UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Uso de algoritmos de *machine learning* en la clasificación de géneros musicales de pistas de audio.

Trabajo de graduación presentado por Saúl Efrain Contreras Godoy
para optar al grado académico de Licenciado en Ingeniería en Ciencias
de la Computación y Tecnologías de la Información

Guatemala,

2022

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

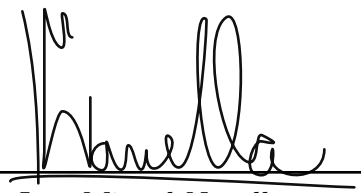
Uso de algoritmos de *machine learning* en la clasificación de géneros musicales de pistas de audio.

Trabajo de graduación presentado por Saúl Efrain Contreras Godoy
para optar al grado académico de Licenciado en Ingeniería en Ciencias
de la Computación y Tecnologías de la Información

Guatemala,

2022

Vo.Bo.:

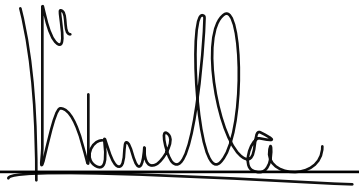
(f) 

Ing. Miguel Novella

Tribunal Examinador:

(f) 

Ing. Douglas Barrios

(f) 

Ing. Miguel Novella

(f) 

Ing. Lynette Garcia

Fecha de aprobación: Guatemala, 13 de Diciembre de 2022.

Índice

Lista de figuras

Lista de cuadros

Resumen

La clasificación de géneros musicales automatizada es un problema que requiere el uso de inteligencia artificial. La solución de esta problemática será útil para el emparejamiento de roles en la industria musical, a través de los géneros musicales correspondientes. Las redes neuronales son consideradas como la mejor solución para poder afrontar este problema. Cuatro tipos de redes neuronales son propuestas como solución a esta problemática: Una red neuronal *feed-forward* que obtuvo un 26.12 % de efectividad, una Red convolucional neuronal *Le-Net* con un 72.30 % de efectividad, una red convolucional neuronal con una capa *LSTM* que tiene una efectividad del 74.67 %, una red convolucional neuronal con *MaxPooling* que obtuvo la mayor efectividad, siendo 77.04 %. Se construyó una plataforma en donde se pueden encontrar todos los modelos construidos en esta investigación.

Abstract

Automated music genre classification is a problem that requires the use of artificial intelligence. The solution of this problem will be useful for the matching of roles in the music industry, through the corresponding musical genres. Neural networks are considered the best solution to deal with this problem. Four types of neural networks are proposed as solution in this problem: A feed-forward neural network that obtained 26.12 % effectiveness, a Le-Net convolutional neural network with 72.30 % effectiveness, a convolutional neural network with an LSTM layer which has an effectiveness of 74.67 %, a convolutional neural network with MaxPooling that obtained the highest effectiveness, being 77.04 %. A platform was built where all the models built in this research can be found.

CAPÍTULO 1

Introducción

En el proyecto de graduación se podrá observar tres fases específicas. La extracción y análisis de datos proveídos por distintas *APIs* de servicios de *Streaming* en donde se encuentran los datos de las canciones disponibles en sus librerías. La construcción de algoritmos de *Machine Learning* para clasificar cada uno de los géneros musicales. La elaboración de un sitio web donde se pueda obtener estos modelos. En la fase de extracción y procesamiento de datos se plantea extraer la mayor cantidad de datos posibles disponibles en *SoundCloud API* (SoundCloud Developers, 2022) y servidores similares. Estos datos deben de ser procesados de tal manera que todos tengan la misma entrada y estructura. En la fase de elaboración de algoritmos de *Machine Learning*, se propone determinar las tecnologías favorables para el experimento y determinar con base en la literatura los algoritmos adecuados para el análisis, tales como redes neuronales, *KNN* o *random forest*, buscando los lenguajes de programación que tienen los paquetes más favorables para su realización. Al tener claro estas tecnologías se propone proceder a realizar los algoritmos de clasificación con sus respectivas métricas. Por último se plantea la elaboración de un sitio web que permita el acceso a los modelos: Se pretende elaborar un sitio en el que los creadores de música puedan acceder a los modelos elaborados para poder facilitar la identificación de género de nuevas pistas de audio.

CAPÍTULO 2

Justificación

En Guatemala existen pocos equipos trabajando en la investigación tecnológica dentro de la industria musical. Además existe una amplia comunidad dentro de la misma industria, que comprende productores, compositores, letristas, representantes, inversionistas, managers, agentes, instrumentistas, curadores e ingenieros de audio que buscan crear un impacto por medio de la creación de música dentro y fuera del país.

Según Richard James Burgess, en su libro *The Art of Music Production: The Theory and Practice* publicado por la Universidad de Oxford. Es necesario un sistema de clasificación para poder enlazar a los productores con los artistas y viceversa. Esto se debe a que los trabajos y proyectos de cada uno de los roles mencionados se sesgan de acuerdo al género musical de las piezas. Es decir que a lo largo de su carrera, los proyectos de un productor o artista serán de un género muy similar. En el mismo libro Burgess menciona que la clasificación por género es uno de los mejores sistemas para emparejar dichos roles. (Burgess, 2013).

Partiendo de la necesidad de automatizar y mejorar los procesos de identificación de género, de tal manera que se pueda crear una nueva pieza musical a través de algoritmos de inteligencia artificial. Y que dichos modelos faciliten el trabajo a los analistas de datos, curadores y representantes (conocidos como A&R). Es necesario que se utilice *machine learning* para poder investigar y clasificar los datos disponibles en los servicios de *streaming*, ya que de esta manera se podrá optimizar el proceso de emparejamiento entre los distintos roles dentro de la industria musical, para que así se pueda generar más ingresos y aumentar la cantidad de piezas musicales, en especial en países latinoamericanos.

A partir de trabajos anteriores se propone una conversión de los archivos de audio (mp3, wav, etc.) en un tipo de archivo relativamente sencillo de leer para ser procesado dentro de los algoritmos de *machine learning*. Pelchat y Gelowitz proponen aplicar filtros en las frecuencias altas (*high pass*) o frecuencias bajas (*low pass*) para poder procesar los datos sin efectos, lo que resulta ser un acercamiento sencillo pero que facilitará el trabajo al procesar los datos. (Pelchat y Gelowitz, 2020).

Tener datos básicos confiables es un requisito esencial para formar clasificadores de género efectivos. Se ha sugerido que sólo se puede llegar a un acuerdo limitado entre los exegetas al clasificar la música por género y que tales restricciones colocan un techo inevitable en el desempeño de la clasificación por género. Las personas pueden diferir no sólo en cómo clasifican una determinada grabación, sino también en el conjunto de etiquetas de género que eligen. Pocos géneros tienen una definición clara y la información disponible suele ser poco clara e inconsistente de una fuente a otra. A menudo hay una superposición considerable entre los géneros, y las grabaciones individuales pueden ser de diferentes géneros en diversos grados. (Xu et al., 2003).

Por último es necesario mencionar la importancia de las ciencias de la computación como una disciplina integral que puede brindar soluciones innovadoras a problemas de otras disciplinas de aprendizaje, debe existir un profundo interés en el estudio e investigación dentro de la industria musical. Con esta investigación se pretende utilizar las herramientas de *machine learning* para ayudar en la identificación automatizada de géneros musicales de nuevas pistas de audio, dado que, según la revista *Music Business Worldwide* , más de 60,0000 canciones son subidas a la plataforma conocida como *Spotify* en un solo día. Esto significa que cada 1.4 segundos una nueva canción es subida a la plataforma, sin tener su mayor alcance posible, debido a un mal sistema de emparejamiento. (Ingham, 2021).

CAPÍTULO 3

Objetivos

3.1. Objetivo general

Analizar métodos de clasificación de *machine learning* para clasificar géneros musicales de pistas de audio con una efectividad mayor al 75 %.

3.2. Objetivos específicos

- Obtener datos de las *APIs* públicas de servicios de *Streaming*.
- Construir modelos de *machine learning* que permitan clasificar los géneros musicales de nuevas pistas de audio.
- Construir un sitio web de acceso público que permita determinar el género musical de una nueva pista de audio.

CAPÍTULO 4

Marco teórico

En la exploración de la identificación de géneros musicales y los algoritmos de *machine learning*, utilizados para poder clasificar y posteriormente detectar cada uno de los mismos. Se ha dado la tarea de revisar la literatura existente a través de bases de datos, buscando antecedentes de estudios, investigaciones, proyectos o artículos similares o relacionados, que permitan una base sólida para el desarrollo del presente proyecto de investigación, permitiendo la generación de conocimiento. Buscando una definición del problema que sea sensata, ajustada a la realidad y plausible de desarrollar.

El primero de los documentos investigados, para obtener un estado del arte completo es el trabajo de Tzanetakis y Cook. Dentro del *set* de datos que presenta el trabajo, los colaboradores decidieron presentar distintas características de la música que se esta evaluando. Las características que posee el modelo final son: Textura, nota y *beat*. La aproximación que utiliza el modelo esta basada en el concepto de *K-nearest neightbor*. La base de datos proveída contiene 1000 datos de distintos géneros presentando los primeros 30 segundos de cada canción. En el trabajo presentado se obtiene un 60 % de efectividad. (Tzanetakis y Cook, 2002)

Un acercamiento basado en *Support Vector Machine* (SVM) y *Linear Discriminant Analysis* (LDA) fue realizado por Li Ogihara y Li. En este trabajo presentan dos bases de datos distintas. La primera base de datos es GTZAN, que presenta 1000 canciones canciones con los primeros 30 segundos de cada canción. La segunda base de datos conocida como *MARSYAS*. Esta base de datos contiene 755 canciones de 5 géneros distintos. Este trabajo evidencia que con la primera base de datos se puede clasificar géneros musicales con un 72 % de efectividad y un 78 % con la segunda base de datos. (Li et al., 2003)

4.1. *Machine Learning*

En esta investigación se da la tarea de investigar sobre los métodos de clasificación de géneros musicales, que han logrado una efectividad mayor al 75 %. Estos métodos han sido

desarrollados con la estadística (Regresión lineal y análisis discriminante) o bien por inteligencia artificial (Redes neuronales, inducción de reglas, árboles de decisión, redes bayesianas, entre otras). (AprendeMachineLearning.com, 2022) A continuación se hablará sobre dichos métodos:

4.1.1. Algoritmos de regresión

Los algoritmos de Regresión "modelan la relación entre distintas variables (*features*) utilizando una medida de error que se intentará minimizar en un proceso iterativo para poder realizar predicciones lo más acertadas posible". (AprendeMachineLearning.com, 2022). Gujarati, define el análisis de regresión como: "El estudio de la dependencia de la variable dependiente, sobre una o más variables explicativas, con el objeto de estimar o predecir el valor promedio poblacional de la primera en términos de los valores conocidos o fijos (en medias muéstrales repetidas) de las últimas" (Parra, 2006)

La técnica de la regresión lineal permite determinar relaciones de dependencia de tipo lineal entre una variable dependiente o endógena, respecto de una variable explicativa o exógena.

4.1.2. Algoritmos Bayesianos

El teorema de Bayes, de acuerdo con Lind (Marchal et al., 2014), describe que en el siglo XVIII, el reverendo Thomas Bayes, un ministro presbiteriano inglés, planteó esta pregunta: ¿Dios realmente existe? Dado su interés en las matemáticas, intentó crear una fórmula para llegar a la probabilidad de que Dios existiera sobre la base de la evidencia de que disponía en la Tierra. Más tarde Pierre-Somón Laplace perfeccionó el trabajo de Bayes y le dio el teorema de Bayes. De una forma entendible, el teorema de Bayes es el siguiente:

$$P(A!|B) = P(A!)P(B|A!)P(A1)P(B|A1) + P(A2)P(B|A2)$$

En este teorema, los eventos A1 y A2 son mutuamente excluyentes y colectivamente exhaustivos, y A! se refiere al evento A1 o a A2. De ahí que en este caso A1 y A2 sean complementos. Para encontrarle significado a los símbolos, Lind (Marchal et al., 2014) nos proporciona el ejemplo siguiente: Suponga que 5 % de la población de un país X tiene una enfermedad. Sea A1 el evento "padece enfermedad" y A2 el evento "no padece enfermedad". Por tanto, al seleccionar al azar una persona del país X, la probabilidad que la persona elegido padezca la enfermedad es del 0.05 o P(A1)=0.05. Esta probabilidad P(A1)=P(padece enfermedad)=0.05 recibe el nombre de probabilidad a priori. Se le da este nombre, porque la probabilidad se asigna antes de obtener los datos empíricos. Por ende, la probabilidad a priori de que una persona no padezca enfermedad es de 0.95, o P(A2)=0.95, que se calcula restando 1-0.05

4.1.3. Algoritmos de *clustering*

El análisis clustering (AC) dentro del programa estructurado, siguiente a Parra (Parra, 2006), consiste en una agrupación de técnicas multivariantes de objetos basados en las características que poseen; este algoritmo de *clustering* clasifica los objetos en clases o conglomerados de tal forma que cada objeto sea parecido a los que hay en el conjunto de su conglomerado. Los conglomerados resultantes deben de tener un alto grado de homogeneidad interna y heterogeneidad externa. Los resultados que se esperan, entre otros están, (a) elaboración de una tipología o clasificación, (b) investigación de esquemas conceptuales útiles para agrupar sujetos, (c) generación de hipótesis por medio de exploración de datos y (d) comprobar si las hipótesis generadas por medio de otros procedimientos se cumplen en la muestra de datos. Las etapas, en términos generales, consisten en: selección de la muestra que se pretende dividir en grupos, selección de las variables que se van a utilizar para realizar el análisis, cálculo de las similitudes entre los casos o sujetos y validación de los resultados obtenidos.

4.1.4. Algoritmos de árboles de decisión

Los árboles de decisión o clasificación es un modelo algorítmico para clasificar particiones sucesivas. (Parra, 2006). Es la representación de todos los cursos de acción y resultados consecuentes posibles. (Marchal et al., 2014). Son apropiados cuando hay un número elevado de datos, siendo una de sus ventajas su carácter descriptivo que permite entender e interpretar fácilmente las decisiones tomadas por el modelo, revelando formas complejas en la estructura de datos que no se pueden detectar con los métodos convencionales de regresión. Los algoritmos que se encuentran, o bien solos o bien integrados en diferentes paquetes informáticos, son los que determinan o generan el procedimiento de cálculo que establece el orden de importancia de las variables en cada interacción. También se pueden imponer ciertas limitaciones en el número de ramas en que se divide cada nodo. Los elementos que caracterizan los algoritmos de clasificación para su construcción, son los siguientes: (a) El criterio de la partición de cada nodo; (b) la regla que declara un nodo terminal: (c) la asignación de una clase a cada nodo terminal, lo que determina la regla de clasificación; (d) determinar la fusión que consiste en relación a la variable dependiente, las categorías de las variables predictoras no significativas se agrupan juntas para formar categorías combinadas que sean significativas; (e) podar las ramas que añaden poco valor de predicción al árbol y (f) la evaluación de la bondad del clasificador obtenido, es decir, la estimación de la validación del árbol y el cálculo del riesgo. (Marchal et al., 2014)

4.1.5. Redes neuronales

En la figura, se muestra un modelo matemático propuesto por McCulloch and Pitts en 1943. Se puede observar como la neurona posee varias entradas, en palabras formales una combinación lineal de entradas que excede un limite propuesto. Este modelo describe a una red neuronal como una colección de unidades conectadas, las propiedades de la red neuronal esta determinada por la topología de las propiedades de las neuronas. (Michie et al., 1994)

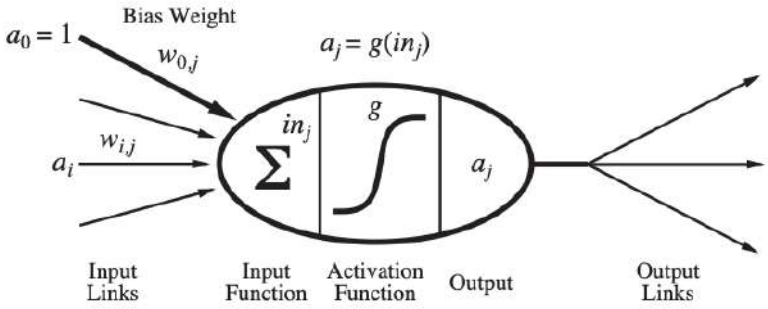


Figura 1: Un modelo matemático sencillo para una neurona artificial.

Desde 1943 se han construidos muchos modelos, y se han desarrollados teorías basados en los mismos. El estudio de estas neuronas y los sistemas grandes construidos a partir de los mismos dieron entrada al cambio de la neurociencia computacional. (Russell y Norvig, 2016).

Por otro lado los investigadores en el cambio de la inteligencia artificial y la estadística se han interesado más en las propiedades abstractas de las redes neuronales, tales como la habilidad de ejecutar el concepto de computación distribuida y tolerar entradas llenas de ruido e incluso aprender. El estudio de las propiedades de las redes neuronales es aún de interés, en la actualidad. (Russell y Norvig, 2016).

