano el acompañamiento.

Élégie

Gabriel Fauré, Op. 24

Molto adagio.
 $\text{♩} = 80$

Violoncello

Piano



Figura 3.2: Ejemplo de melodía acompañada

- **Homofonía:** La melodía está formada por un conjunto de voces o partes melódicas diferentes que suelen tener el mismo ritmo. En la figura 3.3 se muestra un fragmento de *La Pasión según San Mateo de J.S.Bach*, en donde todas las voces en su conjunto forman una melodía.

15. Erkenne mich, mein Hüter

The musical score consists of two staves. The top staff is in common time (C) and the bottom staff is also in common time (C). The key signature changes from C major to A minor (Am) at the end of the measure. Below the staves, harmonic analysis is provided:

Do M: I IV I⁶ IV⁶ V⁵ I ii⁶ V I

la m: Am: vii⁶ i i⁶ V⁷ i

Figura 3.3: Ejemplo de Homofonía

- **Polifonía:** Un conjunto de voces suena de modo simultáneo pero las voces son, en mayor o menor grado, independientes entre sí. En la figura 3.4 se muestra un fragmento del *Motete Sicut cervus de Palestrina*, en donde cada voz tiene su propia melodía y forman un contrapunto entre si.

The musical score is labeled "Prima pars". It features four staves, each representing a different voice. The voices are independent, creating a polyphonic texture. The lyrics are written below the notes:

Prima pars

Sicut cervus de-
Sicut cer- vus de- si - de - rat ad fon-
Sicut cer- vus de - si - de - rat ad fon - tes a - qua rum,
Si-

Figura 3.4: Ejemplo de Polifonía

3.2. Fundamentos de la Inteligencia Artificial

En este apartado se mostrarán los fundamentos de la Inteligencia Artificial necesarios para comprender este proyecto.

Definiciones y aplicaciones de la Inteligencia Artificial

La R.A.E. define la Inteligencia Artificial (I.A.) como la disciplina científica que se ocupa de crear programas informáticos que ejecutan operaciones comparables a las que realiza la mente humana, como el aprendizaje o el razonamiento lógico [8].

El propósito general de la I.A. [9] es desarrollar:

- Modelos conceptuales.
- Procedimientos de reescritura formal de esos modelos.
- Estrategias de programación y máquinas físicas para reproducir de la forma más eficiente y completa posible las tareas cognitivas y científico-técnicas más genuinas de los sistemas biológicos a los que hemos etiquetado de inteligentes.

Paradigmas actuales de la Inteligencia Artificial

Aunque algunos sólo consideran dos (los basados en representaciones o simbólico y los basados en mecanismos o subsimbólico), es usual, distinguir cuatro paradigmas básicos [9]:

- **Simbólico o representacional:** La I.A. Simbólica estudia la programación computacional del razonamiento simbólico enfrentándose, en primer lugar al problema de cómo representar la realidad y en segundo lugar, a cómo manipular los símbolos separadamente de sus representaciones y sus significados. Colateralmente, encontramos que el lenguaje natural, es básicamente esto mismo: un conjunto de símbolos convencionales dotados de representación y significado [10].
- **Situado:** También llamado paradigma reactivo o «basado en conductas», este paradigma, es usado, sobre todo en robótica y aplicaciones en tiempo real. Este paradigma, enfatiza el hecho de que toda percepción

y toda acción están estructuralmente acopladas, a través de sensores y efectores concretos, a un medio externo e interno también concretos [9].

- **Conexionista:** El conexionismo, es una teoría sobre la forma de entender la estructura y funcionamiento de los sistemas cognitivos. En cierto modo, el conexionismo, supuso un desafío para la teoría dominante en las Ciencias Cognitivas y la Inteligencia Artificial en los años ochenta y ofreció nuevas herramientas para entender la forma en la que los procesos cognitivos («como la percepción», «atención», «memoria», «lenguaje», ...) pueden estar implementados o pueden emergir de sistemas neuronales [11].
- **Híbrido [9]:** Este paradigma se divide en varias arquitecturas:
 - **Integración entre los paradigmas Simbólicos y Conexionista:** las redes neuronales actúan como preprocesadores, post-procesadores o coprocesadores subordinados a un procesador simbólico central.
 - **Sistemas Neuroborrosos:** estos sistemas engloban formas de modelar conocimientos y operacionalizar inferencias en las que se mezclan componentes neuronales y borrosas.
 - **Arquitectura Multiagente:** se integran sobre una plataforma con distintos tipos de módulos encapsulados (simbólicos, borrosos, situados o neuronales) que interactúan, a través de una interfaz de comunicación.

Redes Neuronales Convolucionales

Las Redes Neuronales Convolucionales (*Convolutional Neural Network* o CNN), es un tipo de Red Neuronal Artificial con aprendizaje supervisado que procesa sus capas, imitando al cortex visual del ojo humano, para identificar distintas características en las entradas que en definitiva hacen que pueda identificar objetos [12].

Las Redes Neuronales Convolucionales, consisten en múltiples capas de filtros convolucionales de una o más dimensiones. Después de cada capa, por lo general, se añade una función para realizar un mapeo causal no-lineal.

Como redes de clasificación, al principio se encuentra la fase de extracción de características, compuesta de neuronas convolucionales y de reducción de muestreo. Al final de la red, se encuentran neuronas de perceptrón sencillas, para realizar la clasificación final sobre las características extraídas. La fase

de extracción de características se asemeja al proceso estimulante en las células de la corteza visual.

Esta fase, se compone de capas alternas de neuronas convolucionales y neuronas de reducción de muestreo. Según progresan los datos, a lo largo de esta fase, se disminuye su dimensionalidad, siendo las neuronas en capas lejanas mucho menos sensibles a perturbaciones en los datos de entrada, pero al mismo tiempo, siendo estas activadas por características cada vez más complejas [13].

3.3. A Convolutional Approach To Melody Line Identification In Symbolic Scores

En este artículo [3], se propone un enfoque convolucional destinado a identificar la línea melódica de una pieza, utilizando una representación de pianola de la partitura y haciendo uso de una de las formas de definir las «voces» de *Cambouropoulos*:

- Para la música con múltiples instrumentos, se puede decir que cada instrumento constituye una voz separada, esto permitiría la posibilidad de voces no monofónicas en instrumentos que producen acordes.
- Las voces pueden asignarse a corrientes melódicas, a medida que son percibidas y segmentadas por los oyentes, siguiendo principios de agrupación cognitiva.
- En la música monofónica, el contenido armónico de la pieza puede implicar una organización horizontal de voces polifónicas que se despliegan con el tiempo (es decir, polifonía implícita).

En este artículo, se utiliza la segunda definición y se concreta la línea melódica como la voz más destacada. Para poder llevarlo a cabo se ha recurrido a la literatura sobre recuperación de información musical. Se han abordado tres tareas:

- Separación de voces de partituras simbólicas.
- Identificación de la pista principal (de archivos MIDI con múltiples pistas).
- Identificación de la melodía principal del audio.

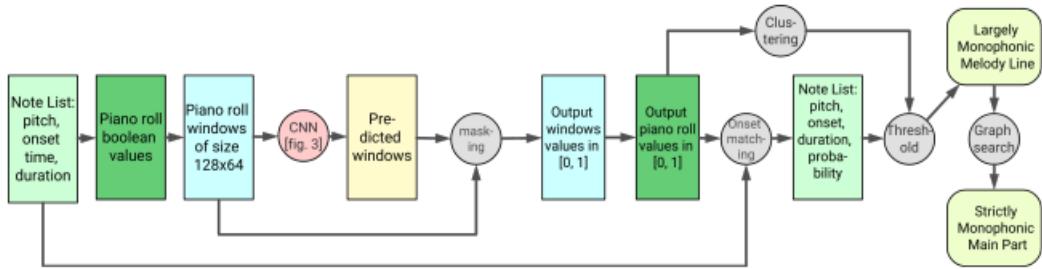


Figura 3.5: Esquema global del algoritmo del articulo. Figura extraída de [3]

Este último es un problema diferente al que se aborda aquí ya que se trata de la tarea de identificar notas de un archivo de audio.

Para poner un punto de partida, el articulo coge dos algoritmos como base a seguir:

- **Algoritmo de Skyline:** este algoritmo, es una heurística que toma la nota más alta en cada momento [14].
- **VoSA:** es un método de separación de voz exitoso. En este enfoque, una pieza se divide en segmentos en función de los puntos de entrada y salida de la voz, de modo que el número de notas que suenan es constante dentro de cada segmento [15].

El resultado final del articulo es un algoritmo que como se puede observar en la figura 3.5

Red Neuronal Convolucional

La columna vertebral del algoritmo consiste en una Red Neuronal completamente convolucional, que toma como entrada segmentos de una partitura, representada como un «piano roll», y estima probabilidad de que cada nota de la partitura pertenezca a la línea melódica. En la figura 3.6 se puede observar el esquema de la Red Neuronal.

El algoritmo usa «piano roll», que es una representación en 2D, de una partitura; donde el eje X indica el tiempo de puntuación y el eje Y indica el tono. El algoritmo recoge cada vez 8 píxeles/tiempo del «piano roll» y cada ventana, tiene una longitud fija de 64 píxeles (es decir, 8 tiempos).

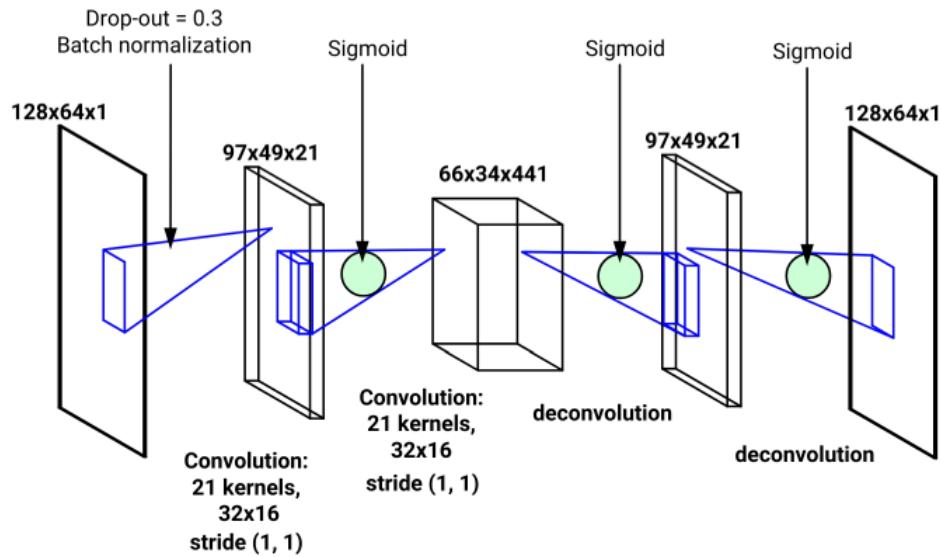


Figura 3.6: Esquema de la Red Neuronal del algoritmo. Figura extraída de [3]

La superposición entre ventanas es del 50 % (es decir, 2 tiempos) y las ventanas más cortas que este tamaño se rellenan con ceros. Se construye un «piano roll» de salida para cada pieza completa promediando, las probabilidades de los píxeles ubicados en ventanas superpuestas.

Posteriormente, aplicamos una máscara en el «piano roll» de salida, multiplicándolo por el «piano roll» de entrada (binario), de modo que las áreas sin notas tomen valores de cero y las probabilidades distintas de cero solo permanezcan donde hay notas. La probabilidad de que cada nota pertenezca a la melodía, se calcula como la mediana de los valores de salida de sus píxeles.

En la figura 3.7 se muestra un extracto de la *Sonata para piano K.545 de Mozart* y tres representaciones de «piano roll» alineados verticalmente correspondientes al extracto:

- La primera fila, representa el fragmento de la partitura original que se quiere analizar.
- La segunda fila, representa el «piano roll» de la partitura de original y es la entrada que recibe la Red Neuronal.

- La tercera fila, representa el «piano roll» de la melodía identificada, esta se obtiene después de aplicar la Red Neuronal.
- La ultima fila, representa la salida del sistema, donde se divide la linea de la melodía y el acompañamiento.

