Huella digital de sonido para reconocimiento musical

Juan B. (A01742342), José Y. (A01740285)

23 de noviembre de 2022

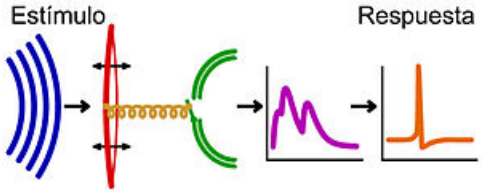
1. Introducción

El reconocimiento de voz o de audio tiene múltiples aplicaciones, como la detección de audios con copyright, desbloqueo de sistemas mediante la voz, o simplemente reconocimiento de canciones que para uno pueden ser desconocidas. Esto último lo hace la famosa aplicación Shazam, la cual fue desarrollada por Avery Wang en el año 2003. Sin embargo, podemos adelantar que el elemento fundamental para la correcta funcionalidad del algoritmo radica en la transformada discreta de Fourier.

2. El sonido

2.1. ¿Cómo se produce?

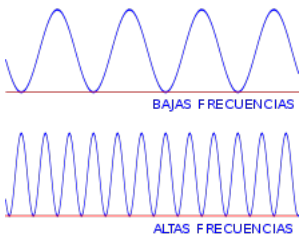
El sonido como fenómeno físico nos interesa para comprender lo que hacemos cuando manipulamos sonido digital. Para que haya sonido es preciso que un cuerpo físico vibre, que haya un soporte físico que propague esas vibraciones y, por último, que las mismas sean capaces de impresionar los nervios auditivos del oído, es decir, no toda vibración del aire puede ser considerada sonido: sólo aquella capaz de estimular el sentido auditivo.



El sonido cuenta con 2 componentes principales: frecuencia y amplitud.

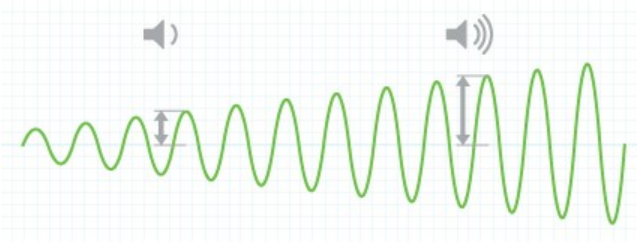
2.2. Frecuencia

La frecuencia es la cantidad de oscilaciones por unidad de tiempo, determina si el sonido es agudo o grave. A mayor frecuencia, el tono del sonido es más agudo; y a menor es grave. Se suele medir en hercios (Hz), y el oído humano es capaz de percibir sonidos entre los 20 y 20,000 hercios.



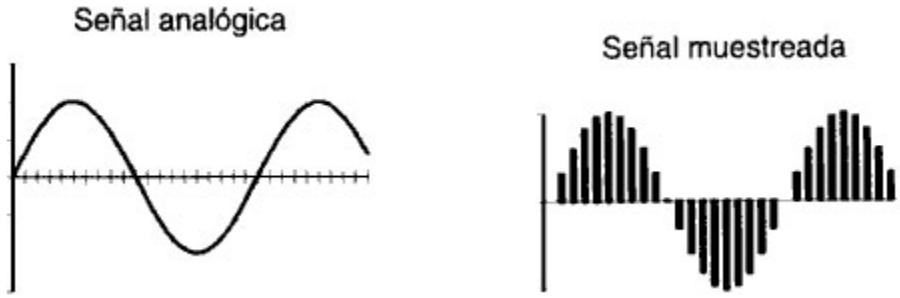
2.3. Amplitud

La amplitud se refiere a la altura de la onda, y esta determina la intensidad o volumen del sonido. La amplitud cero es equivalente al silencio, y mientras más aumenta, los sonidos son más fuertes.