Las redes neuronales están compuestas por nodos o unidades conectadas en enlaces. Un enlace conecta a la unidad i con la unidad j para poder propagar la activación a_i. Este enlace también tiene un peso numérico identificado como $w_{i,j}$. Cada unidad tiene una entrada retardada $a_0 = 1$ que esta asociada el peso $w_{0,j}$. (Russell y Norvig, 2016)

En el presente estudio el concepto de redes neuronales será mucho más útiles que los anteriores mencionados, ya que será algo básico para poder aceptar entradas llenas de ruido, dado que las canciones de un mismo género poseen distintos componentes que pueden representar ruido en la clasificación de las canciones. Además permitirá que las redes neuronales implementadas puedan aprender a lo largo de las distintas épocas de entrenamiento.

4.1.6. Redes convolucionales

Las redes convolucionales o también conocidas como redes neuronales convolucionales o CNNs son un tipo especial de red neuronal para procesar datos que son conocidos como topologías tipo grilla. Un ejemplo claro de los datos tipo grilla de una dimensión son los datos tomados con intervalos de tiempo, como recibidos a través de un sensor. Un claro ejemplo de una topología tipo grilla de dos dimensiones sería una imagen, ya que posee una matriz de $m * n$, siendo m la altura y n la longitud de la imagen. Las redes convolucionales han sido exitosas en las aplicaciones prácticas. El nombre convolucional hace referencia a la operación matemática convolución. Convolución es una operación lineal. Las redes convolucionales son redes neuronales que usan una convolución en lugar de la matriz general de multiplicación en, por lo menos, una de sus capas.(LeCun et al., 1989)

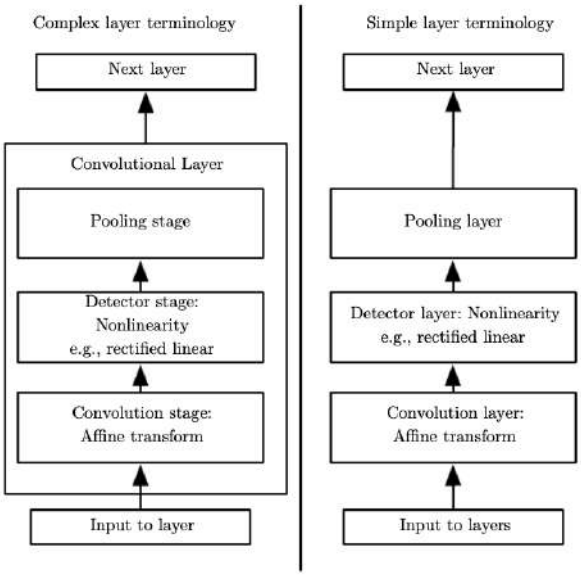


Figura 2: Estructura típica de una red convolucional neuronal simple y compleja.

La operación de convolución

La convolución como fue mencionado con anterioridad es una operación lineal. Es una operación de dos funciones de un argumento. Matemáticamente una convolución está definida como:

$$s(t) = \int_{-\infty}^{\infty} x(a)\omega(t-a)\,da$$

Generalmente esta operación se denota:

$$s(t) = (x * \omega)(t)$$

Cuando se habla en la terminología de redes convolucionales el primer argumento (denotado como la función x en la ecuación) es llamado la entrada y el segundo argumento (denotado como ω) como el kernel. La salida es referida como el mapa de características o *feature map*.

Debido a que en operaciones computacionales se debe de utilizar operaciones discretas es imposible ejecutar una integral (que representa una operación continua). Es necesario utilizar un método numérico para poder ejecutar la integral. Dejando la operación de la siguiente manera:

$$s(t) = (x * \omega)(t) = \sum_{-\infty}^{\infty} x(a)\omega(t-a)$$

En redes neuronales, la entrada es usualmente es un arreglo multidimensional de datos y el kernel correspondiente es un arreglo multidimensional de parámetros que se adaptan por el algoritmo de aprendizaje. Estos arreglos multidimensionales son llamados tensores, ya que cada uno de estos elementos son guardados y/o alocados en la memoria por separado.(LeCun et al., 1989)

Parameter Sharing

Parameter sharing se refiere a utilizar el mismo parámetro en más de una de las funciones dentro de un modelo. En una red tradicional, el peso de cada elemento es utilizado usualmente una vez cuando se ejecuta la ultima capa del modelo. Se multiplica por un elemento de la entrada y luego nunca se vuelve a visitar. Como sinónimo de *parameter sharing*, se puede decir que una red acelera los pesos, porque el valor del peso aplicado a una entrada está ligado al valor de un peso aplicado en otro lugar.(LeCun et al., 1989)

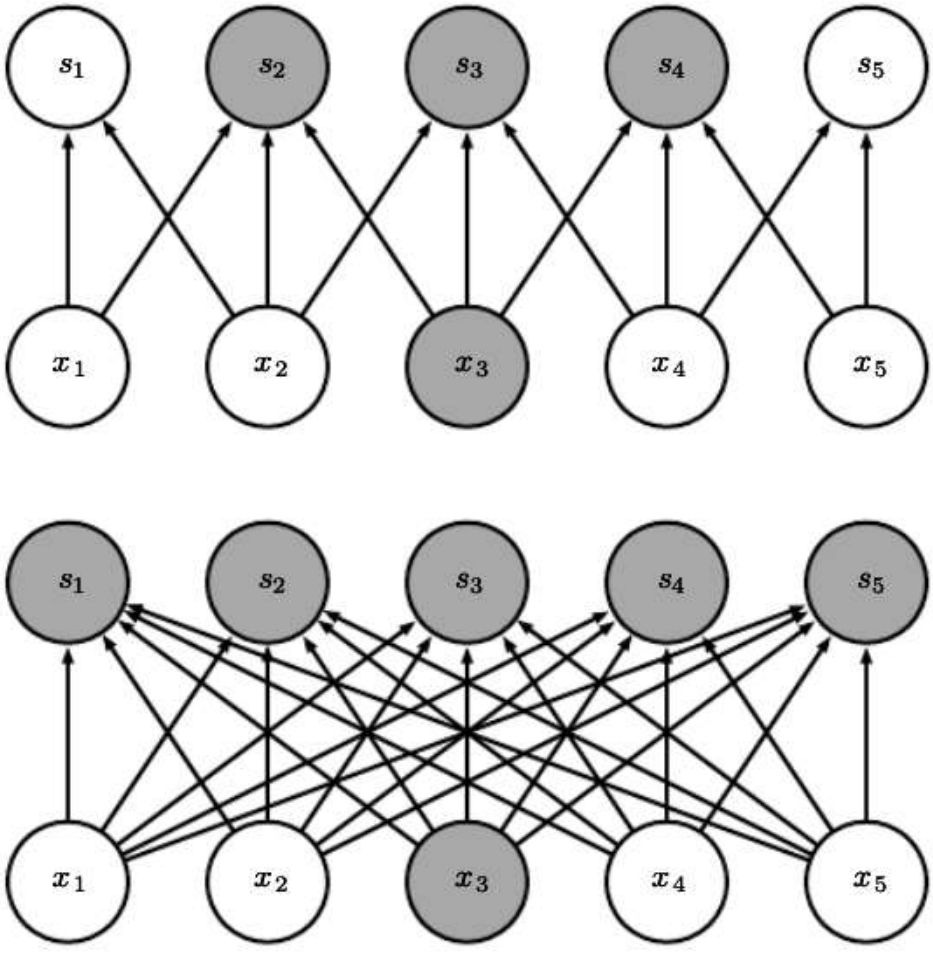


Figura 3: Escasa conectividad, vista desde abajo. (Arriba) Cuando se forma por convolución con un *kernel* de ancho 3, x solo afecta tres salidas. (Abajo) Cuando se forma por multiplicación de matrices, la conectividad ya no es escasa, por lo que todas las salidas se ven afectadas por x_3.

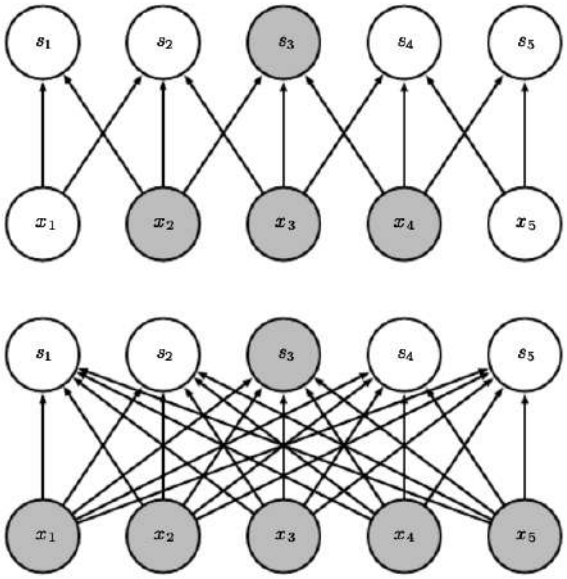


Figura 4: Escasa conectividad, vista desde arriba. Resaltamos una unidad de salida, s_3, y (Arriba) Cuando se forma por convolución con un kernel de ancho 3, solo tres entradas afectan a s_3. (Abajo) Cuando s se forma mediante la multiplicación de matrices, la conectividad ya no es escasa, por lo que todas las entradas afectan a s_3.

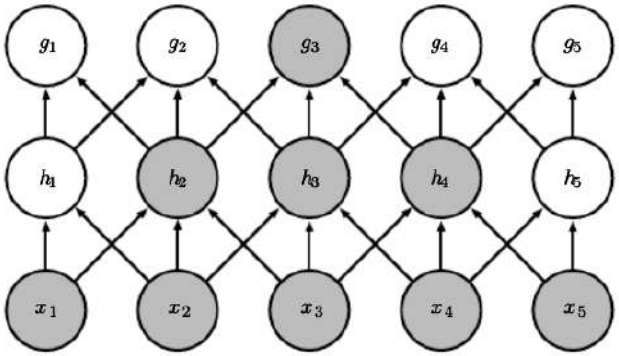


Figura 5: El campo receptivo de las unidades en las capas más profundas de una red convolucional es mayor que el campo receptivo de las unidades en las capas superficiales. Este efecto aumenta si la red incluye características arquitectónicas como convolución estriada (figura 9.12) o pooling

Pooling

Una capa típica de la red convolucional es conocida como *Pooling* que consiste de tres etapas. En la primera etapa, la capa ejecuta varias convoluciones en paralelo para producir un *set* de activaciones lineales. En la segunda etapa cada activación lineas es ejecutada a través de una función de activación no lineal a esta etapa se le conoce como etapa de detección. Por último en la tercera etapa se utiliza la función pooling para modificar la salida de la siguiente capa. (Hochreiter y Schmidhuber, 1997)

4.1.7. Modelo *Le-Net*

El modelo *Le-Net* es uno de los modelos pre-entrenados propuestos por Yann LeCun y otros autores en el año 1998. Los autores utilizaron la arquitectura para poder diferenciar imágenes de escritura a mano y caracteres impresos a máquina. La forma en la que esta arquitectura estaba diseñada fue descrita como sencilla y simple, adjetivos que fueron la razón de su popularidad. Es una red convolucional multicapa para la clasificación de imágenes. (LeCun, Bottou, et al., 1998)

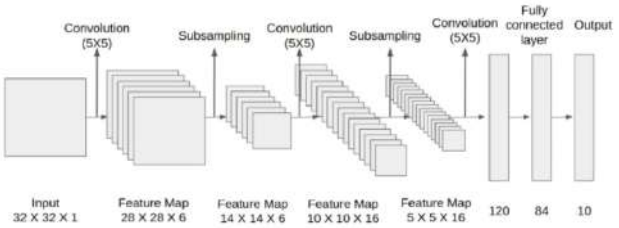


Figura 6: Estrucutra típica de *Le-Net*

4.1.8. Modelo *Feed Forward*

Una red *feed-forward* tiene conexiones sólo en una dirección, dirigidos en un grafo acíclico. Cada nodo recibe una entrada desde los nodos superiores y da la salida a los nodos inferiores. Las redes *feed-forward* son usualmente organizadas en capas, de tal forma que cada unidad reciba la entrada inmediatamente de la capa predecesora. Cada entrada y sus salidas posee una capa oculta que se conecta con las salidas de la red. (Russell y Norvig, 2016).

4.1.9. Perceptrón

Un perceptrón es también llamada red perceptrón o *single-layer neural network*. Los perceptrones son extremadamente inutiles sin embargo son realmente complejas en comparación con las operaciones booleanas. Como se puede observar en la siguiente figura al compararlo con un árbol de decisión el perceptrón es complejo en comparación con el árbol, sin embargo su asertividad es similar. (Russell y Norvig, 2016).

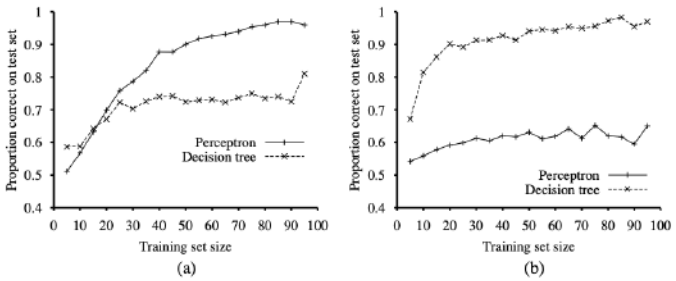


Figura 7: Comparación de efectividad entre perceptrones y arboles de deción. Los perceptrones son mejores aprendiendo con funciones de más de 11 entradas

4.1.10. Modelo *LSTM*

A diferencia de las redes neuronales *feed-forward*, las redes neuronales *LSTM* tienen conexiones de retroalimentación que pueden ser redes neuronales recurrentes que pueden procesar más que solo imágenes. *LSTM* hace referencia a *Long Short Term Memory*. Se han utilizado en reconocimiento de escritura a mano, reconocimiento de voz, traducción por máquina, entre otros. (Hochreiter y Schmidhuber, 1997)

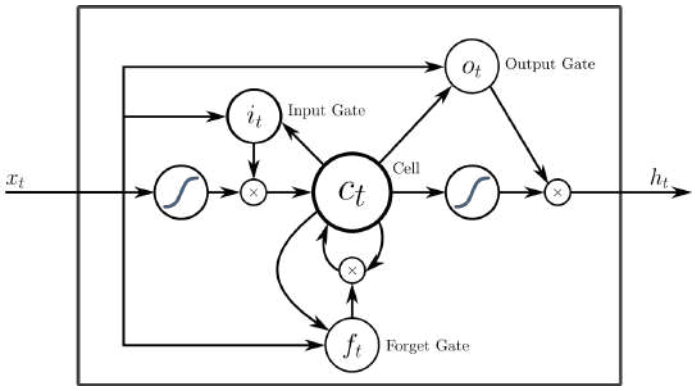


Figura 8: Estructura de un *LSTM peep-hole*

4.2. Teoría musical

La música es un tema amplio que cubre distintos conocimientos tanto físicos como artísticos. A continuación tocaremos los temas esenciales para comprender la teoría musical:

Según Glenn Elert la música y el ruido son ambas mezclas de ondas de sonidos de diferentes frecuencias. Las frecuencias que componen a la música se pueden identificar como discretas (o separables) y racionales (los radios forman fracciones simples). Mientras el ruido está compuesto de frecuencias aleatorias en donde es difícil identificar las frecuencia dominante. (Elert, 2021)

4.2.1. Sonido

El sonido es una onda longitudinal, lo que significa que las partículas del medio vibran paralelamente con la dirección a la que se propaga la onda. El oído humano puede detectar las ondas de sonido debido a que estas se propagan a lo largo del aire, provocando que las partículas del mismo se dilaten y se condensen, modificando la presión del medio. De esta manera el tímpano puede detectarlo. Las ondas de sonido también pueden ser detectadas por el diafragma de un micrófono y ser transformadas a pulsos electromagnéticos, es decir que una computadora puede procesarlas. (Elert, 2021)

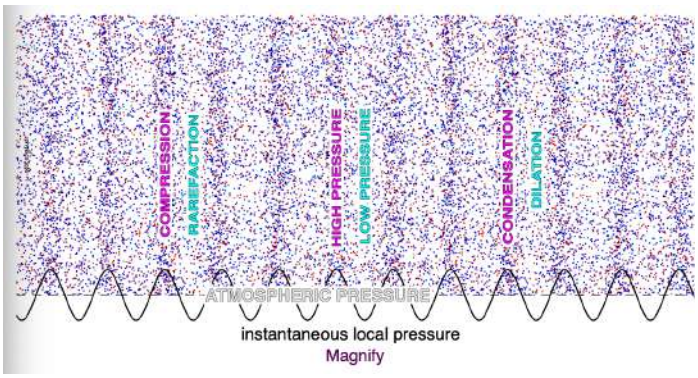


Figura 9: Representación de una onda longitudinal propagándose en el aire

La manera más sencilla de estudiar el sonido es simplicandolo a un análisis matemático. En donde la onda longitudinal esta describida por una única frecuencia (esto también es conocida como tono puro). Serway define la ecuación de una onda longitudinal de la siguiente manera:

$$Y(t) = Asin(2\pi ft)$$

En esta expresión A representa la amplitud del sonido, es decir la intensidad con la que la frecuencia es tocada, f representa la frecuencia que se esta tocando o vibrando y t representa el tiempo. Se puede ver cómo el sonido depende de la frecuencia a la que esta sonando, la amplitud con la que esta suena y que también es distinto a través del tiempo. (Serway y Jewett, 2013)

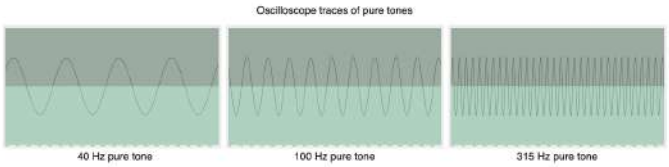


Figura 10: Tonos puros para 40 hz, 100hz y 315 hz (de izquierda a derecha).