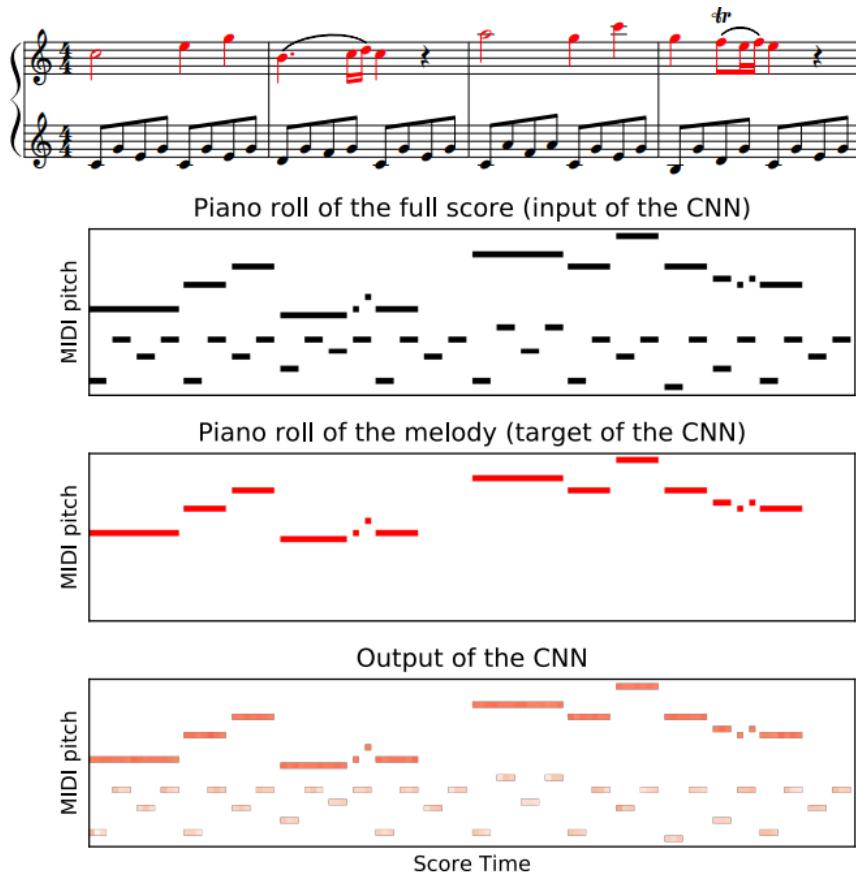


Figura 3.7: Extracto de un ejemplo del algoritmo. Figura extraída de [3]

Grafo Acíclico Dirigido

Una vez identificadas las notas que superan el umbral definido anteriormente (tercera fila de la figura 3.7), tenemos que seleccionar una secuencia de estas notas que maximice la probabilidad de que la secuencia sea monofónica. Esto se logra mediante un enfoque basado en grafos.

El algoritmo que se muestra en la figura 3.8 se usa para construir un grafo acíclico dirigido, dicho grafo consta de un conjunto de nodos cuyo

valor, es el umbral por el que pasa cada nota y un conjunto de arcos dirigidos que contiene la información de tono, inicio y duración. Cada uno de estos arcos especifica una conexión de un nodo a otro.

```

 $L \leftarrow$  list of notes
 $\alpha \leftarrow$  starting node (end time = 0)
 $\omega \leftarrow$  ending node (onset =  $\infty$ , probability = -0.5)
Push  $\alpha$  to the beginning of  $L$ 
Push  $\omega$  to the end of  $L$ 
for note in  $L$  do
     $L' \leftarrow$  notes with onset  $\geq$  end time of note
     $L' \leftarrow$  notes with onset = minimum onset in  $L'$ 
    for note' in  $L'$  do
        if probability of note'  $\geq$  threshold then
             $p =$  probability of note'
            add an edge (note, note') with weight  $-p$ 
        end if
    end for
end for

```

Figura 3.8: Algoritmo para construir un Grafo Acíclico Dirigido. Figura extraída de [3]

Las probabilidades de notas, se utilizan para determinar la fuerza de la conexión entre nodos (similar a una «distancia»; las notas con probabilidades altas se consideran «más cercanas»). Además, se establece un nodo inicial y final al principio y al final de la pieza, respectivamente.

Entrenamiento del Sistema

La Red Neuronal está entrenada de manera supervisada para filtrar las partes de acompañamiento. Las entradas se proporcionan en forma de segmentos de «piano roll» y los objetivos son los «piano roll» correspondientes con solo las notas de melodía.

Las redes se entrenaron usando *AdaDelta* con una tasa de aprendizaje inicial establecida en 1.

Para evitar el sobreajuste, se usó *drop-out* con probabilidad $dropout=0.3$ y regularización de peso de la norma L1.

Para entrenar y evaluar se utilizaron varios conjuntos de datos diferentes para evaluar el desempeño del método:

- El primer conjunto de datos perteneciente a obras de *Mozart* consta de 38 movimientos de (13) Sonatas para piano, para los cuales la línea de la melodía principal fue anotada manualmente por un pianista profesional.
- El segundo conjunto de datos consta de 83 canciones populares (incluidos pop y jazz).
- Un tercer conjunto de datos, que se usa solo para pruebas, comprende archivos MIDI rastreados desde la Web. Este conjunto de datos incluye 169 composiciones musicales de arte occidental de finales del siglo XVI a principios del XX.

Todas estas piezas, incluían un instrumento solista (normalmente voz, flauta, violín o clarinete) y acompañamiento (normalmente cuerdas o piano).

Para investigar lo que está aprendiendo la Red Neuronal, propusieron un método (similar a un análisis de sensibilidad) que evalúa la contribución de las ubicaciones individuales del «piano roll» a las predicciones en otras ubicaciones utilizando mapas de prominencia.

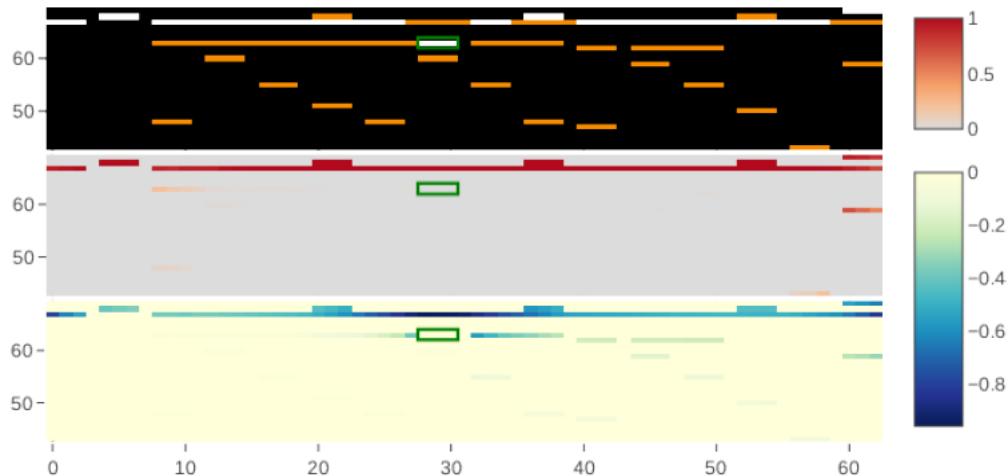


Figura 3.9: Mapa de prominencia. Figura extraída de [3]

El método implica probar cómo cambia la probabilidad de que una nota dada pertenezca a la melodía (es decir, aumenta o disminuye) cuando se eliminan otras notas (es decir, al convertir los píxeles que pertenecen a esas notas a 0).

En la figura 3.9, las notas de tono alto, con un color azulado, que se producen alrededor de los tiempos 20 y 35, tienen valores de prominencia no positivos. Debido a que tienen un tono más alto, estas notas contribuyen negativamente a la nota de melodía resaltada con un recuadro verde, por lo que es poco probable que esta nota se identifique como melodía.

Evaluación del Sistema

Para evaluar la calidad de las predicciones del método propuesto, se realizaron dos experimentos:

- En el primer experimento, se buscaba evaluar la precisión predictiva de los modelos entrenados en diferentes conjuntos de datos.
- En el segundo experimento, se evaluaban cómo de bien estaban generalizados los modelos en diferentes estilos musicales.

Para el primer experimento, se realizó una validación cruzada de 10 archivos en cada uno de los conjuntos de datos de *Mozart* y Pop. El modelo se entrenó con 9 de estos archivos y se probó con el restante.

Se realizó para todas las combinaciones posibles, para que cada pieza de cada conjunto de datos apareciera en el conjunto de prueba una vez.

El segundo experimento se probó cómo de bien entrenados estaban los modelos y como eran capaz de generalizar a nuevos tipos de datos. Presupusieron que los modelos entrenados en el conjunto de datos de *Mozart* superarían a los modelos entrenados en el conjunto de datos Pop, ya que los conjuntos de datos de *Mozart* y Web son más similares en estilo (aunque el conjunto de datos Web es más heterogéneo). Sin embargo, no se encontraron diferencias significativas entre los modelos; ambos modelos funcionaron bien en el conjunto de datos.

Conclusión del Artículo

La conclusión del artículo fue que se implementó y analizó un método novedoso para identificar la línea melódica en una partitura musical simbólica.

Se encontró que algunas de las funciones del modelo eran similares a las funciones del algoritmo de horizonte y *VoSA* (en particular, enfocarse en el tono más alto y definir una línea de melodía para encontrar la secuencia de notas que minimiza el costo de conexión).

Sin embargo, el método no tiene en cuenta la naturaleza secuencial a largo plazo de la música; puede calcular ventanas en cualquier orden. Si bien, tal propiedad puede tener algunos beneficios prácticos, también hace que la red no pueda generalizar a diversas texturas, lo que conduce a malos resultados cuando la textura musical es variada.

Capítulo 4

Técnicas y herramientas

En este capítulo se pretende presentar la metodología de desarrollo seguida y dar una visión de las distintas herramientas utilizadas para llevar a cabo el desarrollo e implementación del sistema.

4.1. Metodologías Ágiles

Las metodologías ágiles son un tipo de metodología de trabajo que comparten unos principios y valores, que permiten desarrollar *Software* rápidamente y responder a cualquier cambio que pueda surgir a lo largo del proyecto.

En febrero de 2001, en una reunión en *Utah-EEUU* aparece por primera vez el término «Ágil». En dicha reunión se creó *The Agile Alliance* y participaron 17 expertos de la industria *Software* en la creación.

The Agile Alliance es una organización, sin ánimo de lucro, fundamentada en el fin de promover los principios del desarrollo ágil ayudando a las organizaciones para adaptar dichos conceptos [16].

Scrum

Scrum, es un método para trabajar en equipo, a partir de iteraciones o *Sprints*. Así pues, *Scrum* es una Metodología Ágil, por lo que su objetivo será controlar y planificar proyectos con un gran volumen de cambios de última hora, en donde la incertidumbre sea elevada.

Se suele planificar por semanas. Al final de cada *Sprint* o iteración, se va revisando el trabajo validado de la anterior semana. En función de esto, se priorizan y planifican las actividades en las que invertiremos nuestros recursos en el siguiente *Sprint* [17].

Teoría Scrum

Scrum, se basa en la teoría de control de procesos empírica o empirismo. El empirismo asegura que el conocimiento procede de la experiencia y de tomar decisiones basándose en lo que se conoce.

Scrum, emplea un enfoque iterativo e incremental para optimizar la predictibilidad y el control del riesgo. Tres pilares soportan toda la implementación del control de procesos empírico [18]:

- **Transparencia:** Los aspectos significativos del proceso deben ser visibles para aquellos que son responsables del resultado. La transparencia requiere que dichos aspectos sean definidos por un estándar común, de tal modo que los observadores compartan un entendimiento común de lo que se están viendo.
- **Inspección:** Los usuarios de *Scrum* deben inspeccionar frecuentemente los artefactos de *Scrum* y el progreso hacia un objetivo, para detectar variaciones indeseadas. Su inspección, no debe ser tan frecuente como para que interfiera en el trabajo. Las inspecciones son más beneficiosas cuando se realizan de forma diligente por inspectores expertos en el mismo lugar de trabajo
- **Adaptación:** Si un inspector determina que uno o más aspectos de un proceso se desvían de límites aceptables y que el producto resultante será inaceptable, el proceso o el material que está siendo procesado deben ajustarse. Dicho ajuste debe realizarse cuanto antes para minimizar desviaciones mayores. *Scrum* prescribe cuatro eventos formales, contenidos dentro del *Sprint*, para la inspección y adaptación, tal y como se describen en la sección Eventos de *Scrum* del presente documento.

El Sprint y los eventos de Scrum

El Sprint [19] es un contenedor para el resto de los eventos de *Scrum*. El Sprint es continuo, es decir, su duración no cambia y se puede interpretar como una medida de ritmo que no cambia a lo largo del tiempo y nos permite

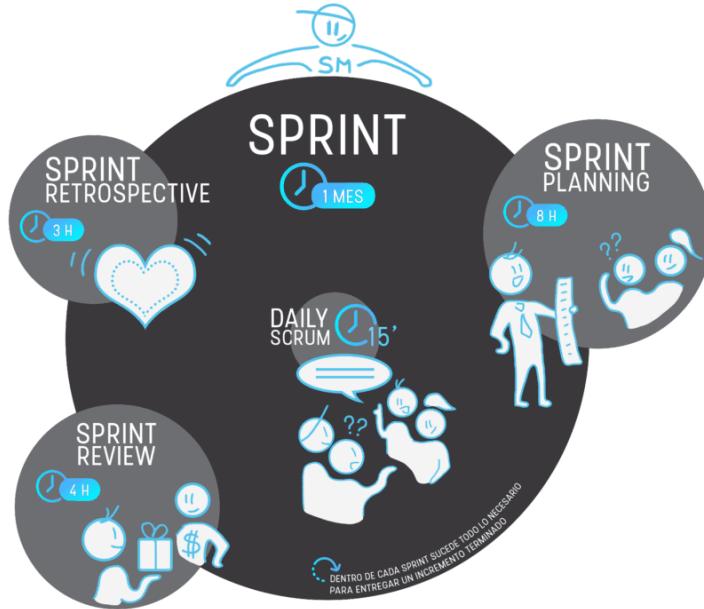


Figura 4.1: Eventos de *Scrum*. Figura extraída de [19].

reducir complejidad y comparar resultados a lo largo de diferentes *Sprints*. El Sprint sirve a la transparencia y permite inspeccionar y adaptar todos los otros eventos de *Scrum*.