4.2.2. Nota musical

Físicamente las notas musicales son producidas, como cualquier sonido, por un cuerpo vibrando en una frecuencia específica. Cada frecuencia está asociada a una nota musical distinta. En la música occidental se trabaja en un sistema modular 12. Es decir existen 12 frecuencias identificables que se repiten a medida que se aumenta la frecuencia. Al conjunto de estas 12 frecuencias se le conoce como octava. Para muchas fuentes, en particular los instrumentos musicales, la onda de presión del sonido es aproximadamente periódica durante periodos breves. Es decir, la onda de presión se repite con el tiempo. La frecuencia fundamental es el número de ciclos de la forma de onda por segundo. Los tonos periódicos son producidos por cuerpos que vibran, como cuerdas en violines y pianos, y columnas resonantes de aire en fagotes y tubos de órgano. Todos los tonos periódicos (excepto las ondas sinusoidales) contienen una serie de componentes de frecuencia, llamados armónicos. (Krumhansl, 2000). A continuación se muestra la tabla de frecuencias para la octava cero

como es conocida en la notación clásica. Se puede observar como en la octava "1", en la notación anglosajona, el rango de frecuencias es de 16.35 Hz a 30.87 Hz.

Notación clásica	Notación anglosajona	Frecuencia (Hz)
Do_0	C_0	16.3516
$Do_0^{\#}$ - Re_0^b	$C_0^{\#}$ - D_0^b	17,3239
Re_0	D_0	18.35441
$Re_0^{\#}$ - Mi_0^b	$D_0^{\#}$ - E_0^b	19,4454
Mi_0	E_0	20.6017
Fa_0	F_0	21.8268
$Fa_0^{\#}$ - Sol_0^b	$F_0^{\#}$ - G_0^b	23,1247
Sol_0	G_0	24.4947
$Sol_0^{\#}$ - La_0^b	$G_0^{\#}$ - A_0^b	25,9565
La_0	A_0	27.5000
$La_0^{\#}$ - Si_0^b	$A_0^{\#}$ - B_0^b	29,1353
Si_0	B_0	30.8677

Cuadro 1: Frecuencia de notas de la octava cero

(ParaBajoelectrico.com, 2022)

4.2.3. Ritmo

El ritmo, según Krumhansl, es la capacidad directa que tiene el ser humano de procesar información temporal. En la música el ritmo representa la manera en la que las distintas notas o figuras musicales son separadas o posicionadas en un espacio de tiempo. Según Krumhansl, la percepción humana de estos patrones se determina por la duración de las figuras musicales. Distintos experimentos han demostrado que la mente humana tiende a agrupar patrones para poder diferenciar ritmos y cuando varias figuras musicales están juntas temporalmente la mente tiende a agruparlos como una sola figura. El ritmo está asociado con el tiempo musical, cada canción posee una notación que indica cuantos golpes por minuto tiene la canción. Por lo general se utilizan los BPM (*beats per minute*), en la mayoría de ocasiones nos indica cuántas negras tendrá la canción en un solo minuto. (Krumhansl, 2000).

Nombre de la figura	Valor en tiempo real o pulsos
Redonda	4
Blanca	2
Negra	1
Corchea	1/2
Semicorchea	1/4
Fusa	1/8
Semifusa	1/16

Cuadro 2: Valor relativo de las notas en tiempos o pulsos

(Michels, 1985)

4.2.4. Combinación lineal en la música

Una sola nota musical representa una único tono puro, sin embargo en la música moderna se combinan distintas notas para formar acordes, en distintos momentos y con distintas intensidades. Y aún un instrumento tocando una sola nota, esta formado por una combinación lineal de distintas funciones de onda que representan la nota que se ejecuta. Esta combinación permite que el sonido que emite una guitarra sea distinguible del emitido por un piano, aún tocando la misma nota. En la siguiente gráfica se puede observar como las funciones de onda son muy distintas a una función sinusoidal, esto se debe a que en realidad son la suma de muchas funciones sinusoidales, concepto que se conoce como combinación lineal. (Elert, 2021)

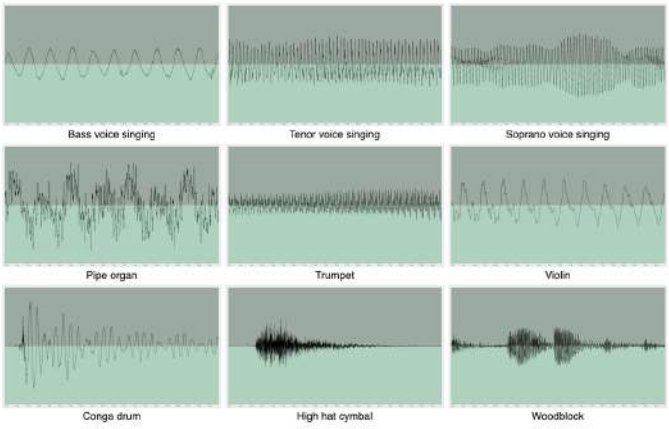


Figura 11: Funciones de onda para distintos instrumentos.

Como muchos otros sistemas mecánicos, los instrumentos musicales vibran en frecuencias relacionadas conocidas como armónicos. La frecuencia más baja se le conoce como la fundamental. El oído humano puede percibir las frecuencias fundamentales como la nota de afinación. Las amplitudes del resto de armónicos relativos dan la cualidad o el timbre. (Elert, 2021)

La permisividad de la combinación lineal o superposición en la música permite que se puedan crear valores de una función en un espacio matemático con cualquier combinación de funciones sinusoidales o cosenoidales que son usadas en la música moderna.

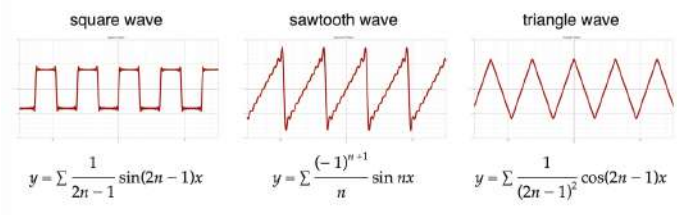


Figura 12: Ejemplo de combinaciones lineal de funciones sinusoidales y cosenoidales

4.2.5. Análisis de Fourier

Matemáticamente es posible transformar cualquier función de onda desde una secuencia continua o combinación lineal a una serie discreta de funciones sinusoidales y cosenoidales.

El proceso es conocido como análisis espectral o análisis de Fourier. El proceso fue nombrado así en honor a Joseph Fourier. Desde el punto de vista musical, esto significa que es posible convertir cualquier onda de sonido e identificar las frecuencias que crean la onda que se escucha. (Stewart et al., 2020)

La transformada de Fourier posee la siguiente forma:

$$\int_{-\infty}^{\infty} f(x)e^{-i2x\pi\epsilon}\,dx$$

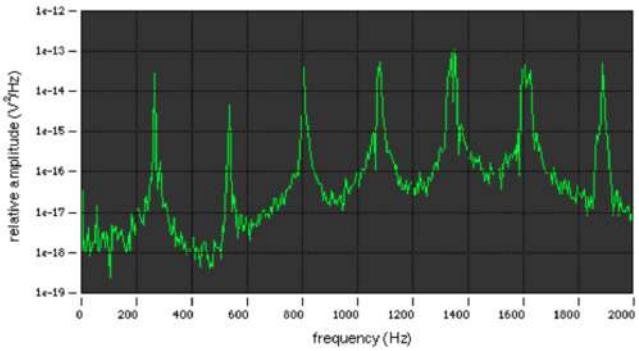


Figura 13: Ejemplo de aplicación de la transformada de Fourier, donde se observa las frecuencias contra las amplitudes

4.2.6. Síntesis de sonido

Según Jordá (Jorda, 1997) el término síntesis no es plenamente acertado. pues gran parte de los instrumentos digitales actuales no sintetizan totalmente el sonido (no parten de cero), sino que utilizan, recombinan y modifican fragmentos almacenados en su memoria. Es decir que existen componentes eléctricos que permiten que se pueda crear sonidos de una manera no mecánica. Existen diferentes métodos para sintetizar sonido:

- Síntesis aditiva: Se crea una onda resultante de ondas almacenadas en memoria. Esta onda resultante mantiene la frecuencia del componente más grave, pero con el timbre alterado. (Jorda, 1997)

- Modulación de Frecuencias (FM): En el caso más sencillo, la síntesis FM necesita tan sólo dos osciladores: la señal portadora y la señal moduladora. Parte de la idea de que cuando la moduladora no es una señal de baja frecuencia sino que entra ya en el rango de las frecuencias audibles (a partir de los 20 Hz) se crean un gran número de frecuencias adicionales que generan un sonido con un gran contenido armónico. (Jorda, 1997)

- Síntesis por tabla de ondas: Los avances tecnológicos de principios de los ochenta hicieron posible la sustitución de las ondas periódicas simples que se venían utilizando como material base, por pequeños fragmentos procedentes de sonidos reales, digitalizados y almacenados en ROM. (Jorda, 1997)

- «Sampler»: El «sampler»a diferencias de los métodos anteriores permite que el sonido sea guardado en RAM y no en ROM, permitiendo que el usuario pueda modificar dicho sonido o pulso electromagnético. Generando una muestra (o «sample») que utiliza como base otro sonido almacenado. (Jorda, 1997)

4.2.7. Género musical

La manera en la que categorizamos la música y cómo se teoriza depende mucho de cómo definimos qué es un género musical. Es decir que propiedades definimos y que barreras vamos a colocar para diferenciar unos géneros de otros.

Algunas de las definiciones que han dado para un género musical proponen que un género es una clase de eventos comunicativos, una manera en la que los miembros comparten con propósitos comunicativos. Desde este punto de vista un género esta relacionado directamente como un fenómeno social siendo totalmente dependiente de las expectativas y la manera en la que este se desarrolla a lo largo de la cultura. (Swales, 2004)

Siguiendo la definición anteriormente descrita las barreras de un género musical y otro son borrosas y difíciles de identificar. Sin embargo Mendoza hace mención a que un género musical no solo es dependiente de la cultura sino que de cuestiones cognitivas y de taxonomías creadas por expertos. Según su definición un género musical depende del ciclo de tres fases, ciclo que permite que se desarrolle la transformación y reconfiguración de los géneros musicales.

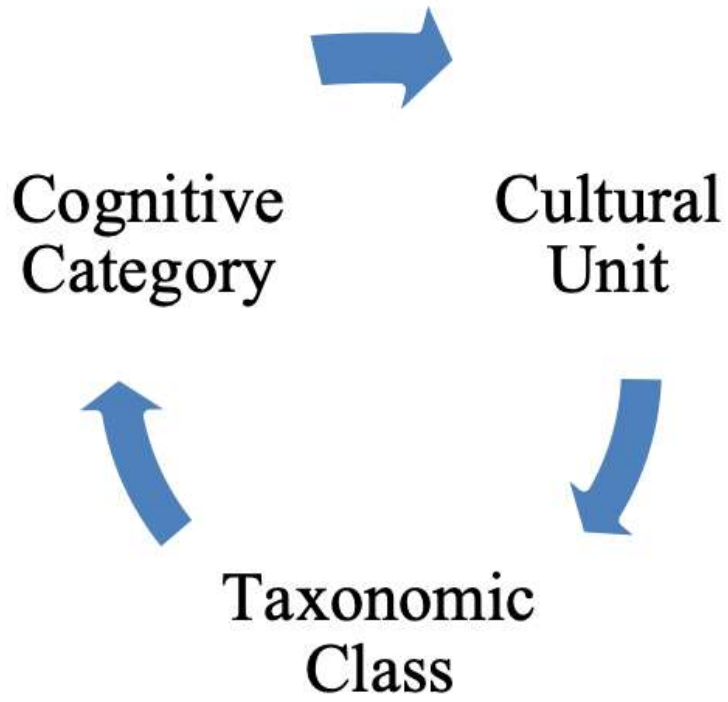


Figura 14: Ciclo de tres fases: Mendoza propone que la definición de un género musical depende de estos tres conceptos: Una variable cognitiva, una variable cultural y la clase taxonómica asignada.

4.3. Herramientas y modelos

Para la construcción de la plataforma se definen las tecnologías, herramientas y argumentos que permitirán la creación de la plataforma web y los modelos de *machine learning*. Entre las herramientas que permitirán la creación de la plataforma web podemos encontrar *React* y *Redux* en el *frontend* y en el *backend* está *Flask*. *Keras* con *Tensorflow* permitirá crear los modelos de inteligencia artificial. Y por ultimo se mencionará el modelo cliente-servidor que nos permitirá que los usuarios puedan utilizar la plataforma.

4.3.1. *React*

React es un proyecto con más de mil desarrolladores. *React* es una librería *open-source* disponible para *Javascript*. Sus funcionalidades permiten crear interfaces de usuario de una sola página, sin embargo se puede utilizar otras herramientas para crear más páginas. *React* permite crear las vistas a través de componentes y convertir el código de *Javascript* a una página HTML que permite que los desarrolladores puedan crear páginas web. (React, 2022)

Figura 15: Logotipo de *React*

4.3.2. *Redux*

Redux es una librería *open-source* del famoso lenguaje *Javascript*. Es utilizada para construir interfaces de usuario. *Redux* posee una pequeña *API* que permite que se puedan manejar los estados de la aplicación. El paradigma de programación en el que se basa es el funcional. Es por eso que se utilizan reductores para poder trabajar con *Redux*. (Redux, 2022)

Figura 16: Logotipo de *Redux*

Redux fue creado por Dan Abramov alrededor de junio de 2015. Se inspiró en *Flux* de *Facebook* y el lenguaje de programación funcional Elm. *Redux* se hizo popular muy rápidamente debido a su simplicidad, tamaño pequeño (solo 2 KB) y excelente documentación. (Redux, 2022)

En combinación con *React* estas dos herramientas permitirán que se pueda construir el *Frontend* de la aplicación. Generando los *dummy-componets* con *React* y conectándolo con *Redux* para poder trabajar los estados de la aplicación.

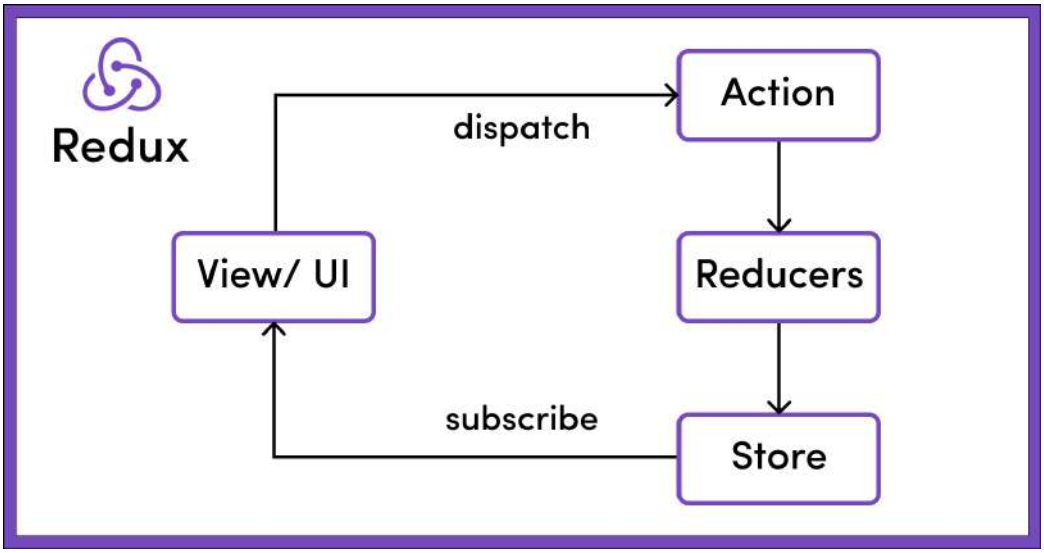


Figura 17: Estructura general de *Redux*

En la figura anterior se puede observar la estructura general de *Redux*. Como podrán ver *Redux* hace usó de un «dispatcher»y un « *suscriber*». El «*dispatcher*»se encargará de ejecutar las acciones y aplicarlas a cada uno de los reductores que el desarrollador haya creado. Cada reductor modificará el estado. Cada vez que el estado sea modificado se le

notificará a través del «*subscriber*»a todos los elementos subscritos, que por lo general es la interfaz gráfica.

4.3.3. *Flask*

Flask se basa en la especificación WSGI de Werkzeug y el motor de templates Jinja2 y tiene una licencia BSD. *Flask* es un *framework* del lenguaje de programación conocido como *Python*. Se le conoce como un *framework* minimalista ya que permite que se pueda crear aplicaciones web fácilmente, escribiendo muy pocas líneas de código. (Flask, 2022)

Es por eso que en comparación con otros *frameworks* de *Python Flask* no carga información innecesaria, ya que su objetivo no es proteger información o mejorar la ejecución de las bases de datos. Para cumplir con los objetivos de este proyecto *Flask* es perfecto, ya que permitirá crear los *endpoints*, de manera fácil, sencilla y haciendo uso de pocas líneas de código. (Flask, 2022)

Figura 18: Logotipo de *Flask*

A continuación se muestra una vista de la frase «Hello World»al hacer uso del método «Get»a la ruta «<MI URL>/»

```
from flask import Flask
app = Flask(__name__)

@app.route("/")
def helloWorld():
    return "Hello World!"

app.run(port=5000)
```

4.3.4. TensorFlow

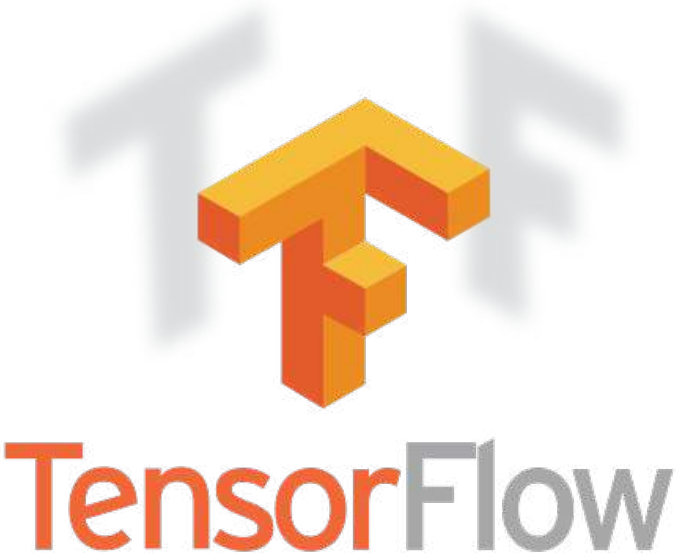


Figura 19: Logotipo de *TensorFlow*

TensorFlow es una librería *open-source* para aprendizaje automático a través de un rango de tareas. Fue desarrollado por *Google* para satisfacer sus necesidades de sistemas capaces de construir y entrenar redes neuronales para detectar y descifrar patrones y correlaciones, análogos al aprendizaje y razonamiento usados por los humanos. *TensorFlow* proporciona una *API* para muchos lenguajes entre estos se pueden listar: *Python*, *C++*, *Haskell*, *Java*, *Go* y *Rust*. También hay bibliotecas de terceros para *C*, *Julia*, *R*, *Scala* y *OCaml* (TensorFlow, 2022)

4.3.5. *Keras*

Es un *framework* de redes neuronales que está escrito en *Python*. *Keras* se puede ejecutar sobre *TensorFlow*. *Keras* contiene varias implementaciones de los bloques constructivos de las redes neuronales como lo son: *layers*, funciones objetivo, funciones de activación, y optimizadores matemáticos. Su código está alojado en *GitHub* y existen foros y un canal de *Slack* de soporte. demás del soporte para las redes neuronales estándar, *Keras* ofrece soporte para las redes neuronales convolucionales y para las Redes Neuronales Recurrentes. (Keras, 2022)

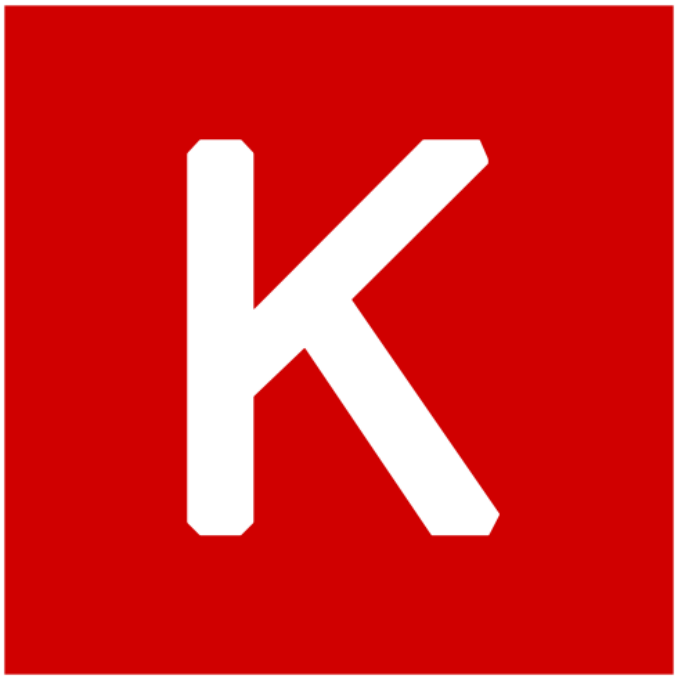


Figura 20: Logotipo de *Keras*

4.3.6. Modelo cliente-servidor

En el modelo cliente-servidor se dividen las tareas a realizar en una aplicación en dos grupos: Los proveedores y los demandantes. Los demandantes o clientes hacen peticiones a los proveedores o llamados también «el servidor». El servidor se encargará de dar la respuesta a la petición del cliente de acuerdo con el protocolo acordado. Esta idea también se puede aplicar a programas que se ejecutan sobre una sola computadora, aunque es más ventajosa en un sistema operativo multiusuario distribuido a través de una red de computadoras. Esto permite que se puedan ejecutar varios clientes sobre un solo servidor.

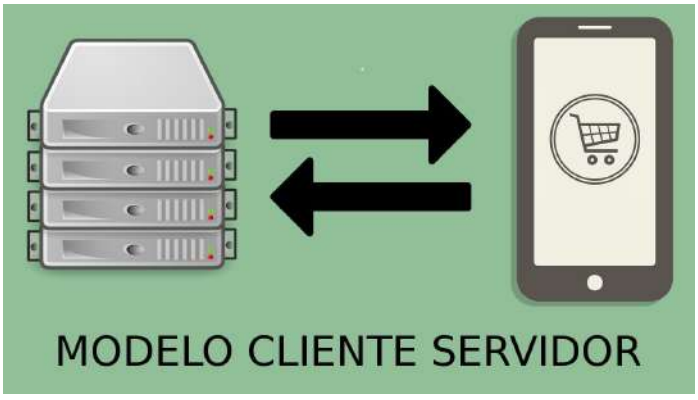


Figura 21: Modelo cliente-servidor

Por lo general, el administrador de procesos y el administrado de la base de datos se encuentran alojados en el servidor, ya que esto permite que exista consistencia en los datos. La interfaz gráfica suele ser el cliente, ya que esto permite que ese puedan crear distintas

aplicaciones con una solo base de datos. (Fazt, 2017)

En nuestra aplicación al servidor le llamaremos *backend* y al cliente *fronted*. En el *backend* estarán alojados los modelos y la función de transformación de audio a espectrograma. Y en el *frontend* se le permitirá al usuario subir un audio y observar los resultados.

CAPÍTULO 5

Metodología

5.1. Fase 1: Obtención de datos

1. Obtención de datos de los servidores de servicios de *Streaming*: Para extraer los datos se utilizarán las *APIs* de cada servicio, colocando un límite a la cantidad de canciones que se pueden extraer, para evitar sobre saturar los datos y que una computadora de uso particular los pueda procesar.

2. Extracción de los archivos de audio, ya sea utilizando el *API* directamente o haciendo uso de un *API* externa como *YouTube API*.

3. Conversión de los archivos de audio a espectrogramas.

5.2. Fase 2: Unificación de los datos

1. Durante esta fase se elaborará una unificación de los datos obtenidos de las *APIs*. Tomando en cuenta las variables que son significativas para el experimento. Se debe de tener en mente que los campos a utilizar tienen que estar presente en todos los *sets* de datos y se tendrá que establecer un protocolo para nombrar a todas las variables de tal forma que se pueda obtener un único *set* de datos.

5.3. Fase 3: Análisis exploratorio

1. Durante esta fase se realizará un análisis exploratorio que servirá para las siguientes fases. El tipo y el tamaño de los datos jugarán un papel importante, y la correlación entre las variables guiará las siguientes fases.

5.4. Fase 4: Evaluación para elección de algoritmos

1. Durante esta fase se estará evaluando la viabilidad de cada algoritmo basándose en análisis exploratorio. Entre los posibles algoritmos se proponen realizar (Se escogerán mínimo 3 algoritmos.):

    - Algoritmos de regresión
    - Algoritmos de regresión
    - Algoritmos Bayesianos
    - Algoritmos de *clustering*
    - Algoritmos de árboles de decisión
    - Redes convolucionales
    - *Feed Forward*
    - *LSTM*
    - *Le-Net*
    - Algoritmos de reducción de dimensión
    - Algoritmos de *Deep learning*
    - Redes convolucionales antagónicas

5.5. Fase 5: Elaboración de cada uno de los algoritmos

Para cada algoritmo:

1. Elección del lenguaje de programación para elaborar el algoritmo
2. Estudio de las mejores prácticas para su elaboración
3. Separación de datos de prueba y de entrenamiento
4. Construcción del algoritmo utilizando los datos de entrenamiento
5. Evaluación del algoritmo utilizando los datos de prueba
6. Si el algoritmo no posee la eficiencia requerida se procederá a regresar a la fase 4 a escoger un algoritmo alternativo.

5.6. Fase 6: Preparación para el sitio web

1. Se realizará un prototipo de interfaz para el sitio web a programar
2. Se validará la interfaz gráfica
3. Estudio de las herramientas para la construcción del mismo

5.7. Fase 7: Elaboración del sitio web

1. Durante esta fase se elaborará el sitio web planeado en la fase anterior siguiendo los siguientes pasos:
2. Elaboración del *backend*
3. Elaboración de un diagrama de entidad - relación
4. Se plantea subir los algoritmos a un servidor en donde puedan estar alojados y que sean llamados por medio de un *API*, siguiendo el modelo cliente-servidor.
5. Elaboración de los modelos
6. Elaboración de los controladores
7. Elaboración de los *endpoints*
8. Elaboración del *frontend*
9. Elaboración de un prototipo
10. Elaboración de la interfaz gráfica
11. Elaboración de sagas

5.8. Fase 8 de publicación del sitio web

1. Durante esta fase se publicará el sitio web utilizando las herramientas acordadas en la fase 6.

5.9. Fase 9 Pruebas de usuario

1. Durante esta fase se realizará pruebas de usuario con miembros de la comunidad de la Universidad del Valle de Guatemala que pertenezcan o pertenecerán a la industría musical.

CAPÍTULO 6

Obtención de datos

6.1. *SoundCloud API*

SoundCloud es una plataforma de *streaming*, que permite que usuarios con acceso a internet puedan subir sus audios a la plataforma, sin necesidad de pagar una subscripción o una tarifa para mantener el el audio dentro de la plataforma. Esto hace que utilizar los datos de las canciones dentro de la plataforma sea más efectivo para cumplir con los objetivos del proyecto.

Souncloud API alojado en un servidor permite que usuarios desarrolladores puedan crear aplicaciones utilizando los datos de *SoundCloud*.

6.1.1. Ventajas de *Souncloud API*

Al analizar las distintas *APIs* de los servicios de *streaming*, *SoundCloud* representa une ventaja objetiva, debido a los siguientes motivos:

- El género de las canciones esta estrictamente relacionado con las mismas canciones, como una propiedad de la misma. En otras plataformas el género esta relacionado con el artista, lo cual representaba una desventaja de análisis para el proyecto.
- El género es asignado por el artista. Esto representa una ventaja ya que nos permite eliminar géneros regionales (British Pop, Latin Rock, Indian Reggea, etc.), que son asignados con un objetivo comercial para el lugar de origen del artista.
- El acceso a los datos es gratuito.

- Las canciones son descargables o se puede encontrar canciones descargables entre los datos.

6.2. Géneros musicales encontrados en *SoundCloud API*

Se muestran los géneros musicales que se encontraron útiles para generalizar al analizar los datos proveídos. Cabe resaltar que el nombre de los géneros se muestra como "Tokens", aquellas cadenas de caracteres que coinciden con el nombre del género. Se creó una carpeta para guardar cada canción en su respectivo género.

Cumbia	Indian	Rap/Hiphop/hip-hop/hip hop
Bachata	Arabic	Funk
Salsa	Soul	Afrobeat
Reggae	Dance/EDM/Electronic/Electro	Dembow
Gospel	Alternative/Indie	Bossa Nova
Punk	Future/Futuristic	Rock
Latin	Folk/Folkloric	Pop
Trap	Norteña/Nortena/Banda/Corridos	Jazz
Country	R&B/R & B/rnb	House
Disco	Reggaeton/Reguetón	Oriental
Acoustic/Singer/Songwriter	Classical/Classic	Metal
Hardstyle	Samba/Brazilian	Acapella
World	Ambient	HardCore

Cuadro 3: Géneros musicales encontrados al utilizar *SoundCloud API*

6.3. Estructura de datos obtenida

A continuación se muestra la estructura de datos obtenida. Esta se encuentra en formato JSON:


```
{
    'artwork_url':
        'https://i1.sndcdn.com/artworks-C91oqJqSiy69FNC1-P4h1rw-large.jpg',
    'caption': None,
    'commentable': True,
    'comment_count': 1974,
    'created_at': '2020-10-30T23:59:29Z',
    'description': '',
    'downloadable': True,
    'download_count': 240,
    'duration': 2380931,
    'full_duration': 2380931,
    'embeddable_by': 'all',
    'genre': 'Dance/EDM/Electronic/Electro',
    'has_downloads_left': True,
```

```
'id': 920671105,
'kind': 'track',
'label_name': None,
'last_modified': '2022-03-24T23:52:45Z',
'license': 'all-rights-reserved',
'likes_count': 51847,
'permalink': 'quien-lo-diria-halloween-by-alex-hard',
'permalink_url':
    'https://soundcloud.com/djalexhard/quien-lo-diria-halloween-by-alex-hard',
'playback_count': 1688079,
'public': True,
'publisher_metadata': {
    'id': 920671105,
    'urn': 'soundcloud:tracks:920671105',
    'contains_music': True
},
'purchase_title': None,
'purchase_url': None,
'release_date': None,
'reposts_count': 847,
'secret_token': None,
'sharing': 'public',
'state': 'finished',
'streamable': True,
'title': '',
'track_format': 'single-track',
'uri': 'https://api.soundcloud.com/tracks/920671105',
'urn': 'soundcloud:tracks:920671105',
'user_id': 151527646,
'visuals': None,
'waveform_url': 'https://wave.sndcdn.com/GCoRWZiUJ33U_m.json',
'display_date': '2020-10-31T01:05:44Z',
'media': {
    'transcodings': []
},
'station_urn': 'soundcloud:system-playlists:track-stations:920671105',
'station_permalink': 'track-stations:920671105',
'monetization_model': 'NOT_APPLICABLE',
'policy': 'ALLOW',
'user': {
    'avatar_url':
        'https://i1.sndcdn.com/avatars-H3Dr3sbQUg0rufJm-amaS5g-large.jpg',
    'city': 'Cali',
    'comments_count': 0,
    'country_code': 'CO',
    'created_at': '2015-05-06T18:48:59Z',
    'creator_subscriptions': [{
        'product': {
            'id': 'creator-pro-unlimited'
        }
    }],
    'creator_subscription': {
        'product': {
            'id': 'creator-pro-unlimited'
```

```
            }
        },
        'description': 'Contacto: +57 3145106448 \\n\\n',
        'followers_count': 52445,
        'followings_count': 71,
        'first_name': 'Alexander',
        'full_name': 'Alexander Cuspin Zuiga',
        'groups_count': 0,
        'id': 151527646,
        'kind': 'user',
        'last_modified': '2022-04-13T01:36:27Z',
        'last_name': 'Cuspin  Zuiga ',
        'likes_count': 43,
        'playlist_likes_count': 0,
        'permalink': 'djalexhard',
        'permalink_url': 'https://soundcloud.com/djalexhard',
        'playlist_count': 0,
        'reposts_count': None,
        'track_count': 5,
        'uri': 'https://api.soundcloud.com/users/151527646',
        'urn': 'soundcloud:users:151527646',
        'username': 'ALEX HARD',
        'verified': False,
        'visuals': {
            'urn': 'soundcloud:users:151527646',
            'enabled': True,
            'visuals': [],
            'tracking': None
        },
        'badges': {
            'pro': False,
            'pro_unlimited': True,
            'verified': False
        },
        'station_urn': 'soundcloud:system-playlists:artist-stations:151527646',
        'station_permalink': 'artist-stations:151527646'
    }
}
```

A continuación se muestra la cantidad de canciones que se obtuvo de cada género, se transformó el nombre, para convertirlo en el identificador de clase para cada género.