Scrum prescribe que un Sprint debe durar 30 días o menos. La duración de un Sprint, está determinada por el periodo mínimo en que un equipo de desarrollo puede generar valor a través de un incremento terminado. El Sprint, es una iteración definida (*time boxed*) que sirve al desarrollo iterativo e incremental. Todo el desarrollo se realiza durante el mismo Sprint y este contiene al resto de los eventos en *Scrum*, teniendo una duración de un mes o menos. La duración del Sprint, se determina por un horizonte de *planning* aceptable. No hay fases en *Scrum*, sólo *Sprints*. No existen los *Sprints de testing, hardening, release* o análisis.

El Sprint, es un evento que contiene a todos los demás eventos en *Scrum*, y su cometido principal es facilitar la transparencia del proceso *Scrum*. Un Sprint, suele tener una duración determinada, pero es bastante habitual que los equipos *Scrum* elijan tener *Sprints* de una duración de dos semanas. Sin embargo, cada caso es diferente, y es el equipo *Scrum* el que debe descubrir cual es su periodo mínimo necesario para generar valor a través de un Incremento terminado.

El *Sprint* también se puede definir como un meta evento que contiene todos los demás eventos. Un *Sprint normal* tendría:

- El *Sprint Planning* al comienzo del *Sprint*.
- *Daily Scrums* para actualizar el *Sprint Backlog* e identificar impedimentos.
- Un *Sprint Review* al final del *Sprint* para inspeccionar el Incremento.
- Justo después, la retrospectiva para hacer transparentes e inspeccionar posibles problemas en la forma de trabajar o de hacer *Scrum*.

4.2. Docker

El término *Docker* [20] se refiere a varias cosas. Esto incluye un proyecto comunitario de código abierto; herramientas para proyectos de código abierto; *Docker Inc.*, el patrocinador principal del proyecto; y herramientas respaldadas oficialmente por la empresa. El hecho de que la tecnología y la empresa compartan el mismo nombre puede generar confusión.

En lo que a este proyecto se refiere, podemos definir *Docker* como una tecnología de creación de contenedores que permite la creación y el uso de contenedores. Un contenedor es como una máquina virtual que elimina la necesidad de administrar directamente el hardware.

Docker Compose

Docker Compose, es una herramienta desarrollada para ayudar a definir y compartir aplicaciones de varios contenedores. Con *Compose*, puede crear un archivo *.yaml* para definir los servicios y, con un solo comando, ponerlo todo en marcha o eliminarlo [21].

Ventajas en el uso de Docker

Entre las múltiples ventajas que ofrece *Docker*, cabe destacar las siguientes [20] :

- **Modularidad:** El enfoque *Docker* para la creación de contenedores se centra en la capacidad de tomar una parte de una aplicación, para actualizarla o repararla, sin necesidad de tomar la aplicación completa.

- **Control de versiones de imágenes y capas:** Cada archivo de imagen de *Docker* se compone de una serie de capas. Estas capas se combinan en una sola imagen. Una capa se crea cuando la imagen cambia. Cada vez que un usuario especifica un comando, como ejecutar o copiar, se crea una nueva capa. *Docker* reutiliza estas capas para construir nuevos contenedores, lo cual hace mucho más rápido el proceso de construcción.
- **Restauración:** Toda imagen tiene capas, si no le gusta la iteración actual de una imagen puede restaurarla a la versión anterior.

4.3. Flask

Flask es un *micro* Framework escrito en Python y concebido para facilitar el desarrollo de Aplicaciones Web bajo el patrón MVC.

La palabra *micro*, no designa a que sea un proyecto pequeño o que nos permita hacer páginas web pequeñas, sino que al instalar *Flask* tenemos las herramientas necesarias para crear una aplicación web funcional. Si se necesita en algún momento una nueva funcionalidad, hay un conjunto muy grande extensiones que se pueden instalar con *Flask* que le van dotando de funcionalidad.

Además, *Flask* está basado en la especificación *WSGI de Werkzeug* [22].

Especificación WSGI de Werkzeug

La especificación *WSGI* (*Web Server Gateway Interface*) describe cómo un servidor web (Gunicorn, uWSGI, etc.) interactúa con una aplicación web (flask, django, etc.), y cómo la aplicación web maneja solicitudes. Es con la especificación *WSGI* que podemos ejecutar varias aplicaciones web en cualquier servidor web [23].

4.4. Otras Herramientas

A continuación, se enumeran las herramientas utilizadas (y no mencionadas) antes en el desarrollo de este Trabajo Fin de Máster:

- **Python:** Lenguaje de Programación de código abierto, orientado a objetos. Tiene una sintaxis sencilla que cuenta con una vasta biblioteca de herramientas, que hacen de *Python* un lenguaje de programación

único [24]. En el sistema utilizamos las versiones 2.7 para la Red Neuronal y la 3.7 para la Aplicación Web.

- **Miniconda:** Herramienta para gestionar y ordenar los paquetes o librerías de cada proyecto. Para el desarrollo del sistema usamos la versión 4.10.
- **Mariadb:** Sistema de Gestión de Bases de Datos derivado de *MySQL*. Para el desarrollo del sistema se ha usado la última versión disponible en *Docker Compose* que es la 3.1. y el conector de *Mariadb* para *Python* cuya versión es la 1.0.6.
- **Visual Studio Code:** Editor de código fuente en la cual se ha instalado los *plugins* de *Python* para poder ejecutar la aplicación *Flask*. La versión usada es la 1.46.1.
- **FortiClient V.P.N.:** Tecnología para conectarse de manera segura a la V.P.N. de la universidad para poder acceder al servidor *Gamma*. La versión usada es la 6.0.
- **Mobaxterm:** Terminal para *Windows* con cliente *SSH* y *SFTP*. Se ha usado para conectarse y controlar el servidor *Gamma*. La versión usada es la 12.4.
- **GitHub:** Sistema de control de versiones para guardar los avances obtenidos del sistema.
- **Overleaf:** Editor colaborativo de *LaTeX* basado en la nube. Es la herramienta usada para escribir este documento.
- **Cubase:** Serie de aplicaciones informáticas para editar audio digital, *MIDI* y un secuenciador de música. Se ha usado para comprobar las distintas pistas que nos da la salida de la I.A. y comprobar que realiza de manera correcta su funcionamiento. La versión usada es la 5.
- **VLC media player:** Reproductor multimedia libre y de código abierto multiplataforma. Se usa para comprobar de manera auditiva las salidas de la I.A. La versión usada es 3.0.
- **Balsamiq Mockup:** Herramienta para crear prototipos y bocetos de interfaces. La versión usada en este T.F.M. es la 4.2.4.

Capítulo 5

Aspectos relevantes del desarrollo del proyecto

En este quinto capítulo se detallan todos los aspectos más relevantes que han ocurrido durante el desarrollo del proyecto.

5.1. Funcionamiento de la Red Neuronal

El código fuente de la Red Neuronal descrita en el artículo [3] está disponible en un repositorio de *GitHub*, el problema ha surgido al intentar ejecutar dicho código, ya que no hay ninguna referencia a los requisitos necesarios para poder probarla, además de estar en *Python 2.7*.

Para intentar solucionar este problema, y tras analizar el código fuente, se ha empezado a instalar una a una las diferentes librerías y se ha probado con las distintas versiones disponible de cada una de ella, hasta dar con la correcta para el funcionamiento de la Red.

Tras un tedioso trabajo, se ha conseguido crear un archivo de requerimientos, para que, cualquiera a través de este fichero pueda instalar todas las librerías y ejecutar el sistema.

5.2. Comunicación entre Dockers

Uno de los objetivos parciales, era dividir cada parte del sistema en *Docker* independientes y que se comunicaran entre ellos. De manera teórica, todo debería funcionar de una manera bastante simple, en la práctica esto

ha sido una tarea bastante costosa, lo primero fue crear un *Docker* para cada parte del sistema e instalar en él todo lo necesario para el funcionamiento de dicha parte.

Por otro lado, la comunicación entre los contenedores ha sido una tarea bastante costosa de lograr, optando al final por crear un archivo *Docker Compose* para administrar cada uno de los contenedores.

Los problemas surgieron al tener poco conocimiento sobre el funcionamiento de *Docker* y esa falta de experiencia, ha supuesto bastante tiempo en prueba y error, mientras se leía documentación relacionada al tema.

5.3. Desarrollo de una Aplicación Flask

Para el desarrollo de la Aplicación Web, se decidió usar el *Framework Flask*, pero la inexperiencia en el desarrollo web y la particularidad de usar *Python* como lenguaje de programación (ya que es uno de los lenguajes con los que menos he trabajado), ha supuesto una formación inicial para poder empezar el desarrollo de la Aplicación Web.

Lo primero que realicé son varios cursos de *Udemy* para profundizar en el lenguaje de programación. Después, para conseguir cierta soltura con el lenguaje, estudié el tutorial que tiene la página oficial de *Flask*, dónde aprendí a utilizar los distintos componentes del *Framework* y las particularidades que este tiene.

Finalmente, los distintos problemas que han ido surgiendo con el desarrollo se han solventado a través de búsqueda de información por internet y consulta de manuales relativos al tema.

5.4. Comunicación entre la Aplicación Web y la Red Neuronal

Finalmente, una vez que tenemos cada parte del sistema en un contenedor *Docker* distinto, la Aplicación Web y la Red Neuronal funcionando sólo quedaba comunicar dichos elementos entre ellos.

La idea principal es que desde la Aplicación Web se envíe un fichero a la Red Neuronal para que esta lo procese y que una vez obtenida la salida, esta lo envíe a la Aplicación Web para que se almacene en la Base de Datos y el usuario pueda descargar dicha salida.

Entre las distintas posibilidades que existían, se decidió por usar dos herramientas distintas:

- **Socket:** Gracias a que tenía una base de conocimientos de redes, la comunicación vía *socket* no fue un problema, a excepción de lo mencionado anteriormente, la comunicación entre los *Docker* complicó un poco este tema, ya que hasta no solventar esta comunicación no se podía probar que realmente funcionaba la transferencia del archivo.
- **Post de Python:** A contrario que los *sockets*, todo lo relacionado sobre el Post de *Python*, fue algo totalmente nuevo para mí (por el cual estuve durante un tiempo buscando información relativa al problema y realizando distintas pruebas), pero era necesario usar esta técnica, ya que queríamos enviar a la Aplicación Web el archivo de salida de la Red Neuronal y que este envío no limitara el uso de dicha Aplicación.

Capítulo 6

Trabajos relacionados

Actualmente existen un amplio abanico de trabajos sobre el uso de la Inteligencia Artificial en la música, por lo que en este documento se van a mencionar los que se han considerado más relevantes.

6.1. DeepBach

El artículo *DeepBach: a Steerable Model for Bach Chorales Generation* [25] presenta un sistema destinado a modelar música polifónica.

El sistema después de ser entrenado en las armonizaciones corales de *Johann Sebastian Bach* es capaz de generar corales altamente convincentes en el estilo de *Bach*.

La fuerza de *DeepBach* proviene del uso de muestreo pseudo-Gibbs junto con una representación adaptada de datos musicales. Esta está en contraste con muchos enfoques automáticos de composición musical que tienden a componer música secuencialmente.

El sistema también es orientable en la forma de que un usuario puede restringir la generación imponiendo restricciones de posición como notas, ritmos o cadencias en la partitura generada.

En la figura 6.1, se muestra el funcionamiento del algoritmo basado en el método *LSTM* de redes neuronales recurrentes.

Para poner a prueba y validar el funcionamiento del sistema *DeepBach*, se llevó a cabo una encuesta realizada entre 1600 personas, entre las cuales se contaban más de 400 músicos. Para dicha prueba se tomaron diferentes muestras de los corales originales, así como de los creados automáticamente.

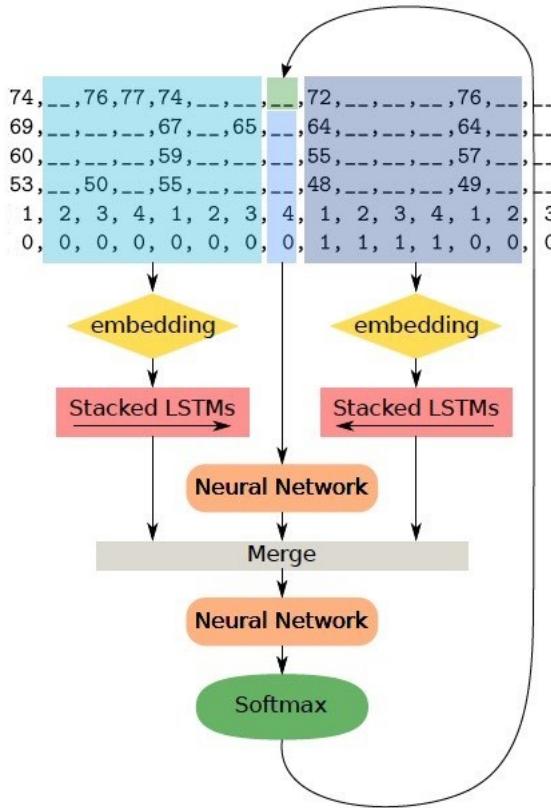


Figura 6.1: Esquema del algoritmo *DeepBach*.Figura extraída de [26]

Resultó que alrededor del 50 % de los encuestados consideraron originales las composiciones de *Johann Sebastian Bach* creadas automáticamente [27].