Género	Cantidad de canciones obtenidas
Salsa	8
Cumbia	1
Bachata	9
Dembow	1
Arabic	2
Folk/Folkloric	2
Jazz	201
Alternative/Indie	2
Metal	102
Afrobeat	3
Jam	3
Reggaeton	3
Future/Futuristic	4
Ambient	4
Country	104
Blues	105
Norteña	5
Trance	10
Gospel	6
Songwriter	6
Soul	6
HardCore	7
Dubstep	11
Hip Hop	317
Disco	193
World	23
Reggae	151
Funk	34
R&B	26
Pop	133
Rock	156
Classical	145
Latin	12

Cuadro 4: Cantidad de canciones obtenidas después de ejecutar el algorimto de obtención de datos

CAPÍTULO 7

Procesamiento de datos

7.1. Transformación de canciones

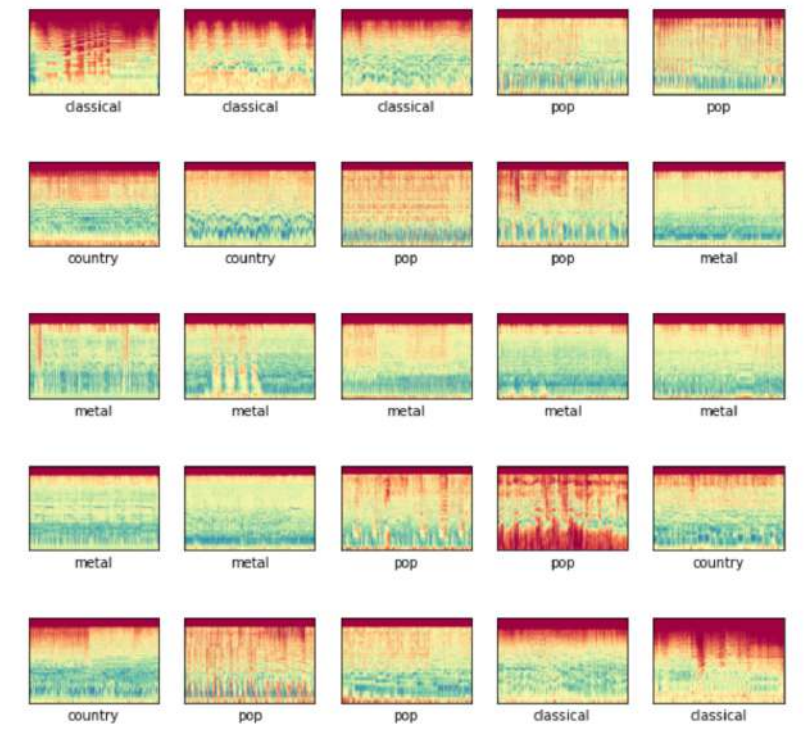


Figura 22: Imágenes formadas al aplicar la transformación de audio a espectrograma

Gracias a que cada canción fue guardada con el ID, tanto dentro del dataframe, como en su respectivo género se procedió a transformar cada canción a un espectrograma.

7.2. Estructura del espectrograma

Como se puede observar en la figura anterior al realizar la transformación de una canción (audio) a un espectrograma, se obtiene un tipo de gráfica de calor. Existen tres dimensiones que se pueden encontrar dentro del espectrograma:

7.2.1. Amplitud

La amplitud o intensidad de audio está gráficada en el eje z, representada por el color observable en la gráfica. *Matplotlib* provee una facilidad para utilizar distintos espectros de colores para poder representar distintos valores. Es necesario utilizar un espectro de colores divergentes para que exista la mayor diferencia entre una amplitud alta y una amplitud baja.

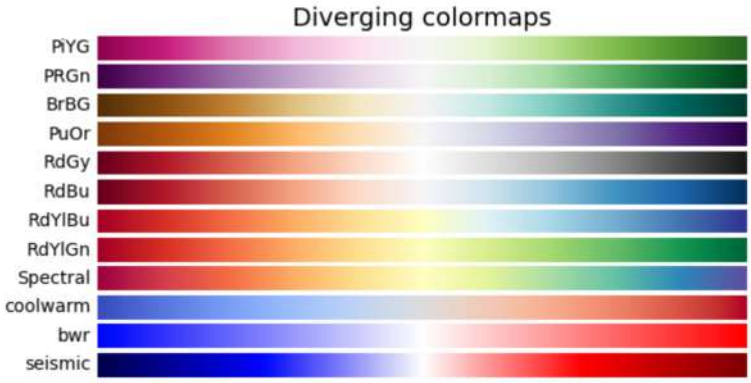


Figura 23: Espectros de colores divergentes proveídos por *Matplotlib* (Matplotlib, 2022)

En la figura anterior se observa distintas opciones para representar la amplitud, en estos casos se puede observar que los espectros más divergentes son el «seismic»y el «spectral». Se escogió el «spectral»gracias a que tiene más valores intermedios y se podrá utilizar para representar amplitudes de valor medio.

7.2.2. Tiempo

El segundo valor que se puede observar dentro de cada espectrograma es el valor temporal. Como fue mencionado con anterioridad en una canción el ritmo es medido a través de los «bpm»(beats por minuto). Esto puede ser evidenciado en el eje de las abscisas. Gracias a que en el formato de audio cada compás es representado a través de un arreglo de objetos que representan cada espacio de audio, que pueden ser transformados en un arreglo que representan pixeles.

7.2.3. Frecuencia

El tercer y último valor que se observa en los espectrogramas es la frecuencia que se puede escuchar. Como se conversó con anterioridad, es posible convertir una combinación lineal de frecuencias y obtener los tonos puros que causan dicha combinación a través de la conocida transformada de Fourier. Las frecuencias se pueden encontrar en eje de las ordenadas, estando las frecuencias bajas en parte inferior del espectrograma y las frecuencias altas en la partes superior del mismo.

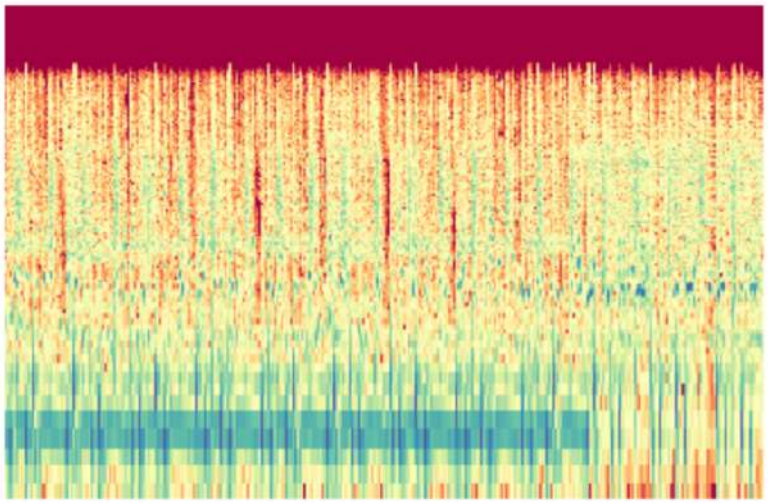


Figura 24: Espectrograma de canción hip-hop, en donde se observa que la canción carece de altos debido a que tiene una amplitud cercana a cero (color rojo). Mientras que se puede observar un rango de verdes en la parte inferior. (Probablemente el kick de la batería, propio del género)

7.3. Normalización de longitud de audio

Existe un problema al momento de entrenar los modelos de inteligencia artificial con los espectrogramas. Este problema se hace presente es que la longitud de audio, ya que esta es distinta para cada canción. Al realizar la transformación a espectrograma obtenemos imágenes de $n * m$ donde m es un número variable, lo cual causa que el tensor a utilizar, para los modelos, no tenga la misma estructura. Es por eso que es necesario realizar una normalización para que toda la canción o parte de ella esté contenida en un mismo tamaño de imagen $n_0 * m_0$ siendo estos dos valores iguales para todos los espectrogramas

CAPÍTULO 8

Primer Modelo Red Convolucional Neuronal (CNN)

Para la creación del primer modelo se decidió trabajar con una red convolucional ya que esta facilita la identificación de imágenes, con amplia demostración sobre el éxito de las mismas. Además se ha decidido limitar la cantidad de clases determinado por la cantidad de canciones que se obtuvieron de dicho género. Las clases utilizadas en el entrenamiento y prueba del modelo son:

Género/Clase	Cantidad de canciones
Jazz	201
Metal	102
Country	104
Blues	105
Hip Hop	317
Disco	193
Reggae	151
Pop	133
Rock	156
Classical	145

Cuadro 5: Clases y cantidad de canciones asignadas para el primer modelo.

8.1. Estructura del modelo

A continuación se muestra la estructura del modelo de red convolucional neuronal que se utilizó. Se puede observar que es un modelo secuencial y cuya entrada es correspondiente al tamaño de las imágenes. Se utilizó una tamaño de 222 x 222 píxeles para poder reducir

espacio al momento de ejecutar el modelo y para una mayor rapidez al momento de ejecutar el modelo dentro de la página web. También se comparó con imágenes de 1024 x 1024 píxeles y 512 x 512 píxeles, sin embargo no se evidenció mejora alguna en la efectividad del modelo, por lo que se optó por utilizar este tamaño para el entrenamiento y las pruebas.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 200, 200, 32)      896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 100, 100, 32)      0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 98, 98, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 49, 49, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 47, 47, 64)        36928
_________________________________________________________________
flatten (Flatten)            (None, 141376)            0
_________________________________________________________________
dense (Dense)                (None, 64)                9048128
_________________________________________________________________
dense_1 (Dense)              (None, 4)                 260
=================================================================
Total params: 9,104,708
Trainable params: 9,104,708
Non-trainable params: 0
_________________________________________________________________
```

Figura 25: Estructura de red convolucional (modelo inicial)

8.2. Precisión del modelo

A continuación se muestra el modelo con quince épocas de entrenamiento:

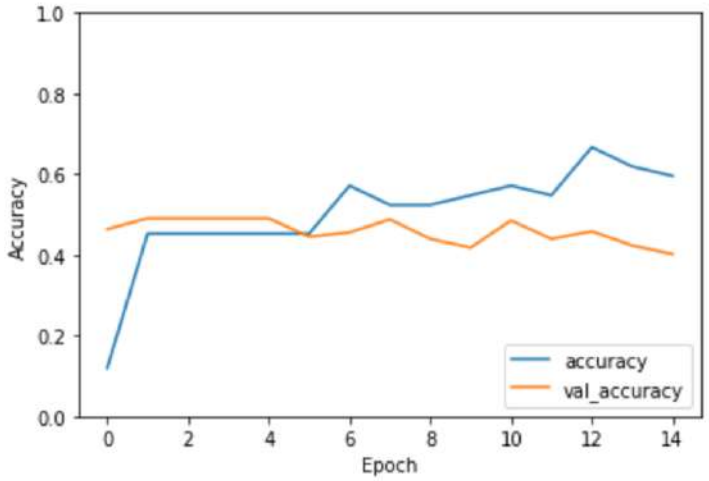


Figura 26: Primer modelo de red convolucional entrenado 15 épocas

Al observar las las gráficas de efectividad del modelo se puede ver como este se queda

estancado en 0.4 y 0.6 de efectividad aproximadamente. Esto con 10 clases distintas como fue mencionado con anterioridad. Esto representa un problema grande ya que una persona podría bien elegir al azar entre 4 y 5 veces y la probabilidad de acertar sería exactamente la misma. La efectividad del modelo no es válida para el objetivo de este estudio, es por eso que se decidirá reducir la cantidad de clases o géneros para poder construir los modelos necesarios.

CAPÍTULO 9

Elección de géneros a utilizar en los modelos

Como parte de la creación del modelo es necesario reducir la cantidad de clases ya que se evidenció que la efectividad de los modelos es inversamente proporcional a la cantidad de clases presentes. Se agrupó la cantidad total de géneros en combinatorias de 4. y se construyeron modelos con dichas combinatorias. A continuación se muestran los géneros con mayor frecuencias en los modelos con las efectividades más altas:

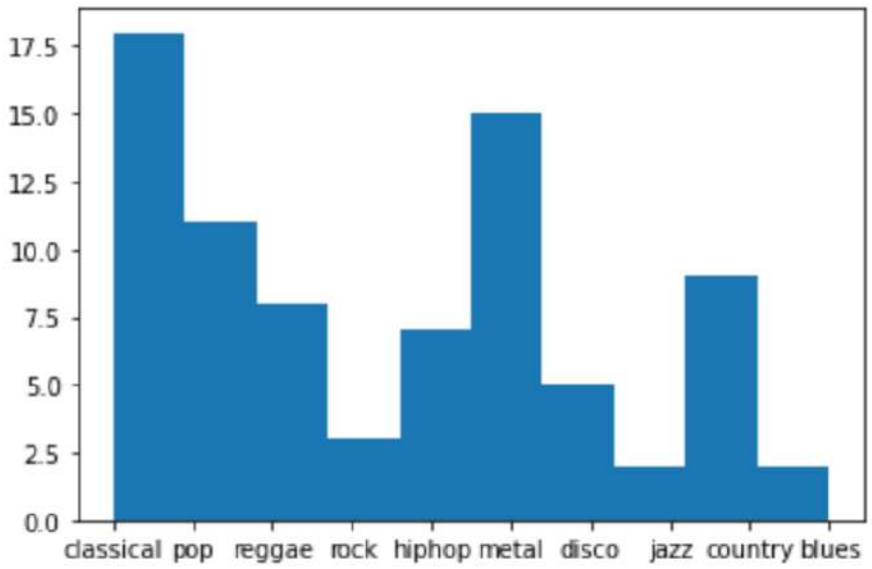


Figura 27: Géneros con mayor frecuencia en modelos de redes convolucionales neuronales arriba de 0.65 de efectividad

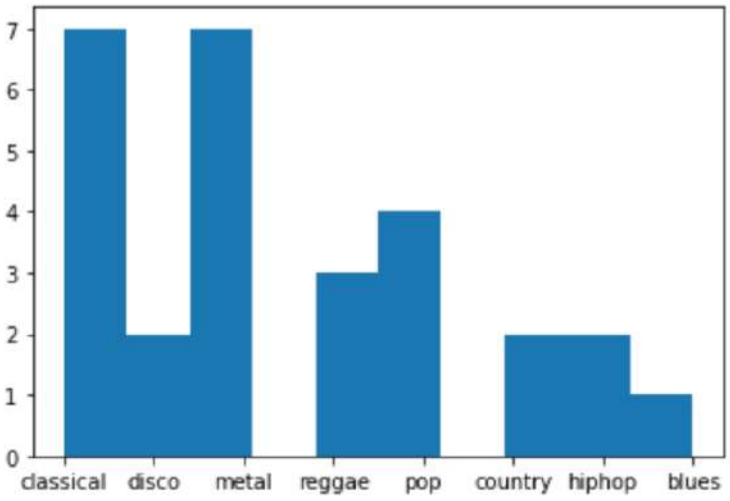


Figura 28: Géneros con mayor frecuencia en modelos de redes convolucionales neuronales arriba de 0.70 de efectividad

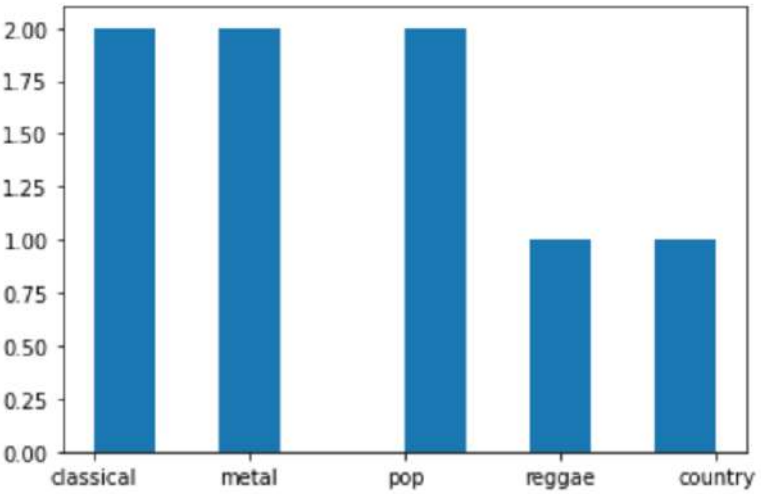


Figura 29: Géneros con mayor frecuencia en modelos de redes convolucionales neuronales arriba de 0.75 de efectividad

Se pudo evidenciar que existían parejas de géneros que provocaban que la efectividad bajase, entre estos se encuentran el rock y el pop. Cabe mencionar que las características estructurales de ambos géneros provocan que estos suenen similares, o que sus espectrogramas se vean similares. En general una canción de pop y una de rock tienen una estructura verso-coro-verso-coro-puente-coro. Lo cual causa que las amplitudes se vean más verdes en las secciones del coro. También cabe mencionar que ambos géneros son interpretados y producidos utilizando los mismos instrumentos o un grupo de instrumentos similares, lo cual causa que los rangos de frecuencias sean muy similares.

Así como el pop y rock existen otros géneros que también son interpretados o producidos

haciendo uso del mismo grupo de instrumentos, lo cual provoca que la efectividad de lo modelos baje.

Si se observa las figuras anteriores se puede observar como los géneros «classical», «metal»y «pop»son aquellos que más apariciones tienen en los modelos con efectividades altas. Es por esto que estos tres modelos son elegibles para el modelo final.

Por otra parte, a lo largo del entrenamiento se pudo evidenciar como las canciones de «reggae»causan ruido debido al uso de la guitarra y la reverberación de la misma, lo cual causa que los espectrogramas sean confundibles con los de «rock», «pop»y «metal». Es por eso que se ha escogido que los géneros disponibles para el modelo sean:

Género	Cantidad de canciones
Metal	102
Country	104
Pop	133
Classical	145

Cuadro 6: Géneros escogidos para entrenamiento de modelos finales.

CAPÍTULO 10

Modelo *Feed-Forward*

Comenzaremos con un modelo que tiene una pésima estructura y que se ha demostrado que en términos de inteligencia artificial son casi inservibles para poder comparar el resto de modelos que se van a elaborar.

10.1. Estructura del modelo

El modelo cuenta con tres capas y una tercera para introducir el tensor de entrada. Cabe resaltar la primera capa « *Dense*»es la capa escondida. Este modelo solo alimenta y se mueve a la siguiente capa.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 198, 198, 64)      1792
_________________________________________________________________
flatten (Flatten)            (None, 2509056)           0
_________________________________________________________________
dense (Dense)                (None, 16)                40144912
_________________________________________________________________
dense_1 (Dense)              (None, 4)                 68
=================================================================
Total params: 40,146,772
Trainable params: 40,146,772
Non-trainable params: 0
_________________________________________________________________
```

Figura 30: Estructura del modelo *Feed Forward*

10.2. Precisión del modelo

Acá se puede observar lo poco efectivo que es la estructura *feed forward* que nos muestra como el modelo se mantuvo con 0.2 de efectividad durante 40 épocas. Obteniendo un 0.2612 de efectividad final para el modelo. Esto es causado debido a que el modelo no tiene retroalimentación y tampoco existe un *pooling* como en una red convolucional neuronal.