6.2. MuseNet

MuseNet [28] es una red neuronal profunda que puede generar composiciones musicales combinando varios estilos.

La red neuronal [29] ha sido entrenada en un conjunto de datos de archivos *MIDI* recopilados de una variedad de fuentes en línea que abarcan los estilos de música variados.

Los investigadores detrás del proyecto dicen que el sistema puede prestar atención a la música durante largos períodos de tiempo, lo que significa que es capaz de comprender el amplio contexto de las melodías de una canción y todos sus matices, en lugar de cómo fluyen juntos en una pequeña sección única. Así el sistema tiene la tarea de predecir la siguiente nota en una

secuencia. A medida que avanza la prueba, la I.A. se va desvinculando más y más de la pieza original, llegando a crear sus propias composiciones.

Algunas de las limitaciones de *MuseNet* son [28]:

- Los instrumentos que solicita son por sugerencias, no por requisitos. *MuseNet* genera cada nota calculando las probabilidades en todas las notas e instrumentos posibles. El modelo cambia para hacer que sus opciones de instrumentos sean más probables, pero siempre existe la posibilidad de que elija otra cosa.
- *MuseNet* tiene un momento más difícil con pares extraños de estilos e instrumentos (como *Chopin* con bajo y batería). Las generaciones serán más naturales si eliges los instrumentos más cercanos al compositor o al estilo habitual de la banda.

6.3. SaxEx

El artículo *SaxEx: a CBR system for generating expressive musical performances* [30] presenta un sistema inteligente basado en el razonamiento por casos.

SaxEx recibe como entrada una melodía simple interpretada por un humano y crea modificaciones sobre esta melodía que añaden expresividad a la pieza.

El método de resolución de problemas en la que se basa *SaxEx* sigue la descomposición de subtareas de los métodos *CBR*:

- **Recuperar** el conjunto de notas más similares al problema actual. Esta a su vez se divide en tres subtareas [31]:
 - **Identificar** los patrones de recuperación utilizando las estructuras de implicación y la importancia métrica de las notas.
 - **Búsqueda** de casos en la Biblioteca de Casos utilizando métodos de recuperación de *Noos* y patrones previamente construidos.
 - **Seleccionar y Clasificar** los casos recuperados utilizando los métodos de preferencia de *Noos*. Los métodos de preferencia utilizan criterios como similitud en la estabilidad armónica, duración de las notas o direcciones melódicas.

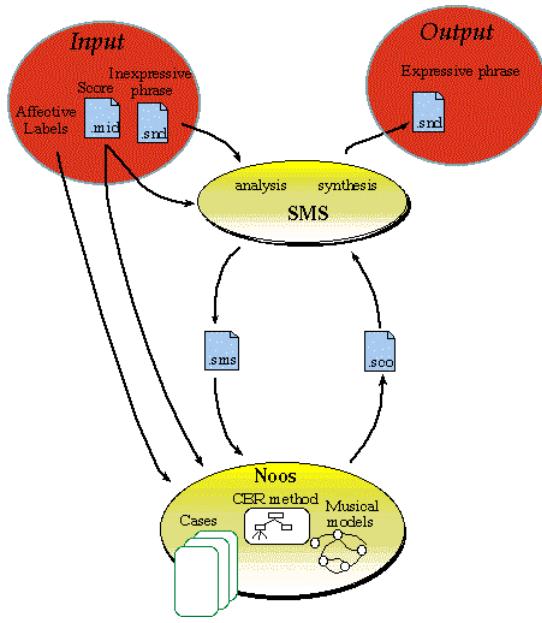


Figura 6.2: Diagrama de bloques de los componentes de *SaxEx*. Figura extraída de [31]

- **Adaptar** escogiendo un conjunto de transformaciones expresivas para aplicar en el problema actual del conjunto de casos más similares. Cuando hay varios casos considerados igualmente similares, se utiliza la regla de la mayoría y si no existiera un criterio valido, se selecciona al azar.
- **Incorporar** el nuevo problema resuelto a la Biblioteca de Casos para que esté disponible para razonamientos futuros.

Capítulo 7

Conclusiones y Líneas de trabajo futuras

Este capítulo analizará los objetivos propuestos y los completados, así como las competencias que se han adquirido durante el desarrollo de este Trabajo Fin de Máster. Se tratará también hacia dónde podría dirigirse un trabajo futuro relacionado con dicho proyecto.

7.1. Consecución de objetivos

El objetivo principal de este Trabajo Fin de Máster es **estudiar la implementación y funcionamiento de la Red Neuronal propuesta en el artículo «A Convolutional Approach To Melody Line Identification In Symbolic Scores» [3]** y ser capaz de usar dicha Red Neuronal a través de una Aplicación Web.

Al haber alcanzado el objetivo principal, los objetivos parciales se han cumplido como se muestra a continuación. Los objetivos parciales del proyecto son:

- Entender, analizar y ser capaz de usar la Red Neuronal propuesta en el artículo «A Convolutional Approach To Melody Line Identification In Symbolic Scores» [3].

Este objetivo parcial se ha completado de una manera satisfactoria ya que dicha Red Neuronal se está usando en la Aplicación Web.

- **Crear una Aplicación Web conectada a una Base de Datos en donde se envíe peticiones a la Red Neuronal y podamos almacenar la entrada y salida de dicha Red.**

Como se ha mencionado a lo largo de este documento, tenemos una Aplicación Web con persistencia en una Base de Datos *Mariadb* que hace llamadas a la Red Neuronal y es capaz de almacenar en la Base de Datos la salida. Con esto afirmamos que dicho objetivo parcial se ha completado de una manera satisfactoria.

- **Modularizar cada una de las partes del sistema (Red Neuronal, Aplicación Web y Base de Datos) en contenedores *Docker* y establecer una comunicación entre ambos contenedores.**

Actualmente cada parte del sistema esta en un contenedor *Docker* distintos pero que establecen una comunicación entre ellos. Por lo cual se concluye que este objetivo parcial también se ha completado con éxito.

7.2. Competencias

Las competencias demostradas en este T.F.M. tal y como se especifica en la página web de la universidad [32] son las siguientes:

- Capacidad para la aplicación de los conocimientos adquiridos y de resolver problemas en entornos nuevos o poco conocidos dentro de contextos más amplios y multidisciplinares, siendo capaces de integrar estos conocimientos.
- Capacidad para la integración de tecnologías y sistemas propios de la Ingeniería Informática, con carácter generalista, y en contextos más amplios y multidisciplinares.
- Capacidad para modelar, diseñar, definir la arquitectura, implantar, gestionar, operar y administrar y mantener aplicaciones, redes, sistemas, servicios y contenidos informáticos.
- Capacidad para diseñar y evaluar sistemas operativos y servidores, y aplicaciones y sistemas basados en computación distribuida.
- Capacidad para aplicar métodos matemáticos, estadísticos y de inteligencia artificial para modelar, diseñar y desarrollar sistemas inteligentes y sistemas basados en conocimiento.

- Capacidad para la creación y explotación de entornos virtuales, y para la creación y distribución de contenidos multimedia.
- Capacidad para llevar a cabo el proceso de análisis, diseño, desarrollo y evaluación de sistemas funcionales que permitan la solución de problemas reales combinando técnicas de ingeniería del software e inteligencia artificial.
- Obtener un conocimiento avanzado acerca de las técnicas y herramientas empleadas en la implementación de los aspectos de presentación e interacción con el usuario en entornos web.
- Capacidad para entender las características de los sistemas complejos y sus comportamientos emergentes, y utilizar las herramientas y técnicas básicas para su modelado, análisis, diseño y explotación.

7.3. Trabajo futuro

Tras analizar los resultados obtenidos a partir del desarrollo del sistema y su rendimiento con usuarios reales, se plantean diversas mejoras y avances a implementar en un futuro para incrementar la eficacia y usabilidad del sistema:

- Ampliar la Red Neuronal para que se pueda usar con mas estilos de música diferentes.
- Migrar la Red Neuronal de *Pyton 2* a *Python 3*.
- Actualizar las versiones de las librerías usadas en la Red Neuronal, como por ejemplo *Theano* y *Lasagna*.
- Ampliar el número de funcionalidades que el usuario puede hacer en la aplicación web, por ejemplo poder fusionar acompañamientos y melodías de las distintas salidas que se obtiene de la Red Neuronal.
- Permitir al usuario renderizar las salidas del sistema para que pueda verla de una manera visual.
- Crear una Biblioteca compartida en donde los usuarios puedan compartir sus archivos procesados.

Apéndice A

Plan de Proyecto Software

En este primer anexo se mostrará todo lo relativo a la planificación del Trabajo Fin de Máster, así como la viabilidad legal que va a tener dicho trabajo.

A.1. Planificación temporal

En esta sección se mostrarán los diferentes *Sprints* que se han realizado durante en el desarrollo del proyecto. Así como, las conclusiones tomadas en las reuniones de *Sprint Review*.

Sprint 0: Inicio del proyecto

En este primer *sprint* se ha establecido una serie de *items*:

- Planificar la forma de organización del T.F.M.
- Establecer el método de trabajo a seguir.
- Poner en común las diversas opiniones y sugerencias para la realización del T.F.M.

En la *Sprint Review*, se ha comentado sobre los diferentes enfoques de lo que puede tratar el T.F.M. planificando la búsqueda de documentación relacionada a estos.

Finalmente, se ha acordado seguir una metodología de trabajo tipo *Scrum*, asignando a cada *sprint* una duración de dos semanas.

Sprint 1: Herramientas y Entorno de trabajo

Los objetivos marcados para este primer *sprint* son los siguientes:

- Buscar documentación relacionada a los enfoques hablados en el *sprint* anterior.
- Crear la plantilla del documento en L^AT_EX.
- Crear un repositorio en *GitHub*.

En la *Sprint Review* se han comentado los diversos artículos relacionados y se ha realizado una criba hasta quedarse solo con tres de ellos, los cuales se han subido a una *Issue* del repositorio de *GitHub*.

Finalmente, para la plantilla del documento se ha decidido usar la herramienta *Overleaf*.

Sprint 2: Análisis de los artículos e Inicio de la Memoria

Los objetivos de este *sprint* son los siguientes:

- Empezar el documento con los capítulos de Introducción y Conceptos Teóricos.
- Analizar minuciosamente los artículos seleccionados.
- Complementar los artículos con un *Dataset*.

En la *Sprint Review* se han hecho un análisis los diferentes artículos y se ha seleccionado el que va a ser el principal del T.F.M.

Por otro lado, se ha complementado el artículo con un *Dataset* (que se ha añadido a una *Issue* del repositorio de *GitHub*).

Finalmente, se ha repasado todo lo redactado en el documento y se ha indicado ciertas pautas para continuar.

Sprint 3: Configuración inicial, Análisis del Artículo y Fundamentos

Los objetivos de este *sprint* son los siguientes:

- Completar fundamentos básicos de Música y la I.A. del documento.
- Configurar la V.P.N de la universidad para futuros desarrollos.
- Analizar en profundidad el artículo [3] elegido y obtener el código fuente.

En la *Sprint Review* se ha revisado el artículo y visto como está estructurado el código fuente.

Finalmente, se ha repasado lo redactado en el documento y decidido añadir un apartado en el documento explicando el contenido del artículo.

Sprint 4: Puesta en marcha de la Red Neuronal y Explicación del Artículo

Los objetivos de este *sprint* son los siguientes:

- Escribir apartado con la explicación del artículo.
- Obtener acceso a la máquina de altas prestaciones.
- Montar la infraestructura de codificación y pruebas.
- Descargar el código fuente.

En la *Sprint Review* se ha revisado toda la configuración realizada en la máquina de altas prestaciones para poder empezar a probar la Red Neuronal.

Por otro lado, se ha obtenido y analizado los casos de pruebas que se mencionan en el artículo para futuras pruebas.

Finalmente, se ha repasado el apartado en donde se ha explicado el artículo.

Sprint 5: Instalación de la Red Neuronal

Los objetivos de este *sprint* son los siguientes:

- Crear una nueva rama del código fuente en *GitHub*.
- Revisar, instalar y actualizar los paquetes necesarios para que el código funcione.
- Realizar las pruebas propuestas en el artículo [3].

En la *Sprint Review* se ha mostrado y analizado el funcionamiento de la Red Neuronal, analizando las entradas y salidas. Las pruebas se han realizado con los casos de pruebas obtenidos en el *sprint* anterior.

Sprint 6: Documentación librerías y Entrenamiento de la Red Neuronal

Los objetivos de este *sprint* son los siguientes:

- Ser capaz de entrenar nuestros propios modelos con el sistema.
- Documentar la lista de todas las librerías y versiones usadas en el sistema.

En la *Sprint Review* se ha mostrado el fichero en donde se especifican todas las librerías y sus versiones. Dicho fichero se ha añadido a *GitHub*.