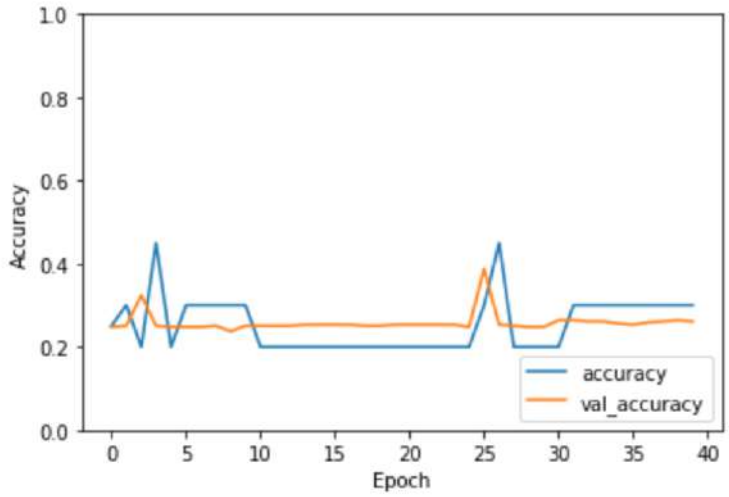


Figura 31: Efectividad del modelo *Feed Forward*

CAPÍTULO 11

Modelo *Le-Net*

Este modelo es más complejo en comparación con el *feed forward*, y ha sido utilizado en entornos de desarrollo y producción en otras aplicaciones prácticas. Es por eso que este modelo tiene una efectividad relativamente más alta pero también tiene una estructura más completa.

11.1. Estructura del modelo

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 198, 198, 6)       168
_________________________________________________________________
average_pooling2d (AveragePo (None, 99, 99, 6)         0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 97, 97, 16)        880
_________________________________________________________________
average_pooling2d_1 (Average (None, 48, 48, 16)        0
_________________________________________________________________
flatten (Flatten)            (None, 36864)             0
_________________________________________________________________
dense (Dense)                (None, 128)               4718720
_________________________________________________________________
dense_1 (Dense)              (None, 4)                 516
=================================================================
Total params: 4,720,284
Trainable params: 4,720,284
Non-trainable params: 0
_________________________________________________________________
```

Figura 32: Estructura del modelo *Le-Net*

En la estructura del modelo *Le-Net* se puede observar como el este tiene dos parejas de capas de convolución y *pooling* seguidas. Permitiendo que sea un modelo de red convolucional fuerte. Luego tenemos tres capas que permite reducir la dimensión de los parámetros a uno para poder llegar al número de clases.

11.2. Precisión del modelo

Al observar la gráfica de precisión se observa como esta avanza mucho más a lo largo de las épocas en comparación al modelo de *feed foward*. Acá se observa como a partir de la época 20 van creciendo la efectividad comienza a crecer con constancia. Si se traza una regresión lineal a partir de ese punto se podrá observar como esta tiene una pendiente positiva, lo cual puede representar que con más entrenamiento el modelo puede mejorar, sin embargo la distancia entre los parámetros que se observan en la gráfica comienzan a separarse, por lo que se puede llegar a un sobre entrenamiento. La efectividad final del modelo es de 0.7230.

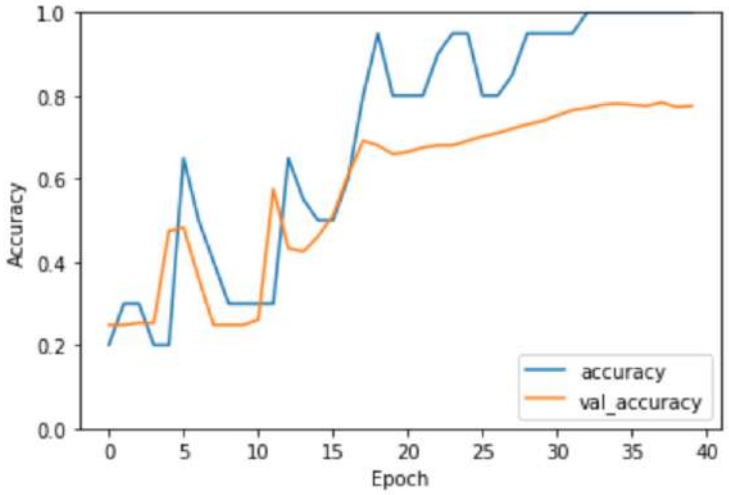


Figura 33: Efectividad del modelo *Le-Net*

CAPÍTULO 12

Modelo red convolucional neuronal

Para mejorar la efectividad y alcanzar los objetivos propuestos en este documento será necesario hacer pequeñas modificaciones al modelo *Le-Net* y proponer nuestro propio modelo de red convolucional neuronal.

12.1. Estructura del modelo

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 200, 200, 32)      896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 100, 100, 32)      0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 98, 98, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 49, 49, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 47, 47, 64)        36928
_________________________________________________________________
flatten (Flatten)            (None, 141376)            0
_________________________________________________________________
dense (Dense)                (None, 64)                9048128
_________________________________________________________________
dense_1 (Dense)              (None, 4)                 260
=================================================================
Total params: 9,104,708
Trainable params: 9,104,708
Non-trainable params: 0
_________________________________________________________________
```

Figura 34: Estructura del modelo CNN

En este modelo se puede observar como se reemplazan las capas de *pooling* por *max-pooling* permitiendo que la efectividad suba. Además cabe resaltar que las tres capas finales fueron reemplazadas a dos nuevas capas para poder llegar al número de clases finales.

12.2. Precisión del modelo

La efectividad aumenta relativamente respecto al modelo *LetNet*, La nueva efectividad es de 0.7704. El cambio en las capas de *MaxPooling* permite que la efectividad aumente, además de que las diferencias entre los dos parámetros de las gráficas ha disminuido relativamente.

Este modelo permite que el objetivo sea alcanzado. A partir de la época 25 el objetivo esta alcanzado sin embargo se puede notar como la efectividad este modelo se mantiene cercana a los 0.75, lo cual nos indica que se requiere de más datos para poder elevar la efectividad del modelo.

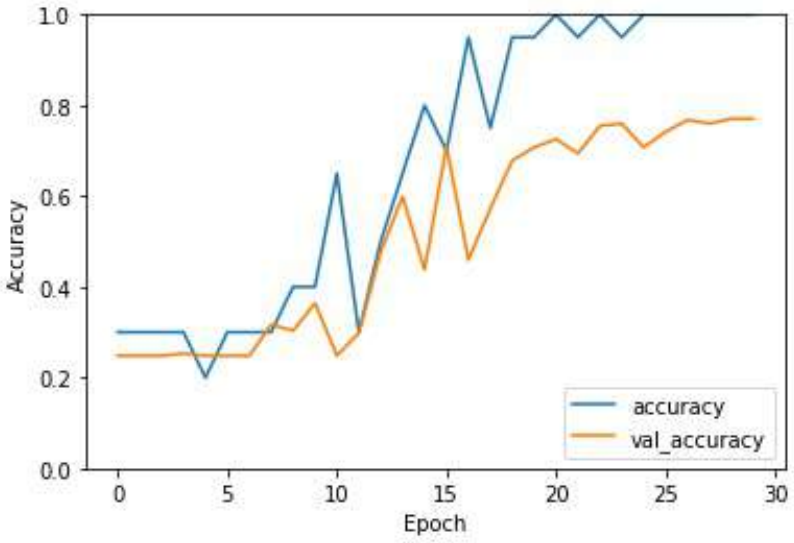


Figura 35: Efectividad del modelo CNN

CAPÍTULO 13

Modelo *LSTM*

Al agregar una capa *LSTM* al modelo resultante de la red convolucional neuronal, se puede observar que la efectividad asciende de una manera más constante en comparación con el modelo de la red convolucional neuronal sin *LSTM*. Lo cual indica que la estructura del modelo es válido para un entrenamiento más completo, con más datos y con más épocas.

13.1. Estructura del modelo

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 198, 198, 6)       168
_________________________________________________________________
average_pooling2d (AveragePo (None, 99, 99, 6)         0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 97, 97, 16)        880
_________________________________________________________________
average_pooling2d_1 (Average (None, 48, 48, 16)        0
_________________________________________________________________
flatten (Flatten)            (None, 36864)             0
_________________________________________________________________
reshape (Reshape)            (None, 1, 36864)          0
_________________________________________________________________
lstm (LSTM)                  (None, 128)               18940416
_________________________________________________________________
dense (Dense)                (None, 4)                 516
=================================================================
Total params: 18,941,980
Trainable params: 18,941,980
Non-trainable params: 0
_________________________________________________________________
```

Figura 36: Estructura del modelo *LSTM*

Si se observa la estructura del modelo se puede ver como en la penúltima línea aparece la capa *LSTM* agregada que permite que el modelo tenga una estructura de retroalimentación.

13.2. Precisión del modelo

Como fue mencionado con anterioridad se puede observar como la efectividad asciende con constancia. Si se traza una regresión lineal a la gráfica en lugar de hacer trazados continuos en cada punto del diagrama, se podrá observar que se tiene una pendiente positiva que seguirá ascendiendo si se entrena con más épocas. Sin embargo debido a las diferencias entre los valores «*accuracy*»y «*val_ accuracy*»se ha decidido continuar con 50 épocas para evitar una posible causa de sobre entrenamiento. Obteniendo una precisión final de 0.7467.

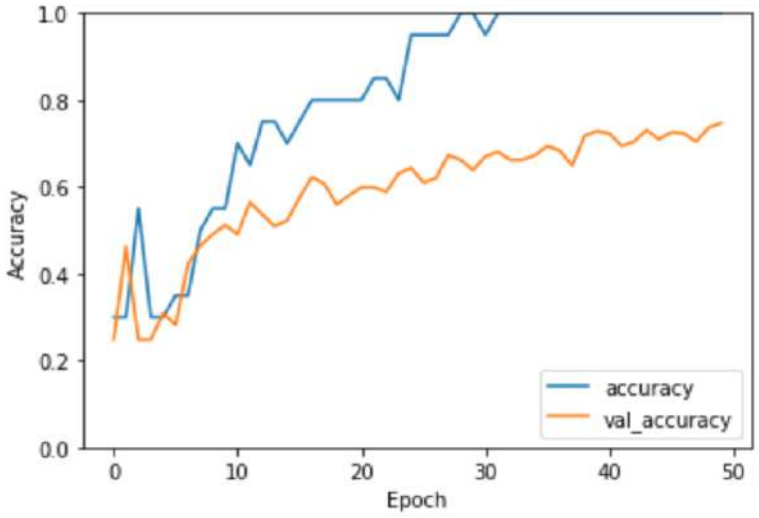


Figura 37: Efectividad del modelo *LSTM*

CAPÍTULO 14

Página Web (*backend)*

Para poder proporcionar acceso a todos los modelos creados fue necesario crear una plataforma *backend*. Esta plataforma fue construida en *Flask*, que permite la facilidad de carga de los modelos construidos en *Python* y no carga espacio innecesario como lo hacen otros *frameworks*.

14.1. Estructura de respuesta