Finalmente, se ha enseñado como se entrena el sistema para que sea capaz de reconocer diferentes estilos.

Sprint 7: Modelado de la Aplicación Web

Los objetivos de este *sprint* son los siguientes:

- Estudiar la creación de una Aplicación Web.
- Hacer el modelado de la aplicación que usará el sistema y añadirla en la memoria.

En la *Sprint Review* se ha hecho un análisis de los *Mockup* de la Aplicación Web.

Finalmente, se ha mencionado una serie de documentación y cursos para obtener los conocimientos fundamentales de la programación web.

Sprint 8: Inicio de la Aplicación Web y Tablas del Sistema

Los objetivos de este *sprint* son los siguientes:

- Hacer el Diagrama de Tablas del Sistema.
- Investigar y estudiar como se realiza la programación web y Flask.

En la *Sprint Review* se ha hecho mostrado el diagrama de Tablas que va a tener el sistema y se ha mencionado el avance en la formación de programación web.

Sprint 9: Programación de la Aplicación Web I

Los objetivos de este *sprint* son los siguientes:

- Conectar la aplicación con MariaDB.
- Crear diagrama Entidad-Relación de la estructura de la BBDD.
- Realizar Login del sistema.
- Realizar Pantalla Principal y Carga de fichero.
- Crear Pantalla de Biblioteca del usuario.

En la *Sprint Review* se ha hecho mostrado el funcionamiento del Login del sistema, y se ha enseñado y discutido las otras pantallas del sistema.

Finalmente, se ha enseñado el diagrama Entidad-Relación de la estructura de la BBDD.

Sprint 10: Programación de la Aplicación Web II

Los objetivos de este *sprint* son los siguientes:

- Crear consultas para almacenar los ficheros en la BBDD.
- Preparar recibimiento del archivo por parte de la I.A.
- Añadir funcionalidades para convertir ficheros a binario y viceversa.

En la *Sprint Review* se ha comentado de manera conceptual como se ha implementado cada una de las historias del *sprint*.

Sprint 11: Docker I

Los objetivos de este *sprint* son los siguientes:

- Crear un *Docker* con *Mariadb*.
- Crear un *Docker* con la aplicación web.
- Conectar los dos contenedores *Dockers* anteriores.

En la *Sprint Review* se ha enseñado como se ha creado los contenedores *Docker* y como se han comunicado entre ellos.

Sprint 12: Docker II

Los objetivos de este *sprint* son los siguientes:

- Crear un *Docker* con la Red Neuronal.
- Comunicar *Docker* con la Aplicación Web con el *Docker* de la Red Neuronal

En la *Sprint Review* se ha enseñado como se ha creado el contenedor *Docker* de la Red Neuronal y como se han comunicado con el *Docker* Web.

Sprint 13: Comunicación Web - Red Neuronal I

Los objetivos de este *sprint* son los siguientes:

- Conectar y enviar un fichero mediante *Socket* a la Red Neuronal y procesarlo.
- Crear Servidor para administrar ficheros y funcionamiento de la Red Neuronal.

En la *Sprint Review* se ha mostrado como la Aplicación Web envía un fichero a la Red Neuronal y esta lo procesa.

Finalmente, se ha enseñado como se ha implementado el Servidor de la Red Neuronal.

Sprint 14: Comunicación Web - Red Neuronal II

Los objetivos de este *sprint* son los siguientes:

- Conectar y enviar el fichero procesado de la Red Neuronal a la Aplicación Web.
- Guardar el fichero procesado en la BBDD.
- Modificar Servidor de la Red Neuronal para que permita recibir más peticiones.

En la *Sprint Review* se ha mostrado como a través de *Post Python* se manda el fichero procesado y se guarda en la BBDD, y como el Servidor de la Red Neuronal permite procesar otro archivo (si la Red no está ocupada).

Sprint 15: Funcionalidad Biblioteca y Documentación

Los objetivos de este *sprint* son los siguientes:

- Implementar las funcionalidades de la Biblioteca del usuario: ver lista de ficheros y permitir descargar el fichero original y el procesado.
- Continuar con la documentación.

En la *Sprint Review* se ha mostrado las funcionalidades de la biblioteca.

Finalmente, se ha revisado lo avanzado en la documentación.

Sprint 16: Internacionalización

Los objetivos de este *sprint* son los siguientes:

- Implementar la funcionalidad de internacionalización de la Aplicación Web.
- Continuar con la documentación.

En la *Sprint Review* se ha mostrado como se ha realizado la internacionalización.

Finalmente, se ha revisado lo avanzado en la documentación.

Sprint 17: Cambios de estilos en la Aplicación Web

Los objetivos de este *sprint* son los siguientes:

- Cambiar ciertos estilos para obtener una interfaz más amigable.
- Continuar con la documentación.

En la *Sprint Review* se ha mostrado los cambios realizados en la interfaz de la Aplicación Web.

Finalmente, se ha revisado lo avanzado en la documentación.

Sprint 18: Perfilamiento final del Sistema y de la Documentación

Los objetivos de este *sprint* son los siguientes:

- Realizar las ultimas modificaciones en la Aplicación Web.
- Continuar con la documentación hasta tener una versión previa terminada.

En la *Sprint Review* se ha mostrado los cambios realizados en la Aplicación Web.

Finalmente, se ha revisado la documentación y discutido ámbitos de mejoras.

Sprint 19: Limpieza de Código y Documentación final

Los objetivos de este *sprint* son los siguientes:

- Limpieza de código para tener una versión final de código fuente.
- Terminar la documentación con las pautas recibidas en el *sprint* anterior.

En la *Sprint Review* se ha mostrado los cambios realizados en la documentación.

Sprint 20: Pruebas finales y Creación Presentación

Los objetivos de este *sprint* son los siguientes:

- Realizar pruebas finales del sistema completo.
- Crear una Presentación para la defensa del T.F.M.

En la *Sprint Review* se ha mostrado la presentación y se ha dado unas pautas de cara a la defensa del T.F.M.

A.2. Viabilidad Legal

Para definir la viabilidad legal del T.F.M. se ha usado la licencia más restrictiva que es la relativa al código fuente del artículo [3]. La viabilidad es la *Licencia MIT*, que permite:

- **Permisos:**

- Uso comercial
- Modificación
- Distribución
- Uso privado

- **Limitaciones:**

- Responsabilidad
- Garantía

En otras palabras, se otorga permiso, sin cargo, a cualquier persona que obtenga una copia de este software y los archivos de documentación asociados (el *Software*), para tratar en el *Software* sin restricciones, incluidos, entre otros, los derechos para usar, copiar, modificar, fusionar, publicar, distribuir, sublicenciar y/o vender copias del *Software*, y para permitir a las personas a las que el *Software* es amueblado para ello, sujeto a las siguientes condiciones:

El software se proporciona «tal cual», sin garantía de ningún tipo, expresa o implícitas, incluidas, pero no limitadas a las garantías de comerciabilidad, aptitud para un propósito particular y no infracción.

En ningún caso los autores o titulares de los derechos de autor serán responsables de cualquier reclamo, daños u otra responsabilidad, ya sea en una acción de contrato, agravio o de otro modo, que surja de, fuera o en relación con el software o el uso u otros negocios en el software.

Apéndice B

Especificación de Requisitos

B.1. Introducción

En este apéndice se van a mostrar los objetivos que se persiguen en este Trabajo Fin de Máster, así como todos los requisitos funcionales y no funcionales que se pretenden alcanzar.

B.2. Objetivo general

El objetivo principal de este Trabajo Fin de Máster es estudiar la implementación y funcionamiento de la Red Neuronal propuesta en el artículo «A Convolutional Approach To Melody Line Identification In Symbolic Scores» [3] y ser capaz de usar dicha Red Neuronal a través de una Aplicación Web.

Objetivos Parciales

- Entender, analizar y ser capaz de usar la Red Neuronal propuesta en el artículo «A Convolutional Approach To Melody Line Identification In Symbolic Scores» [3].
- Crear una Aplicación Web conectada a una Base de Datos en donde se envíe peticiones a la Red Neuronal y podamos almacenar la entrada y salida de dicha Red.

- Modularizar cada una de las partes del sistema (Red Neuronal, Aplicación Web y Base de Datos) en contenedores *Docker* y establecer una comunicación entre todos los contenedores.

B.3. Especificación de requisitos

En esta sección se mostrará la lista de los requisitos funcionales y no funcionales del T.F.M.

Requisitos Funcionales

- Cada parte del sistema debe estar en un contenedor *Docker* distinto, para facilitar la ejecución de todo el sistema se creará un *DockerFile* por cada parte del sistema y un *Docker Compose* que se encargue de gestionarlos.
- La Aplicación Web, la Red Neuronal y la BBDD funcionarán de manera independiente.
- En la Base de Datos se guardará toda la información relativa al usuario además de toda la información relativa a las canciones y en su defecto a los ficheros cargados y procesados por la aplicación web y la Red Neuronal.
- La aplicación web será realizada en *Flask*.
- La Web tendrá control de usuarios.
- La Web permitirá procesar las melodías, descargarlas, gestionarlas...
- La comunicación de la aplicación web y la Red Neuronal se realizará mediante *Socket*, en la cual la aplicación web enviará una petición al *Docker* donde se encuentra la Red Neuronal para que procese una canción, y cada vez que se recargue la página de la aplicación web se enviará una petición al *Docker* donde se encuentra la Red Neuronal para comprobar si está listo el procesado del fichero y poder guardar dicho fichero en una ruta específica y añadir los datos relativos en la Base de Datos.

Requisitos no Funcionales

- Los datos que se manejan en la aplicación deberán ser privados, pudiendo usarse sólo desde la aplicación. Desde fuera solo el administrador podrá acceder a los datos.
- El proceso de comunicación entre la aplicación web y la Red Neuronal debe ser transparente al usuario.
- El tiempo de respuesta de la salida de la Red Neuronal variará con el volumen de peticiones que esta tenga, por ello se mostrará al usuario en qué proceso se encuentra su petición.
- La aplicación web deberá impedir que usuarios sin identificar accedan a la información almacenada.
- La aplicación web deberá impedir que se añadan datos a la base de datos sin una fase de negociación satisfactoria.
- Se usará *MariaDB* como gestor de la Base de Datos.

Apéndice C

Especificación de diseño

En este apéndice se va a mostrar el diseño realizado para el sistema. Para ello se mostrarán bocetos de la interfaz deseada, la estructura de la base de datos y otros diagramas para la comprensión de la estructura y funcionamiento del sistema.

C.1. Diagrama de Casos de Uso

En la figura C.1 se muestra el diagrama de Casos de Uso del sistema.

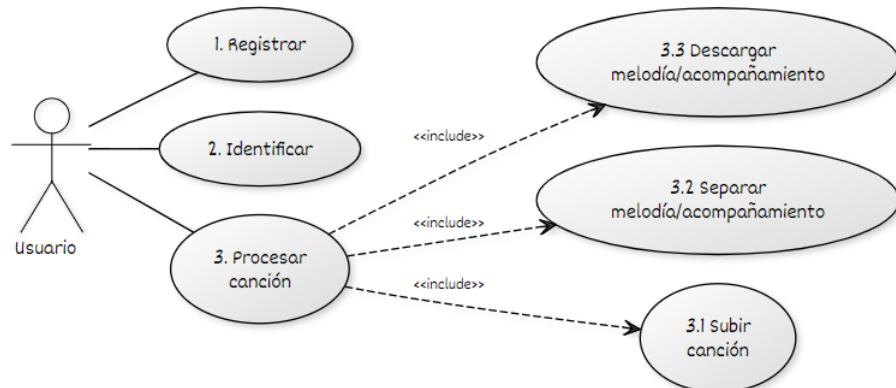


Figura C.1: Diagrama general de Casos de Uso

Caso de Uso Registrar	
Actor Principal	Usuario
Pre-condición	Ninguna
Post-condición	Entrar en el Sistema
Pasos	<ol style="list-style-type: none"> 1. Introducir nombre de Usuario. 2. Introducir nombre. 3. Introducir Contraseña. 4. Introducir Dirección de Correo.
Excepciones	<ol style="list-style-type: none"> 1. Nombre de Usuario ya existe en el Sistema. 2. Dirección de Correo con formato incorrecto.

Tabla C.1: Tabla Caso de Uso Registrar

Caso de Uso Identificar	
Actor Principal	Usuario
Pre-condición	Ninguna
Post-condición	Entrar en el Sistema
Pasos	<ol style="list-style-type: none"> 1. Introducir nombre de Usuario. 2. Introducir Contraseña.
Excepciones	<ol style="list-style-type: none"> 1. Nombre de Usuario incorrecto. 2. Contraseña incorrecta.

Tabla C.2: Tabla Caso de Uso Identificar

Caso de Uso Procesar Canción	
Actor Principal	Usuario
Pre-condición	Identificarse en el Sistema
Post-condición	Descargar canción Procesada
Pasos	<ol style="list-style-type: none"> 1. Subir Canción. 2. Separar Melodía/Acompañamiento. 3. Descargar Melodía/Acompañamiento.
Excepciones	<ol style="list-style-type: none"> 1. Canción con mismo nombre en el Sistema. 2. No tener ningún fichero seleccionado. 3. Fallo en el procesamiento de la canción. 4. Botón Descarga Procesado deshabilitado.

Tabla C.3: Tabla Caso de Uso Procesar Canción

Caso de Uso Subir Canción	
Actor Principal	Usuario
Pre-condición	Identificarse en el Sistema
Post-condición	Enviar fichero a Procesar
Pasos	<ol style="list-style-type: none"> 1. Introducir nombre de la Canción. 2. Seleccionar fichero a procesar.
Excepciones	<ol style="list-style-type: none"> 1. Canción con mismo nombre en el Sistema. 2. No tener ningún fichero seleccionado.

Tabla C.4: Tabla Caso de Uso Subir Canción

Caso de Uso Separar Melodía/Acompañamiento	
Actor Principal	Usuario
Pre-condición	Enviar una canción a Procesar
Post-condición	Archivo Procesado
Pasos	<ol style="list-style-type: none"> 1. Cargar la Red entrenada. 2. Procesar canción con la Red. 3. Devolver canción procesada
Excepciones	<ol style="list-style-type: none"> 1. Fallo en el procesamiento de la canción.

Tabla C.5: Tabla Caso de Uso Separar Melodía/Acompañamiento

Caso de Uso Descargar Melodía/Acompañamiento	
Actor Principal	Usuario
Pre-condición	Enviar una canción a Procesar
Post-condición	Canción Procesada
Pasos	<ol style="list-style-type: none"> 1. Pulsar botón descargar del fichero deseado.
Excepciones	<ol style="list-style-type: none"> 1. La canción no ha sido procesada.

Tabla C.6: Tabla Caso de Uso Descargar Melodía/Acompañamiento

C.2. Diseño Base de Datos

En esta sección se va a mostrar cómo se ha planteado el diseño conceptual de la Base de datos del sistema.

Diagrama Entidad-Relación

El diagrama de Entidad-Relación ver figura C.2 muestra cómo se relacionan las tres entidades de la base de datos.

Como se puede ver, un Usuario puede tener de ninguna Canción a N canciones en el sistema y por cada Canción se va a tener cero o dos ficheros (el original y el procesado).



Figura C.2: Diagrama de diseño

Diagrama Relacional

El diagrama de diseño como se puede ver en la figura C.3 muestra que la Base de Datos final esta formada por tres tablas:

- **Usuario:** En esta tabla se almacena toda la información relativa a los usuarios del sistema. El atributo Usuario debe ser único para cada usuario del sistema.
- **Canción:** En esta tabla se almacenan la información relativa a las canciones. El atributo Procesado hace referencia a si ya se ha procesado el archivo original y se ha creado el nuevo fichero de salida.
- **Fichero:** En esta tabla se guarda la información relativa a los archivos (tanto originales como procesados) de las canciones.



Figura C.3: Diagrama Relacional

C.3. Diagrama de Secuencia

En la figura C.4 se muestra el diagrama de Secuencia del sistema. En dicho diagrama se ilustra como el usuario interactúa con el sistema, y las comunicaciones/acciones que hace el sistema por dentro.

Como se puede observar, en el diagrama se encuentran 3 partes bien diferenciadas: Aplicación Web, BBDD y Red Neuronal. Cada una de estas partes corresponde a un contenedor *Docker* distinto.

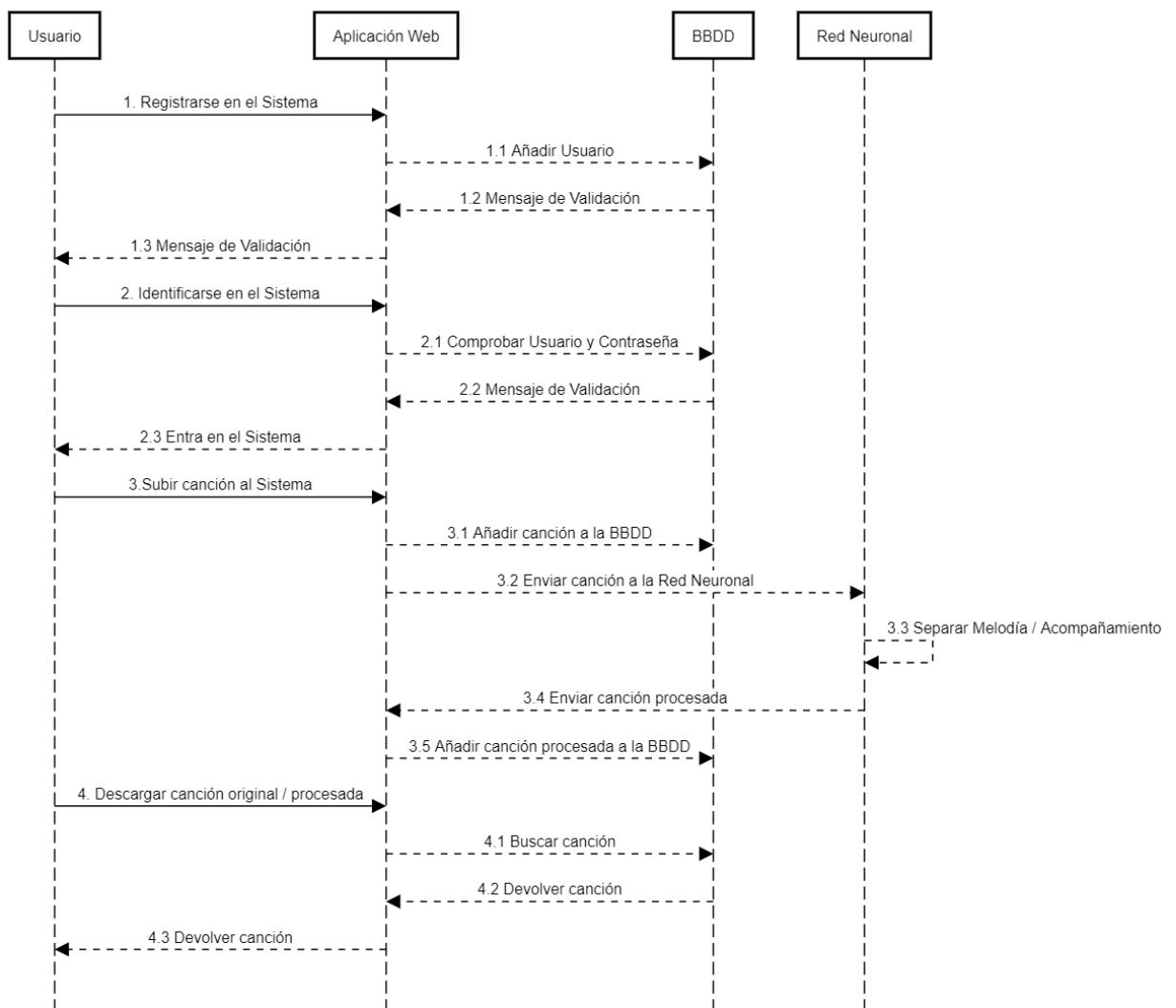


Figura C.4: Diagrama de Secuencia

C.4. Diagrama de Despliegue

El diagrama de Despliegue es el que se observa en la figura C.5. En dicho diagrama se ha indicado cada contenedor *Docker*, lo que contiene y se ha definido la comunicación que hay entre ellos.

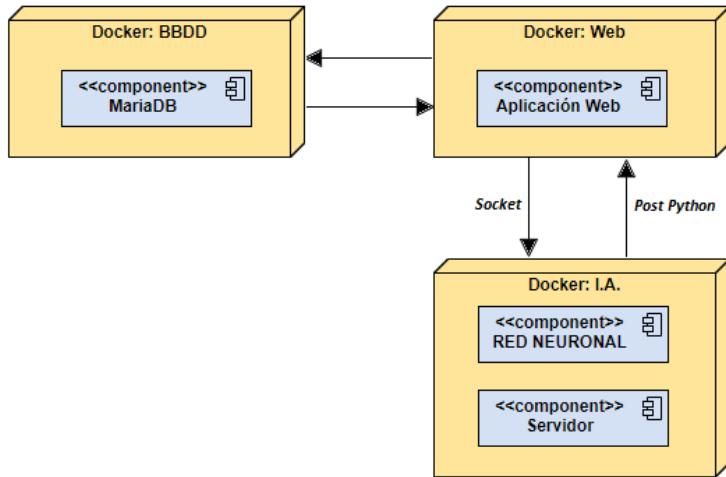


Figura C.5: Diagrama de Despliegue

C.5. Diseño de interfaz

El diseño del interfaz del sistema es el siguiente:

Login del sistema

Tal y como se muestra en las imágenes C.6 y C.7, la primera pantalla para acceder al sistema es la de Login/Registro, en la cual si el usuario no tiene cuenta podrá crear una.

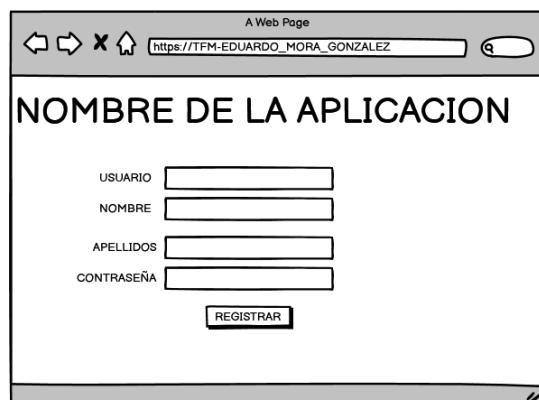
Inicio del sistema

Una vez que el usuario ha entrado en el sistema verá la pagina principal donde se mostrará la información relativa del sistema. En dicha página se le permitirá la opción de subir un archivo al sistema para procesarlo como se ve en C.8.



Este es un prototipo de interfaz de usuario para la aplicación de login. La pantalla tiene un encabezado "LOGIN APP" y un barra superior con iconos de navegación. El título de la página es "NOMBRE DE LA APLICACION". Los campos de entrada son "USUARIO" y "CONTRASEÑA", cada uno con su respectivo cuadro vacío. Abajo de los cuadros hay dos botones: "REGISTRAR" y "INICIO".

Figura C.6: Prototipo de Login en el sistema



Este es un prototipo de interfaz de usuario para la aplicación de registro. La pantalla tiene un encabezado "A Web Page" y una barra superior con iconos de navegación y la URL "https://TFM-EDUARDO_MORA_GONZALEZ". El título de la página es "NOMBRE DE LA APLICACION". Los campos de entrada son "USUARIO", "NOMBRE", "APELLIDOS" y "CONTRASEÑA", cada uno con su respectivo cuadro vacío. Abajo de los cuadros hay un solo botón "REGISTRAR".

Figura C.7: Prototipo de Registro en el sistema

El sistema cargará el archivo y cuando se haya obtenido el resultado se le mostrará un mensaje de finalización como se ve en las figuras C.9 y C.10.

Biblioteca del sistema

El usuario podrá ver en todo momento las canciones subidas y procesadas para que pueda reproducirlas, volver a descargar o modificar sus datos como se puede ver en la figura C.11.