```
{
    "CNN Model":  {
        "data": {
            "classical":"0.32583416",
            "country":"0.20497286",
            "metal":"0.0065581384",
            "pop":"0.46263486"
            },
        "metrics": {
        "accuracy":"0.7704485654830933",
        "loss":"0.7446410059928894\n"
        }
    },
    "success":true
}
```

14.2. *Deployment*

Por cuestiones de costo se tomó la decisión de alojar el *backend* a través de *Ngrok* debido a que el peso de los modelos superan 1GB esto se debe a que el modelo *Feed Forward* es muy pesado.

Alojar toda esta información conlleva un consto adicional que en la curva de costo-oportunidad del proyecto nos indica que no es necesario gastar, ya que el objetivo es llegar a tener un primer acercamiento en la clasificación de géneros.

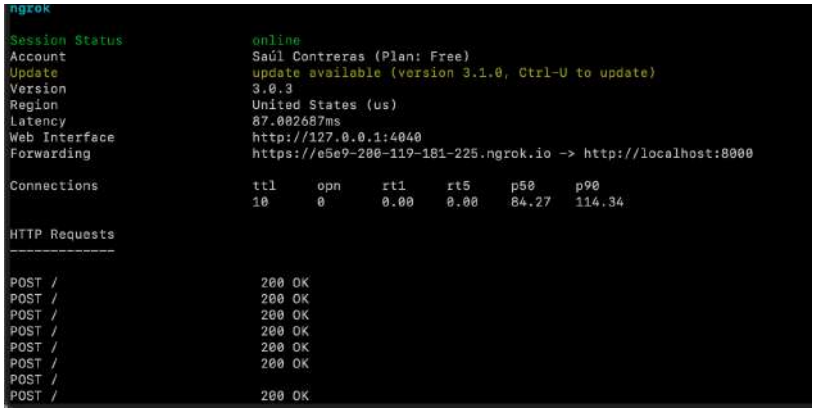


Figura 38: Captura de terminal con servidor corriendo desde *Ngrok*

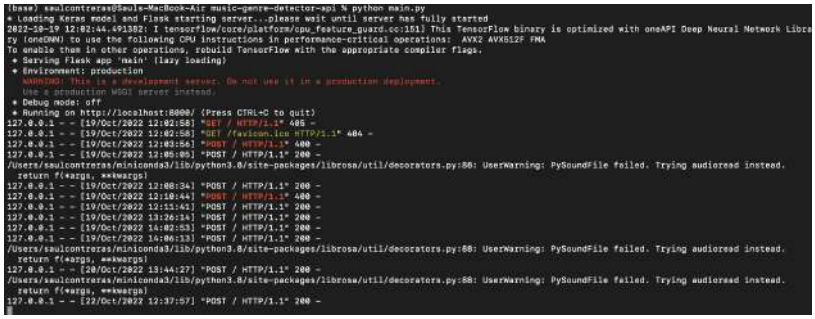


Figura 39: Captura de terminal con servidor corriendo desde la IP de originen

CAPÍTULO 15

Página Web (*frontend*)

15.1. Enlace

`https://suulcoder.github.io/music-genre-detector/`

15.2. Paleta de colores

La paleta de colores que se utilizó para el proyecto es la siguiente que se construyó haciendo uso de la plataforma de *Adobe Color Wheel*(Adobe, 2022)

Se puede observar como el color principal es el que se utiliza en el encabezado del sitio web. El color más oscuro de toda la paleta se utiliza como fondo de la aplicación para que el blanco y los colores claros de la paleta puedan ser utilizados como color para la fuente.

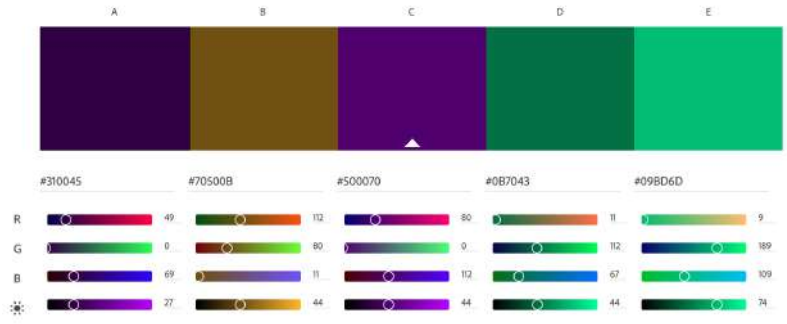


Figura 40: Paleta de colores utilizada en la plataforma

15.3. Interfaz gráfica

A continuación se muestra la interfaz gráfica creada para acceder a los modelos.

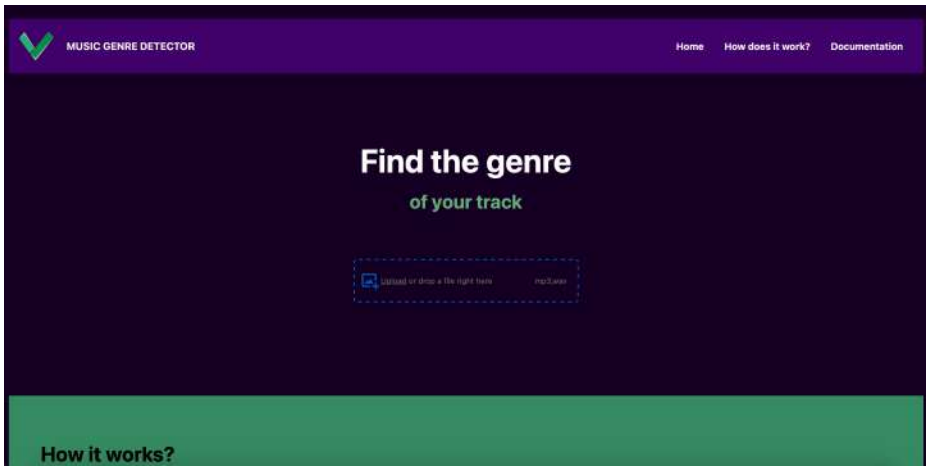


Figura 41: Interfaz gráfica. Página *Home*

En la figura anterior se puede observar la página de inicio en donde se puede cargar un archivo. Se le solicita al usuario cargar un archivo de audio para identificar el género.

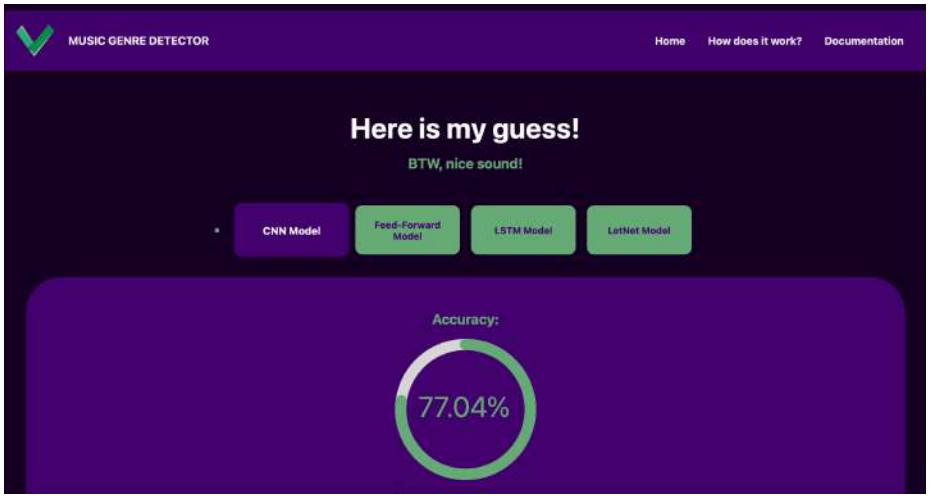


Figura 42: Interfaz gráfica. Modelo cargado

En esta figura se observa como se ve cuando el modelo está cargado, mostrando los cuatro modelos que fueron mencionados con anterioridad.

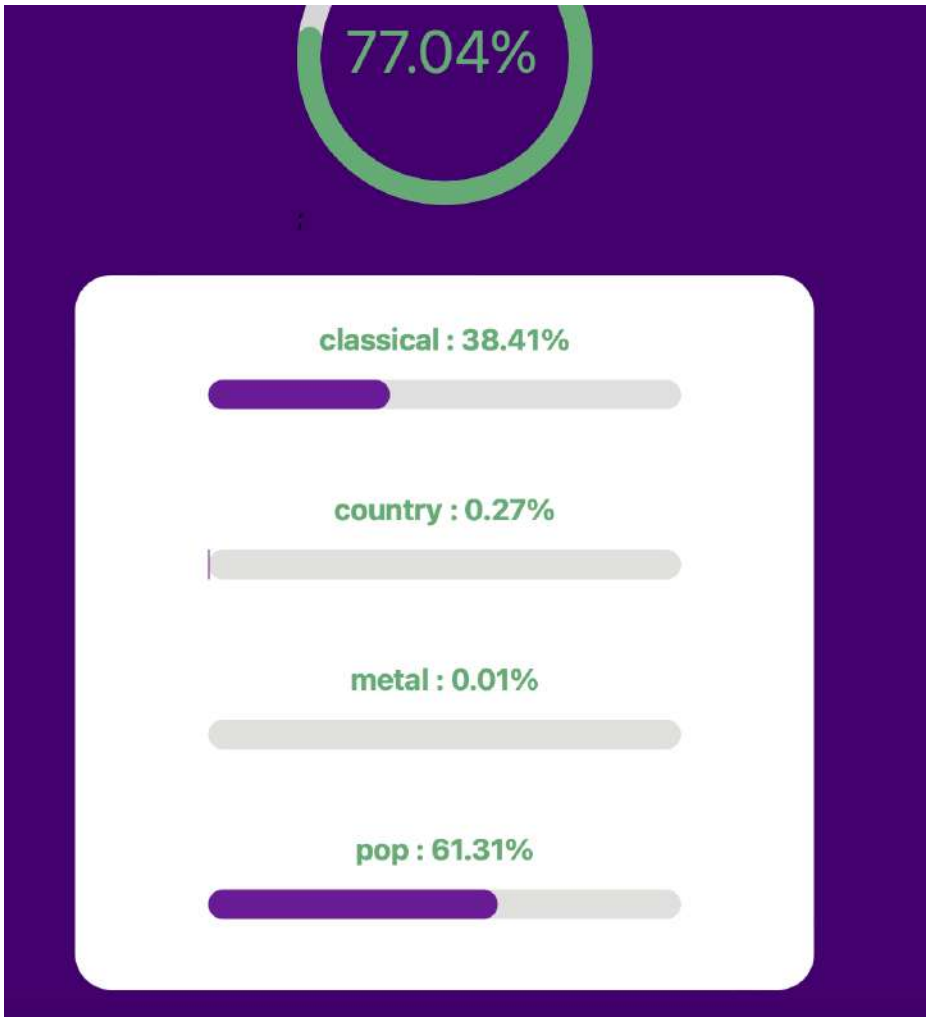


Figura 43: Interfaz gráfica. Barras de proporción dependiendo del modelo.

En estas fotografías se puede observar como para cada canción cargada se devuelve un porcentaje para los cuatro géneros que se encuentran en los modelos. Esto se debe a que una canción moderna puede o no pertenecer a varios modelos. Siguiendo esta hipótesis en la interfaz gráfica no solo se muestra el modelo con el porcentaje más grande de probabilidad sino que todas las probabilidades.

Figura 44: Interfaz gráfica. Botón de retorno a página *Home*

15.4. *Deployment*

Siguiendo el mismo principio de costo-oportunidad se optó por realizar un *deployment* integrado, sin costo. a través de *Github-pages*. La plataforma se encuentra disponible para todos aquellos que deseen utilizarlo. (Contreras, 2022)

CAPÍTULO 16

Pruebas de usuario

Para poder verificar las funcionalidades de la plataforma se realizaron pruebas de usuarios en la comunidad de la Universidad del Valle de Guatemala. La muestra de usuarios seleccionada se concentra en estudiantes de composición y producción musical bajo el curriculum académico de la Universidad del Valle de Guatemala en el año 2022.

Parte del público objetivo de la plataforma está subdivido en estudiantes de composición musical, producción, *screenwriting* para películas y obras teatrales o carreras académicas afines a las anteriores. Se consideró necesario hacer pruebas de usuario para poder comprobar la funcionalidad principal de la plataforma. Este grupo de estudiantes se convertirán en miembros activos de la industria musical en cada una de sus subdivisiones, por lo que son una muestra ejemplo de la utilidad de la plataforma.

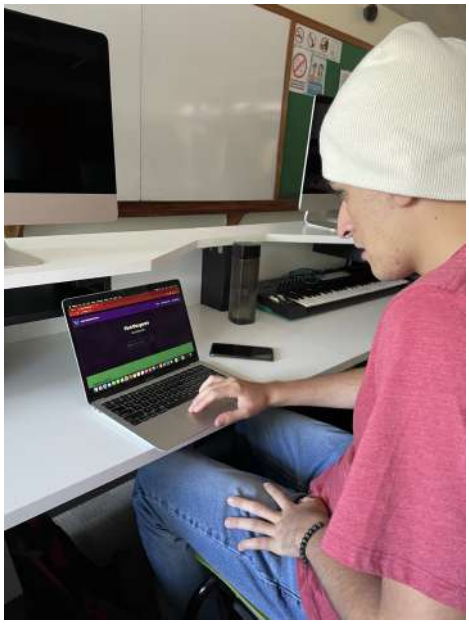


Figura 45: Estudiante de composición y producción musical (usuario 1) probando la funcionalidad de los modelos

El primer usuario que puso a prueba la plataforma es un estudiante de composición musical y producción musical. Sus intereses son la música experimental y la composición de arreglos orquestales. Encontró interesante que la aplicación logrará identificar entre pistas de audio de música clásica. Sin embargo, hizo la observación que sería un agregado que la aplicación identificara más géneros musicales más comerciales.

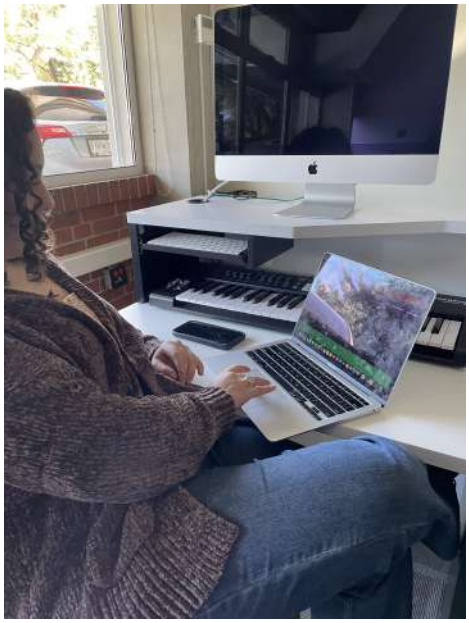


Figura 46: Estudiante de composición y producción musical (usuario 2) probando la funcionalidad de los modelos

El segundo usuario que comprobó la utilidad de los modelos, esta interesada en el manejo y organización de eventos musicales. Mencionó que sería de gran utilidad para poder organizar festivales. Ya que por lo general la música creada o interpretada por los artistas invitados tienen similitudes. Estas similitudes son el género o géneros en los que pueden ser clasificados. Mencionó que una aplicación así haría el trabajo de organizar un evento mucho más sencillo. Sin embargo mencionó que agregar más géneros mejoraría su funcionalidad.

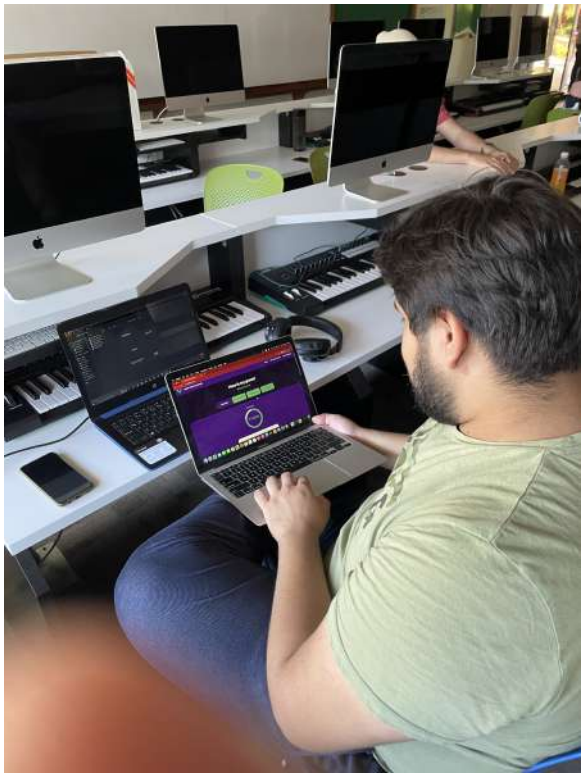


Figura 47: Estudiante de composición y producción musical (usuario 3) probando la funcionalidad de los modelos

El tercer usuario que participó en las pruebas de usuario es un estudiante que le interesa convertirse en compositor o productor de música comercial. Mencionó que la aplicación es útil para los artistas ya que permitirá que ellos puedan conseguir un contrato con un *Record Label* con mayor facilidad. Y que también permitirá que los artistas puedan realizar un *pitch* más adecuado en las plataformas de distribución y reproducción de música en línea.

Figura 48: Estudiante de composición y producción musical (usuario 4) probando la funcionalidad de los modelos

El cuarto usuario realizó muchas más pruebas que los demás. Esto permitió que la aplicación se sometiera a pruebas que necesitaban más fidelidad. Se pudo evidenciar que la música comercial es identificable como género POP, sin embargo existen problemas al diferencias las canciones comerciales acústicas o que solo llevan un instrumento. Ya que la aplicación la identifica como música clásica.

Figura 49: Estudiante de composición y producción musical (usuario 5) probando la funcionalidad de los modelos

El último usuario mencionó que le agradó la interfaz de usuario que es intuitiva y que permite que se conozca lo que se debe de hacer para identificar un género. Sin embargo mencionó que le confundió los nombres de los modelos de inteligencia artificial, ya que estos no son de su conocimiento. "No sabía que me está mostrando la aplicación, hasta que hice *scroll down*".

CAPÍTULO 17

Conclusiones

- Los modelos de redes convoluciones representan un alcance positivo para poder identificar géneros con una efectividad mayor al 75 %.
- *SoundCloud API* representa una base de datos amplia que permitirá mejorar la efectividad de los modelos para clasificar géneros musicales.
- Las redes convoluciones son funcionales en la clasificación de géneros musicales si se puede eliminar el exceso de ruido que existe entre algunos géneros musicales
- Es posible construir un sitio web público en donde se puede identificar el género musical de una pista de audio.

CAPÍTULO 18

Recomendaciones

- Realizar conexión con *labels* y miembros de la industria musical para poder enviar notificaciones automáticas a los A&R cuando una canción que se adecúa a su trabajo se sube a la plataforma. Esta acción debe realizarse sí y solo sí, el creador del contenido autoriza dicha acción.
- Realizar conexiones con productores y artistas para poder enviar notificaciones cuando canciones con géneros similares se han subido a la plataforma. Ambos podrían trabajar en una canción similar.
- Con ayuda de musicólogos clasificar cada pista dependiendo de las características teóricas que posee cada género. Debido a que en ocasiones los usuarios que suben pistas a *SoundCloud* son músicos por afición sin ningún estudio en teoría musical y mucho menos en taxonomías de géneros. Es por eso que la clasificación debe de ser realizada por expertos en el tema. Esto ayudará a eliminar el capítulo 9 de este documento ya que no se tendrá que escoger los géneros que ayudan a que la efectividad suba.
- Considerar el uso de algoritmos supervisados para la realización de este proyecto.
- Acortar el tamaño de los espectrogramas, es decir hacer un acercamiento a la imagen resultante para poder eliminar las frecuencias más altas y las más bajas. Ya que estas por lo general no se utilizan en la música moderna.
- Realizar un análisis por instrumento. Esto permitiría poder no solo hacer una identificación de género sino que con modelos de inteligencia artificial poder realizar un transformador de género, una herramienta que permita transformar una canción de un género a otro. Esta recomendación, en conjunto con la primera de esta lista ayudará a incluir más géneros a los modelos.

CAPÍTULO 19

Bibliografía

Adobe. *Color Wheel*. Adobe Color, color.adobe.com/create/color-wheel. Accedido 3 de septiembre de 2022.

Aprende machine learning. Principales algoritmos utilizados. 2017, www.aprendemachinelearning.com/principales-algoritmos-usados-en-machine- learning. Accedido 4 de septiembre de 2022.

Burgess, Richard James. The Art of Music Production: The Theory and Practice. Oxford UP, 2013.

Contreras. *Music Genre Detector Plataform*. Music Genre Detector, 2022, suulcoder.github.io/music-genre-detector. Accedido 7 de noviembre de 2022.

Elert, Glenn. *The Physics Hypertextbook*. The Physics Hypertextbook, 2021, physics.info.

Fazt. *Modelo Cliente Servidor, Explicación Simple*. YouTube, 3 de diciembre de 2017, www.youtube.com/watch?v=49zdlyLSwhQ.

Gómez, Ana M. *Redes neuronales artificiales: The Self-Organizing Maps (SOM) para el reconocimiento de patrones*. Cuadernos de Estadística Aplicada, vol. 1, n.o 1, febrero de 2015, pp. 27-38.

Google. *TensorFlow: Open source machine learning*. YouTube, 9 de noviembre de 2015, www.youtube.com/watch?v=oZikw5k_2FM.

Hochreiter, Sepp, y Jürgen Schmidhuber. *Long Short-Term Memory*. Neural Computation, vol. 9, n.o 8, The MIT Press, noviembre de 1997, pp. 1735-80. https://doi.org/10.1162/neco.1997.9.8.1735.

Ingham. *Over 60,000 tracks are now uploaded to Spotify every day. That's nearly one per second.Music Business Worldwide*. Music Business Wolrdwide, 2021, www.musicbusinessworldwide.com/over-60000-tracks-are-now-uploaded-to- spotify-daily-thats-nearly-one-per-second.

Jorda. Guía Audio digital y MIDI. 1997, www.ccapitalia.net/reso/articulos/audiodigital/pdf/09-Sintesis.pdf. Accedido 12 de septiembre de 2022.

Krumhansl, Carol L. *Rhythm and pitch in music cognition.* Psychological Bulletin, vol. 126, n.o 1, American Psychological Association, enero de 2000, pp. 159-79. https://doi.org/10.1037/0033-2909.126.1.159.

LeCun, Yann, Bernhard E. Boser, et al. *Backpropagation Applied to Handwritten Zip Code Recognition.* Neural Computation, vol. 1, n.o 4, The MIT Press, diciembre de 1989, pp. 541-51. https://doi.org/10.1162/neco.1989.1.4.541.

LeCun, Yann, Léon Bottou, et al. *Gradient-based learning applied to document recognition.* Proceedings of the IEEE, vol. 86, n.o 11, Institute of Electrical and Electronics Engineers, enero de 1998, pp. 2278-324. https://doi.org/10.1109/5.726791.

Li, et al., editores. A Comparative study on content-based Music Genre Classification. Proceedings of the 26th Annual International ACM SI- GIR Conference on Research and Development in Informaion Retrieval, 2003.

Loncomilla, editor. Deep learning: Redes convolucionales. 2016, ccc.inaoep.mx/ pgomez/deep/presentations/2016Loncomilla.pdf.

Marchal, William, et al. Statistical Techniques in Business and Economics. McGraw-Hill Education, 2014.

Matplotlib — Visualization with Python. matplotlib.org.

Mendoza-Halliday, editor. A Theory of the Musical Genre: The Three-Phase Cycle Programa De Doctorado En Música UNAM, Mexico,. 10th Inter- national Conference of Students of Systematic Musicology (SysMus17), London, UK, 2017.

Michels, Ulrich. Atlas de música. Alianza Editorial Sa, 1985.

Michie, Donald, et al. Machine Learning, Neural and Statistical Classification. Prentice Hall, 1994.

ParaBajoelectrico.com. *Frecuencia de las Notas Frecuencias musicales.* ParaBajoelectrico.com, 17 de noviembre de 2022, parabajoelectrico.com/frecuencia-notas-musicales.

Parra. Estadística y Machine Learning con R. Bookdown.org, 2006, bookdown.org/content/2274/modelo-lineal-general.html.

Pelchat, Nicholas, y Craig M. Gelowitz. *Neural Network Music Genre Classification.* Canadian journal of electrical and computer engineering, vol. 43, n.o 3, Institute of Electrical and Electronics Engineers, agosto de 2020, pp. 170-73. https://doi.org/10.1109/cjece.2020.2970144.

React – A JavaScript library for building user interfaces. React, reactjs.org.

Redux · An Introduction — Smashing Magazine. Smashing Magazine, 28 de junio de 2016, www.smashingmagazine.com/2016/06/an-introduction-to-redux.

Russell, Stuart, y Peter Norvig. Artificial Intelligence: A Modern Approach, Global Edition. 2016.

Serway, Raymond A., y John W. Jewett. Physics for Scientists and Engineers. Cengage Learning, 2013.

SoundCloud Developers. developers.soundcloud.com.

Stewart, James, et al. Calculus. Cengage Learning, 2020.

Swales, John M., y Swales. Research Genres: Explorations and Applications. Cambridge UP, 2004.

Team, Keras. Keras: the Python deep learning API. keras.io.

Tzanetakis, George, y Peter G. Cook. *Musical genre classification of audio signals.* IEEE Transactions on Speech and Audio Processing, vol. 10, n.o 5, Institute of Electrical and Electronics Engineers, noviembre de 2002, pp. 293-302. https://doi.org/10.1109/tsa.2002.800560.

Valencia-Aguirre, et al. Comparación de Métodos de Reducción de Dimensión basados en Análisis por Localidades. Tecnológicas. repositorio.itm.edu.co/bitstream/handle/20.500.12622/860/127-Manuscrito- 237-1-10-20170208.pdf.

Welcome to Flask — Flask Documentation (2.2.x). flask.palletsprojects.com/en/2.2.x.

Xu, Changsheng, et al. *Musical genre classification using support vector machines.* International Conference on Acoustics, Speech, and Signal Processing, abril de 2003, https://doi.org/10.1109/icassp.2003.1199998.

CAPÍTULO 20

Anexos

20.1. Obtención de datos

20.1.1. *SoundCloud API*

Obtención de datos de los servidores de servicios de *streaming*: Para extraer los datos se utilizarán las *APIs* de cada servicio, colocando un límite a la cantidad de canciones que se pueden extraer, para evitar sobre saturarlos datos y que una computadora de uso particular los pueda procesar.