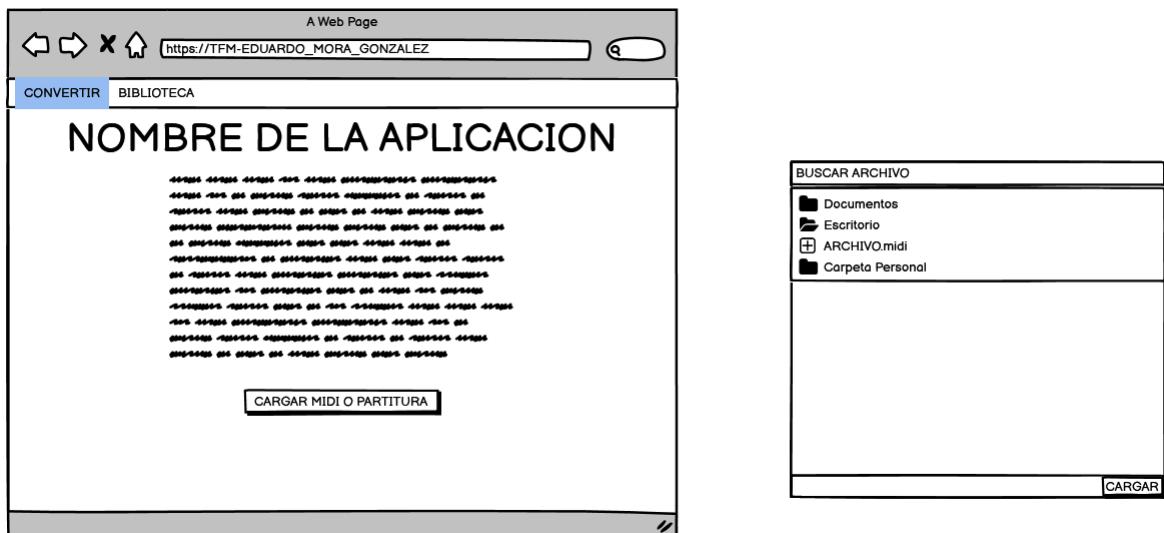


Figura C.8: Carga del fichero

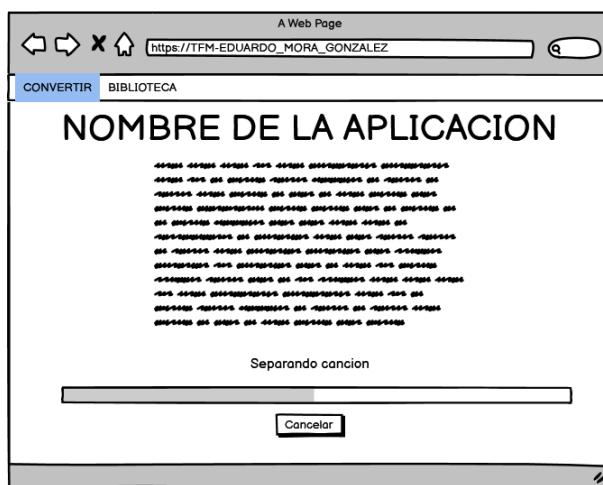


Figura C.9: Proceso de conversión

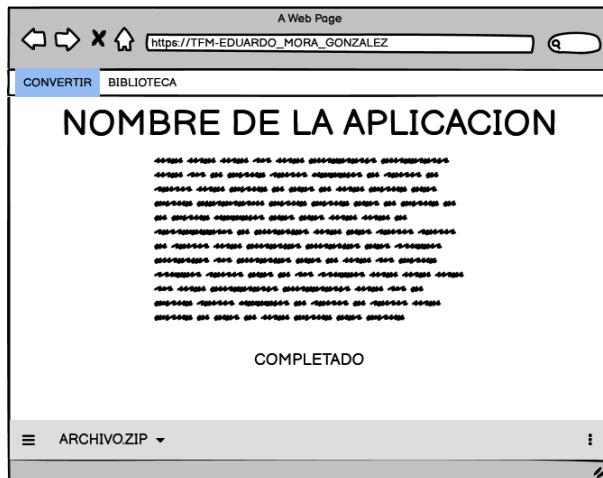


Figura C.10: Descarga de la salida

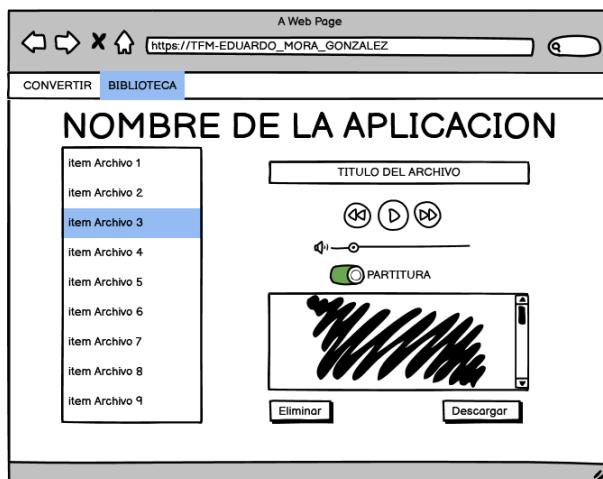


Figura C.11: Biblioteca del sistema

Apéndice D

Documentación técnica de programación

D.1. Introducción

En este apéndice se va a mostrar todo lo relacionado con la parte del desarrollo del sistema.

D.2. Estructura del Sistema

Este T.F.M. ha sido dividido en tres partes bien diferenciadas:

- Inteligencia Artificial
- Aplicación web
- Base de Datos

Cada una de las partes ha sido desplegada en un *Docker* distinto. Además, como cada contenedor *Docker* obtiene el código fuente de *Github* se ha creado un repositorio para cada contenedor.

Inteligencia Artificial

El código fuente relativo a la Inteligencia Artificial se encuentra en el repositorio [Symbolic-Melody-Identification](#).

Entre los distintos archivos y directorios del repositorio caben destacar los siguientes:

- **requirements.yml**: En este fichero se encuentran todas las librerías necesarias para poder ejecutar la Red Neuronal.
- **server.py**: Este es el fichero que lanza el *DockerFile* cuando se inicializa el contenedor. En dicho fichero se encuentra toda la lógica para recibir un fichero, ejecutar la Red Neuronal y devolver el fichero procesado a la Aplicación Web.
- **./usage**: En este directorio se explica el funcionamiento de la Red Neuronal: entrenamiento de modelos, ejecución...
- **./Pruebas TFM**: En este directorio se tienen los ficheros con los cuales se ha probado la Red Neuronal.

Aplicación web

El código fuente relativo a la Aplicación Web se encuentra en el repositorio [Aprendizaje De Melodías](#).

Entre los distintos archivos y directorios del repositorio caben destacar los siguientes:

- **main.py**: Este es el fichero que lanza el *DockerFile* cuando se inicializa el contenedor. En dicho fichero se encuentra toda la lógica de la aplicación.
- **./templates**: En este directorio, se encuentra todo el código *HTML* de cada una de las distintas pantallas de la aplicación.
- **./static**: En este directorio se tienen los ficheros estáticos del sistema, como estilos, imágenes...

Base de Datos

El código fuente relativo a la Base de Datos se encuentra en el repositorio [Aprendizaje De Melodías/BBDD](#).

En dicho directorio del repositorio se encuentra el fichero **TFM.sql** en donde se describe la estructura de tablas del Sistema.

D.3. Manual del programador

En este apartado, se hace mención de la estructura de los distintos *DockerFiles* que componen el sistema.

DockerFile Inteligencia Artificial

El *DockerFile* es el siguiente:

```
1  FROM continuumio/miniconda
2
3  RUN apt-get update
4  RUN apt-get install -y git
5  RUN apt-get install -y gcc
6
7  WORKDIR /app
8  RUN git clone https://github.com/Masetto97/Symbolic-Melody-
9      Identification.git
10 WORKDIR ./Symbolic-Melody-Identification
11 RUN git pull
12
13 RUN conda init bash
14 RUN conda env create -f requirements.yml
15
16 RUN conda run -n IA pip install --upgrade https://github.com/
17     Theano/Theano/archive/master.zip
18 RUN conda run -n IA pip install --upgrade https://github.com/
19     Lasagne/Lasagne/archive/master.zip
20 RUN conda run -n IA pip install scikit-learn==0.20.4
21 RUN conda run -n IA pip install matplotlib==2.2.5
22 RUN conda run -n IA pip install music21==2.1.0
23 RUN conda run -n IA pip install fastcluster==1.1.26
24 RUN conda run -n IA pip install midiutil==1.2.1
25 RUN apt-get install -y g++
26
27 CMD ["python", "server.py"]
```

Como se observa en la línea 1, el contenedor elegido es el oficial de *miniconda*.

Desde la línea 3 hasta la 5 se instala todo lo necesario para poder obtener el código fuente del repositorio.

A partir de la línea 7 hasta la 10, se gestiona todo lo relativo a obtener y actualizar el código fuente.

Una vez que tenemos ya todo el código, empezamos a instalar las librerías necesarias para ejecutar la Red Neuronal, para esta instalación hacemos uso del fichero *requirements.yml* mencionado anteriormente y la herramienta *pip de Python*.

Finalmente, la línea 24 es lo que se va a lanzar cuando se inicie el contenedor.

DockerFile Aplicación Web

El *DockerFile* es el siguiente:

```
1  FROM ubuntu:latest
2
3  RUN apt-get update
4  RUN apt-get install -y git
5  RUN apt-get install -y libmariadb-dev
6  RUN apt --fix-broken install
7  RUN apt-get install -y python3
8  RUN apt install -y python3-pip
9  RUN pip3 install mariadb
10 RUN pip3 install flask
11
12 WORKDIR /app
13 RUN git clone https://github.com/Masetto97/
14     APRENDIZAJE_DE_MELODIAS.git
15 WORKDIR ./APRENDIZAJE_DE_MELODIAS
16 RUN git pull
17 EXPOSE 5000
18
19 CMD ["python3","main.py"]
```

Como se observa en la línea 1, el contenedor elegido es el oficial de *Ubuntu*.

Desde la línea 3 hasta la 10 se instala todo lo necesario para poder obtener el código fuente del repositorio.

A partir de la línea 12 hasta la 16, se gestiona todo lo relativo a obtener y actualizar el código fuente. Además, se expone el puerto de comunicación.

Finalmente, la línea 18 es lo que se va a lanzar cuando se inicie el contenedor.

Docker-Compose

Una vez explicado la estructura de los *DockerFiles*, se procederá a explicar como está montado el *Docker-Compose*:

```
1  version: '3.7'
2
3  networks:
4      external:
5          name: network
6
```

```

7   services:
8
9     ia:
10       image: docker_ia
11       ports:
12         - "10000:10000"
13       tty: true
14       stdin_open: true
15       container_name: ia
16       networks:
17         - external
18
19     web:
20       image: docker_web
21       ports:
22         - "5000:5000"
23       tty: true
24       stdin_open: true
25       container_name: web
26       networks:
27         - external
28       depends_on:
29         - ia
30         - mariadb
31
32     mariadb:
33       image: mariadb:latest
34       environment:
35         TZ: Europe/Madrid
36         MYSQL_ROOT_PASSWORD: root
37         MYSQL_DATABASE: TFM
38         MYSQL_USER: user
39         MYSQL_PASSWORD: password
40       container_name: mariadb
41       ports:
42         - "3306:3306"
43       volumes:
44         - ./BBDD/TFM.sql:/docker-entrypoint-initdb.d/TFM.sql
45       networks:
46         - external

```

El fichero se divide en 2 partes bien diferenciadas:

- **Creación de Red:** esta primera parte abarca desde la línea 3 a la 5, en donde se crea una red de comunicación para todos los contenedores.
- **Creación de Servicios:** esta segunda sección se divide a su vez en 3 partes, una por cada contenedor:
 - **Servicio ia:** Este servicio llama a al *Dockerfile* de la Inteligencia Artificial (como se puede ver en la línea 10). Además, abre el puerto 1000 para comunicaciones (como esta indicado en la línea 12) y hace uso de la red creada anteriormente.

- **Servicio web:** Este servicio llama a al *Dockerfile* de la Aplicación Web (como se puede observar en la línea 20). Además, abre el puerto 5000 para comunicaciones (como está indicado en la línea 20), hace uso de la red creada anteriormente. Finalmente, para establecer una comunicación entre los *Docker*, se ha tenido que crear una dependencia como se puede ver desde la línea 28 a la 30.
- **Servicio Mariadb:** Este último servicio no tira de ningún fichero, como se puede ver en la línea 33 se obtiene la última versión *Docker* de *Mariadb*. Después, entre la línea 36 a la 42 se hace todo lo relativo a la configuración (contraseñas, puertos...) y finalmente en la línea 44 se hace la carga de la estructura de tablas (con el fichero mencionado en el punto anterior).

D.4. Compilación, instalación y ejecución del proyecto

Para compilar, instalar y ejecutar el sistema se ha creado un fichero *Bash* llamado «run.sh» que es el encargado de automatizar todo este proceso. Las opciones que permite este fichero son:

- **Instalar:** esta opción permite ejecutar todo el proceso de instalación descrito en cada uno de los *DockerFile*. El comando para ejecutar esta opción es:

```
1 sh run.sh install
```

- **Ejecutar:** esta opción permite iniciar todos los contenedores *Docker* y ejecutar los servicios de cada contenedor. El comando para ejecutar esta opción es:

```
1 sh run.sh start
```

- **Eliminar:** esta opción elimina los contenedores. El comando para ejecutar esta opción es:

```
1 sh run.sh remove
```

D.5. Pruebas del sistema

En este apartado se realizarán una serie de pruebas funcionales para probar el perfecto funcionamiento del sistema. Indicar que debido a que es una aplicación web no se ha podido probar formalmente mediante pruebas automatizadas por lo que se han creado una serie de pruebas funcionales.

- **Registrar un Usuario que ya existe:** intentamos añadir un usuario que ya existe en la BBDD. Figuras D.1 y D.2.

The screenshot shows a mobile application interface titled 'REGISTRO AL SISTEMA'. At the top, there are two tabs: 'LOGIN' and 'REGISTRO', with 'REGISTRO' being the active tab. Below the tabs are four input fields with icons: a user icon followed by 'edu', a user icon followed by 'edu', a lock icon followed by '...', and an envelope icon followed by 'edu@edu.es'. At the bottom is a green 'Registrar' button.

Figura D.1: Registrar un Usuario existente

The screenshot shows the same mobile application interface as Figure D.1. The input fields are identical: user 'edu', user 'edu', '...', and email 'edu@edu.es'. A new message '¡El usuario ya existe!' (The user already exists!) is displayed below the input fields. The 'Registrar' button is at the bottom.

Figura D.2: Salida registrar un Usuario existente

- **Poner un correo con formato incorrecto en el registro:** intentamos añadir un usuario y ponemos mal el correo electrónico. Figuras D.3 y D.4.

The screenshot shows a registration form titled "REGISTRO AL SISTEMA". The "REGISTRO" tab is selected. There are four input fields: 1. First Name: "edu" (with a person icon). 2. Last Name: "edu" (with a person icon). 3. Password: "..." (with a lock icon). 4. Email: "edu@edu.es" (with an envelope icon). Below the fields is a green "Registrar" button.

Figura D.3: Registrar un Usuario con correo erróneo

The screenshot shows a registration form titled "REGISTRO AL SISTEMA". The "REGISTRO" tab is selected. There are four input fields: 1. First Name: "prueba" (with a person icon). 2. Last Name: "prueba" (with a person icon). 3. Password: "....." (with a lock icon). 4. Email: "prueba" (with an envelope icon). Below the fields is a green "Registrar" button. A yellow warning box at the bottom left contains the text: "Incluye un signo '@' en la dirección de correo electrónico. La dirección 'prueba' no incluye el signo '@'."

Figura D.4: Salida registrar un Usuario con correo erróneo

- **Identificarse con usuario incorrecto:** intentamos entrar al sistema con un usuario erróneo. Figuras D.5 y D.6.

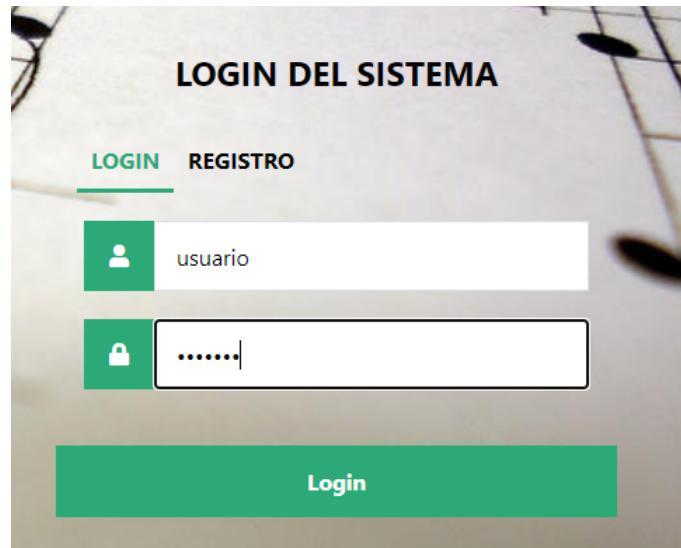


Figura D.5: Identificarse con usuario incorrecto

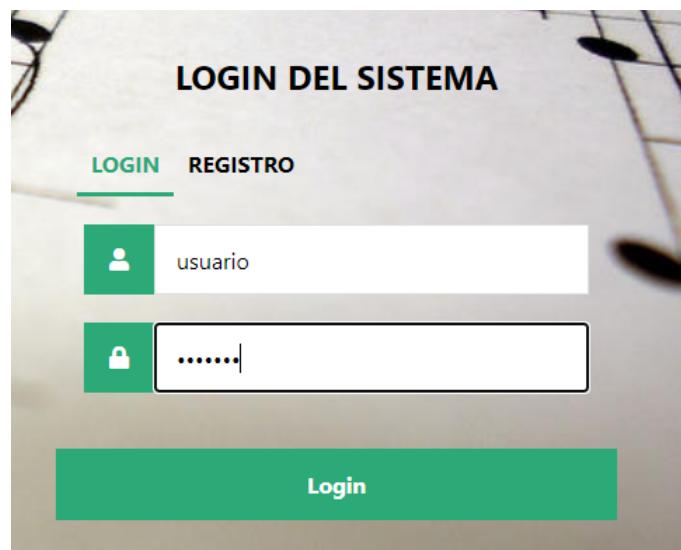


Figura D.6: Salida identificarse con usuario incorrecto

- **Identificarse con usuario correcto:** intentamos entrar al sistema con un usuario correcto. Figuras D.7 y D.8.

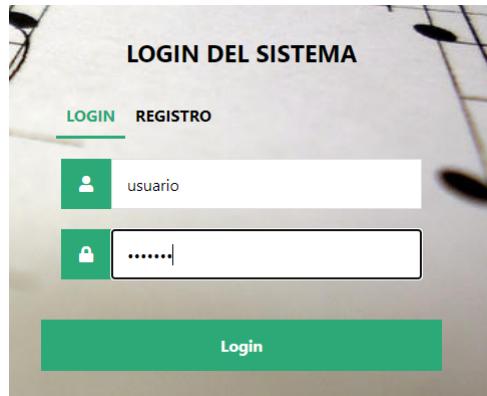


Figura D.7: Identificarse con usuario correcto



Figura D.8: Inicio del Sistema

- **Querer Procesar mientras hay otro fichero procesándose:** intentamos enviar un fichero a procesar mientras hay otro fichero procesándose. Figura D.9



Figura D.9: Mensaje de No se puede procesar

Apéndice E

Documentación de usuario

E.1. Introducción

En este apéndice se va a mostrar todo lo necesario para que cualquier usuario pueda ejecutar el sistema.

E.2. Requisitos de usuarios

Los requisitos necesarios ejecutar el sistema son:

- Tener un buscador Web.
- Necesidad que el sistema esté desplegado en una máquina¹.

E.3. Manual del usuario

A continuación, se muestra el funcionamiento de la Aplicación Web:

Inicio del sistema

La primera pantalla es la de inicio del sistema, en donde podemos identificarnos o registrarse en el sistema. En las figuras E.1 y E.2 se muestran ambas pantallas.

¹ Debido a problemas de versiones con los servidores de la universidad, es necesario desplegar el sistema en la máquina local, las instrucciones para esto se encuentra en D.4.



Figura E.1: Pantalla Login del sistema



Figura E.2: Pantalla Registro del sistema

Para registrarse, se debe añadir el Nombre, Usuario (que no puede existir), Contraseña y Correo electrónico (que debe tener dicho formato). Una vez que te registres, ya puedes acceder al sistema.

Pantalla Principal

En la figura E.3 se muestra la pantalla principal del sistema, en donde se muestra una barra con las distintas funcionalidades del sistema.



Figura E.3: Inicio del Sistema

Cargar Archivo

En la opción de la barra Cargar Archivo (mientras no exista una canción procesándose figura E.4) permite al usuario cargar una canción al sistema y enviarla a procesar. En la figura E.5 se muestra dicha pantalla.



Figura E.4: Mensaje de No se puede procesar



Figura E.5: Subir Canción para Procesar

Biblioteca

En la opción de la Biblioteca, se muestra la lista de canciones del sistema, si la canción no está procesada (en la columna de la tabla Procesada pone el indicador de procesado) el botón de descarga esta deshabilitado [E.6](#).

The screenshot shows the library section of the system. At the top, it says "SISTEMA DE RECONOCIMIENTOS DE MELODIAS". Below that is a navigation bar with "INICIO", "BIBLIOTECA" (which is highlighted), "CARGAR ARCHIVO", and "CERRAR SESIÓN". The main area is titled "BIBLIOTECA DEL USUARIO" and contains a table. The table has columns: ID, TÍTULO, PROCESADO, ESTILO, DESCARGAR ORIGINAL, and DESCARGAR PROCESADO. A single row is shown: ID 29, TÍTULO albinoni, PROCESADO NO, ESTILO CLÁSICO, DESCARGAR ORIGINAL (disabled), and DESCARGAR PROCESADO (enabled).

Figura E.6: Biblioteca canción sin procesar

En el caso de que la canción este procesada, se le permite descargarla como se ilustra en la figura [E.7](#).

The screenshot shows the 'BIBLIOTECA DEL USUARIO' (User Library) section of the system. At the top, there's a header with the University of Burgos logo and the text 'SISTEMA DE RECONOCIMIENTOS DE MELODIAS'. Below the header is a navigation bar with links: 'INICIO', 'BIBLIOTECA', 'CARGAR ARCHIVO', and 'CERRAR SESIÓN'. The main content area is titled 'BIBLIOTECA DEL USUARIO'. It displays a table with one row of data:

ID	TÍTULO	PROCESADO	ESTILO	DESCARGAR ORIGINAL	DESCARGAR PROCESADO
29	albinoni	SI	CLASICO	DESCARGAR	DESCARGAR

Figura E.7: Biblioteca canción procesada

Este proceso se puede repetir las veces que quiera el usuario. Para salir del sistema, solo se le debe dar a la funcionalidad Cerrar Sesión y se vuelve a la pantalla de inicio (figura E.3).

Bibliografía

- [1] Eduardo Mora González. Propuesta tecnológica para apoyar el aprendizaje de un instrumento monódico mediante comparación de pasajes musicales. <https://github.com/Masetto97/TFG>, 2019.
- [2] Edward Hagen y Peter Hammerstein. Did neanderthals and other early humans sing? seeking the biological roots of music in the territorial advertisements of primates, lions, hyenas, and wolves. *Musicæ Scientiae*, 2009.
- [3] Federico Simonetta; Carlos Cancino-Chacón; Stavros Ntalampiras y Gerhard Widmer. A convolutional approach to melody line identification in symbolic scores. <https://arxiv.org/pdf/1906.10547.pdf>, 2019.
- [4] Adolfo Nuñez. *Informática y Electrónica Musical*. ED. PARANINFO, 1993.
- [5] Gustavo Santos. *Inteligencia Artificial y Matemática Aplicada*. Universidad de Valladolid, 2002.
- [6] Grosvenor Cooper y Leonard B. Meyer. *Estructura rítmica de la música*. Idea Books, 2000.
- [7] musicnetmaterials. La textura musical. <https://musicnetmaterials.wordpress.com/2018/06/26/la-textura-musical>, 2018.
- [8] R.A.E. *Diccionario de la lengua española : Inteligencia Artificial*. 2019.
- [9] José Tomás Palma Méndez y Roque Marín Morales. *Inteligencia artificial : métodos, técnicas y aplicaciones*. Editorial McGraw Hill, 2008.

- [10] Germán García de Gurtubay. La inteligencia artificial simbólica. <https://www.atomian.com/pdf/La-IA-simbolica.pdf>, 2016.
- [11] Vicente Raja. *Conexionismo*. Sociedad Española de Psicología Analítica, 2019.
- [12] Juan Ignacio Bagnato. ¿cómo funcionan las convolutional neural networks? <https://www.aprendemachinelearning.com/como-funcionan-las-convolutional-neural-networks-vision-por-ordenador/>, 2018.
- [13] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. Biological Cybernetics. <http://www.cs.princeton.edu/courses/archive/spr08/cos598B/Readings/Fukushima1980.pdf>, 1980.
- [14] Wei Chai y Barry Vercoe. *Melody retrieval on the web*. In Martin G. Kienzle and Prashant J. Shenoy, 2020.
- [15] Elaine Chew y Xiaodan Wu. *Separating voices in polyphonic music: A contig mapping approach*. Uffe Kock Wil, 2004.
- [16] Patricio Letelier. Metodologías ágiles para el desarrollo de software: Extreme programming. <http://www.cyta.com.ar/ta0502/v5n2a1.htm>, 2005.
- [17] Sinnaps. Metodología scrum. <https://www.sinnaps.com/blog-gestion-proyectos/metodologia-scrum>, 2020.
- [18] Ken Schwaber y Jeff Sutherland. La guía definitiva de scrum: Las reglas del juego. <https://www.scrumguides.org/docs/scrumguide/v1/scrum-guide-es.pdf>, 2016.
- [19] Jeronimo Palacios. Scrum, la guía definitiva. <https://jeronomopalacios.com/scrum>, 2016.
- [20] Inc Red Hat. ¿qué es docker? <https://www.redhat.com/es/topics/containers/what-is-docker>, 2020.
- [21] Docker Inc. Overview of docker compose. <https://docs.docker.com/compose/>.
- [22] Flask Inc. Flask: Guia de usuario. <https://flask.palletsprojects.com/en/0.12.x/>.

- [23] Programador Click. Comprensión del protocolo wsgi. <https://programmerclick.com/article/39111685051/>, 2018.
- [24] Ana Soloaga. Principales usos de python. <https://www.akademus.es/blog/programacion/principales-usos-python/>, 2018.
- [25] Gaetan Hadjeres; Francois Pachet y Frank Nielsen. Deepbach: A steerable model for bach chorales generation. <https://arxiv.org/abs/1612.01010>, 2017.
- [26] Francois Pachet. Deepbach architecture. https://www.researchgate.net/figure/DeepBach-architecture_fig37_319524552.
- [27] Nadezhda Bessonova. La inteligencia artificial ahora puede imitar la música de bach. <https://nmas1.org/news/2016/12/28/inteligencia-bach#:~:text=El%20modelo%20DeepBach%20est%C3%A1%20fundamentado,las%20bibliotecas%20Keras%20y%20Tensorflow.,> 2016.
- [28] Christine McLeavey Payne. Musenet. <https://openai.com/blog/musenet/>, 2019.
- [29] Manuel Fernandez. Musenet: Esta web genera música hecha con i.a. con solo pulsar un botón. https://www.elespanol.com/omicrono/software/20190429/genera-musica-hecha-ia-solo-pulsar-boton/394711757_0.html, 2019.
- [30] Josep-Luis Arcos; Ramon López de Mántaras y Xavier Serra. Saxex : a case-based reasoning system for generating expressive musical performances. <https://www.iiia.csic.es/~arcos/old/ICMC.html>, 2018.
- [31] Josep Lluís Arcos. A case-based reasoning system for generating expressive performances. <https://www.iiia.csic.es/~arcos/projects/music/Saxex.html>, 1997.
- [32] Universidad de Burgos. Máster universitario en ingeniería informática: Objetivos y competencias. <https://www.ubu.es/master-universitario-en-ingineria-informatica/informacion-basica-i/objetivos-y-competencias>, 2020.