```
from google.colab import drive
drive.mount('/content/gdrive')
```

El servicio de *streaming* más abierto del mundo es el de *soundCloud*, permitiendo que cualquier persona pueda subir música a la plataforma. Por el momento hemos encontrado dos casos que no nos son favorables, es por eso que tendremos que ignorar ambas situaciones. Para poder buscar tracks vamos a utilizar una técnica utilizada en scrappers. También tilizaremos la query del buscador de SoundCloud, avanzando en offset y moviendo el query en orden alfabético. Tampoco queremos tener tantos datos que no puedan ser procesados por el algoritmo y que una sola época tarde días en ejecutar. Es por eso que vamos a limitarnos a encontrar 1000 tracks.

20.1.2. Algoritmo de obtención de datos

Tendremos un algoritmo de búsqueda con una complejidad de $O(n^2)$, pero con la validación de la información esto será $O(n^3)$

```
import string
import os
```

```
import time

data = []
ids = []
for letter in string.ascii_lowercase:
    for number in range(0,10000):
      tracks = getRequest('https://api-v2.soundcloud.com/search?q=' + letter +
          '&client_id=' + client_id + '&limit=200&offset=' + str(number))
      if 'collection' in tracks:
        for track in tracks['collection']:
          if (
              'downloadable' in track and
              'genre' in track
              ):
            genre = track['genre']
            id = track['id']
            if id not in ids:
              downloadable = track['downloadable']
              has_downloads_lefts = track['has_downloads_left']
              if (downloadable and has_downloads_lefts):
                validGenre = getValidGenre(genre)
                if validGenre!=None:
                  downloadable_link =
                      getRequest('http://api-v2.soundcloud.com/tracks/' +
                      str(track['id']) + '/download?client_id=' + client_id)
                  while 'redirectUri' not in downloadable_link:
                    time.sleep(1)
                    print('waiting for API')
                    downloadable_link =
                        getRequest('http://api-v2.soundcloud.com/tracks/' +
                        str(track['id']) + '/download?client_id=' + client_id)
                  if 'redirectUri' in downloadable_link:
                    track['genre'] = validGenre
                    path = "/content/gdrive/My Drive/Tesis/" +
                        (track["genre"]).replace(' ','_').replace('/','-')
                    exists = os.path.exists(path)
                    if not exists:
                      os.makedirs(path)
                    urllib.request.urlretrieve(downloadable_link['redirectUri'],
                        "/content/gdrive/My Drive/Tesis/" +
                        (track["genre"]).replace(' ','_').replace('/','-') + "/" +
                        str(track['id']) + ".mp3")
                    data.append(track)
                    ids.append(id)
```

20.1.3. Escritura de los datos

```
import csv
keys = data[0].keys()

with open('/content/gdrive/My Drive/Tesis/data.csv', 'a', newline='') as
    output_file:
    dict_writer = csv.DictWriter(output_file, keys)
    #dict_writer.writeheader()
    dict_writer.writerows(data)
```

20.2. Función de transformación a espectrograma

```
def song_to_spectogram(path, savepath):
    print('working on ' + path)
    data, sr = librosa.load(path, sr=44100) #Amplitude vs time
    data_by_frequency = librosa.stft(data)
    Xdb = librosa.amplitude_to_db(abs(data_by_frequency))
    librosa.display.specshow(Xdb, sr=sr, x_axis='time', y_axis='log',
        hop_length=1024, cmap=cm.Spectral)
    plt.axis('off')
    plt.savefig(savepath, bbox_inches='tight', pad_inches=0)
    print('saved in ' + savepath)
```

20.3. Carga de datos

20.3.1. Limpieza de clases

```
spectograms_directories = []
for (dirpath, dirnames, filenames) in walk('./Data/spectograms/'):
    spectograms_directories = list(filter(
        lambda spectogram: 'png' in spectogram,
        [dirpath + filename for filename in filenames]))

valid_genres = [
    'classical',
    'metal',
    'pop',
    'country'
]
spectograms = []
x = []
classNames = []
for spectogram in spectograms_directories:
    genre = spectogram.split('.')[1].split('/')[3]
    genre = ''.join([i for i in genre if not i.isdigit()])
```

```
    if genre=='blues':
        genre = 'r&b'
    if genre=='disco':
        genre = 'EDM'
    if genre in valid_genres:
        img_array = cv2.imread(spectogram)[...,::-1]
        resized_arr = cv2.resize(img_array, (img_size, img_size))
        spectograms.append(tf.keras.preprocessing.image.load_img(spectogram))
        x.append(resized_arr)
        classNames.append(genre)
```

20.3.2. Preparación para el modelo

```
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(classNames)

size = len(x)

train_images = x[int(0.95*size) - 1:-1]
test_images = x[0:int(0.95*size) - 1]
train_labels = y[int(0.95*size) - 1:-1]
test_labels = y[0:int(0.95*size) - 1]

train_images = np.array(train_images) / 255
test_images = np.array(test_images) / 255
train_labels = np.array(train_labels)
test_labels = np.array(test_labels)


train_images.reshape(-1, img_size, img_size, 1)
test_images.reshape(-1, img_size, img_size, 1)
```

20.4. Modelo *Feed Forward*

```
model = models.Sequential()
model.add(layers.Conv2D(filters=64, kernel_size=(3, 3), activation='relu',
    input_shape=(img_size,img_size,3)))
model.add(layers.Flatten())
model.add(layers.Dense(units=16, activation='relu'))
model.add(layers.Dense(len(valid_genres), activation = 'softmax'))
model.summary()
```

20.5. Modelo *Let Net*

```
model = models.Sequential()
model.add(layers.Conv2D(filters=6, kernel_size=(3, 3), activation='relu',
    input_shape=(img_size,img_size,3)))
model.add(layers.AveragePooling2D())
model.add(layers.Conv2D(filters=16, kernel_size=(3, 3), activation='relu'))
model.add(layers.AveragePooling2D())
model.add(layers.Flatten())
model.add(layers.Dense(units=128, activation='relu'))
model.add(layers.Dense(len(valid_genres), activation = 'softmax'))
model.summary()
```

20.6. Modelo red convolucional neuronal

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(img_size,
    img_size, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(len(valid_genres), activation='softmax'))
model.summary()
```

20.7. Modelo *LSTM*

```
model = models.Sequential()
model.add(layers.Conv2D(filters=6, kernel_size=(3, 3), activation='relu',
    input_shape=(img_size,img_size,3)))
model.add(layers.AveragePooling2D())
model.add(layers.Conv2D(filters=16, kernel_size=(3, 3), activation='relu'))
model.add(layers.AveragePooling2D())
model.add(layers.Flatten())
model.add(layers.Reshape((-1, 36864)))
model.add(layers.LSTM(128))
model.add(layers.Dense(len(valid_genres), activation = 'softmax'))
model.summary()
```

20.8. Compilación y entrenamiento de los modelos

```
model.compile(optimizer='adam',
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
      metrics=['accuracy', 'mse'])

history = model.fit(train_images, train_labels, epochs=40,
    validation_data=(test_images, test_labels))
```

20.9. Predicción con los modelos

20.9.1. Audio a espectrograma

```
def song_to_spectogram(path):
    data, sr = librosa.load(path, sr=44100) #Amplitude vs time
    data_by_frequency = librosa.stft(data)
    Xdb = librosa.amplitude_to_db(abs(data_by_frequency))
    librosa.display.specshow(Xdb, sr=sr, x_axis='time', y_axis='log',
        hop_length=1024, cmap=cm.Spectral)
    plt.axis('off')
    plt.savefig("./temporal.png", bbox_inches='tight', pad_inches=0)
```

20.9.2. Predicción de clases

```
def predict_with_model():
    img_array = cv2.imread("./temporal.png")[...,::-1]
    image = cv2.resize(img_array, (img_size, img_size))
    tests_files = [image]
    tests_files = np.array(tests_files) / 255
    tests_files.reshape(-1, img_size, img_size, 1)
    prediction = model.predict_classes(tests_files)
    return(label_encoder.inverse_transform(prediction))
```

20.9.3. Predicción con el modelo

```
def predict(path):
    song_to_spectogram(path)
    return predict_with_model()
```

20.10. *Backend*

```
from flask import Flask, jsonify, request
from flask_cors import CORS, cross_origin
```

```
from matplotlib import cm
import os
from tensorflow import keras
import cv2
import librosa
import librosa.display
import matplotlib
import numpy as np
import tensorflow as tf
import uuid


matplotlib.use('Agg')
img_size = 200
app = Flask(__name__)
CORS(app)
model1 = None
model2 = None
model3 = None
model4 = None

model1_name_ = 'CNN Model'
model2_name_ = 'LetNet Model'
model3_name_ = 'LSTM Model'
model4_name_ = 'Feed-Forward Model'

model1_name = './Models/CNN/'
model2_name = './Models/LETNET/'
model3_name = './Models/LTSM/'
model4_name = './Models/FeedForward/'

model1_metrics = './Models/CNN.txt'
model2_metrics = './Models/LETNET.txt'
model3_metrics = './Models/LTSM.txt'
model4_metrics = './Models/FeedForward.txt'

def load_models():
    global model1
    model1 = tf.keras.models.load_model(model1_name)

    global model2
    model2 = tf.keras.models.load_model(model2_name)

    global model3
    model3 = tf.keras.models.load_model(model3_name)

    global model4
    model4 = tf.keras.models.load_model(model4_name)

def song_to_spectogram(path, savepath):
    data, sr = librosa.load(path, sr=44100) #Amplitude vs time
    data_by_frequency = librosa.stft(data)
    Xdb = librosa.amplitude_to_db(abs(data_by_frequency))
```

```
    librosa.display.specshow(Xdb, sr=sr, x_axis='time', y_axis='log',
        hop_length=1024, cmap=cm.Spectral)
    matplotlib.pyplot.axis('off')
    matplotlib.pyplot.savefig(savepath, bbox_inches='tight', pad_inches=0)

def predict_with_CNN_model(path, genres, model):
    img_array = cv2.imread(path)[...,::-1]
    image = cv2.resize(img_array, (img_size, img_size))
    tests_files = [image]
    tests_files = np.array(tests_files) / 255
    tests_files.reshape(-1, img_size, img_size, 1)
    prediction = model.predict(tests_files)
    result = {}
    for i in range(0, len(prediction[0])):
        result[genres[i]] = prediction[0][i]
    return(result)

def predict_with_model1(path):
    return predict_with_CNN_model(path, ['classical', 'country','metal','pop'],
        model1)

def predict_with_model2(path):
    return predict_with_CNN_model(path, ['classical', 'country','metal','pop'],
        model2)

def predict_with_model3(path):
    return predict_with_CNN_model(path, ['classical', 'country','metal','pop'],
        model3)

def predict_with_model4(path):
    return predict_with_CNN_model(path, ['classical', 'country','metal','pop'],
        model4)


def serialize(results):
    new_results = {}
    for result in results:
        new_results[result] = str(results[result])
    return new_results

def get_current_accuracy(model_metrics_path):
    metrics = {}
    with open(model_metrics_path) as f:
        lines = f.readlines()
        for line in lines:
            if ('accuracy' in line):
                metrics['accuracy'] = line.split(" : ")[-1]
            if ('loss' in line):
                metrics['loss'] = line.split(" : ")[-1]
    return metrics


@app.route('/', methods=['POST'])
@cross_origin()
```

```
def index():
    # Initiate returned structure
    data = {
        "success": False
        }

    name = uuid.uuid4()
    audio_name = './Files/' + str(name) +'.wav'
    spectogram_name = './Files/' + str(name) +'.png'
    if request.method == "POST":
        if('audio_file' in request.files):
            #Get the audio and save in a temporal file
            request.files['audio_file'].save(audio_name)

            #Generate the spectogram
            song_to_spectogram(audio_name, spectogram_name)

            #Predict with CNN Model
            results1 = predict_with_model1(spectogram_name)

            #Predict with Let Net Model
            results2 = predict_with_model2(spectogram_name)

            #Predict with LSTM Model
            results3 = predict_with_model3(spectogram_name)

            #Predict with Feed Forward Model
            results4 = predict_with_model4(spectogram_name)


            data = {}
            data[model1_name_] = {}
            data[model1_name_]['data'] = serialize(results1)
            data[model1_name_]['metrics'] = get_current_accuracy(model1_metrics)

            data[model2_name_] = {}
            data[model2_name_]['data'] = serialize(results2)
            data[model2_name_]['metrics'] = get_current_accuracy(model2_metrics)


            data[model3_name_] = {}
            data[model3_name_]['data'] = serialize(results3)
            data[model3_name_]['metrics'] = get_current_accuracy(model3_metrics)

            data[model4_name_] = {}
            data[model4_name_]['data'] = serialize(results4)
            data[model4_name_]['metrics'] = get_current_accuracy(model4_metrics)

            os.remove(audio_name)
            os.remove(spectogram_name)
            # return the data dictionary as a JSON response
    return jsonify(data)

if __name__ == "__main__":
```

```
print(("* Loading Keras model and Flask starting server..."
    "please wait until server has fully started"))
load_models()
app.run(host='localhost', port=8000)
```