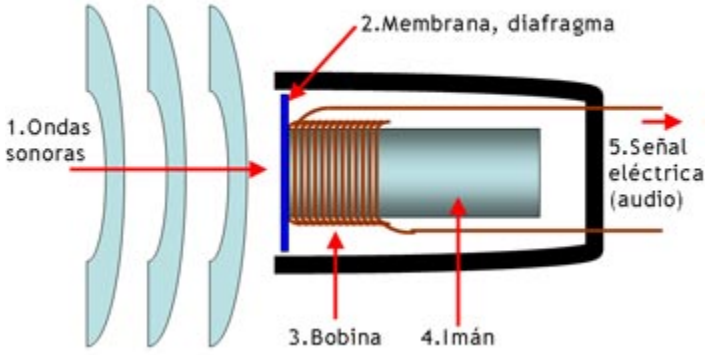


2.4. Principio de grabado (sonido analógico)

Los dispositivos de grabación imitan el proceso de escucha humano mediante la presión de la onda de sonido para convertirla en una señal eléctrica. Para esto es necesario establecer el parámetro de frecuencia de muestreo, el cual en la gran mayoría de discos compactos y archivos de audio es de 44,100 Hz. La razón por la que se utiliza este valor es que el teorema de Nyquist-Shannon, que nos dice que para que el ser humano pueda captar todas las frecuencias la muestra de señal debe equivaler a una frecuencia que duplique el rango auditivo humano, que detecta frecuencias de entre 20 Hz y 20,000 Hz.



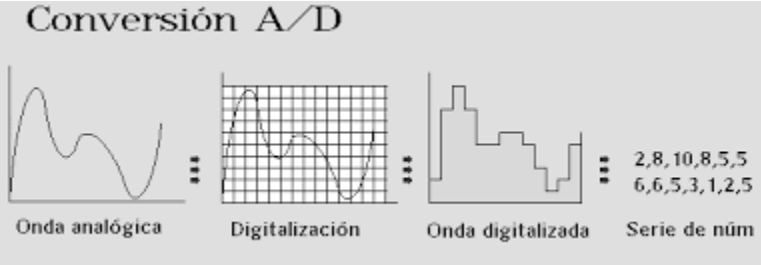
Una onda de sonido real en el aire es una señal de presión. El instrumento que transforma las ondas sonoras en sonido analógico se llama micrófono. Este cumple la función de transformar las ondas sonoras, las variaciones de presión en el aire, en pequeños impulsos eléctricos, proporcionales y equivalentes a los sonidos. Esta señal puede ser almacenada en diversos soportes, amplificada, propalada, transmitida, etc.



2.5. Digitalización del sonido

El sonido analógico varía en forma ininterrumpida y continua. Cualquier fragmento puede ser dividido en fragmentos aún más pequeños, es decir, es infinitamente divisible, pudiendo cada uno de esos fragmentos tener valores diferentes.

Las computadoras trabajan solo con datos finitos, por lo que se aplica un proceso de digitalización a el sonido analógico. Se toman muestras del valor analógico del sonido por trozos con respecto al tiempo. El conjunto de las muestras produce una réplica de la curva analógica, y mientras más muestras se tengan, mayor será la fidelidad del registro.



3. La importancia de Fourier

3.1. La transformada de Fourier

Las funciones de Schwartz son aquellas funciones definidas en R que son infinitamente diferenciables y rápidamente convergentes a cero. Podríamos definirla más formalmente como:

Una función f se llama función de Schwartz si $f \in C^\infty(R)$ y $\lim_{|x|\to\infty} \left(1+x^2\right)^k f^{(p)} = 0$, para todo par de enteros no negativos kyp. Equivalentemente, f es una función de Schwartz si $\lim_{|x|\to\infty} P(x)f^{(n)}(x) = 0$ para todo entero no negativo n y para todo polinomio $P(x)$.

El conjunto formado por todas las funciones de Schwartz se denota por $S(R)$. Evidentemente $S(\boldsymbol{R}) \subset L^1(\boldsymbol{R}) \cap L^2(\boldsymbol{R})$. Puede probarse fácilmente que $S(\boldsymbol{R})$ es denso en $L^1(\boldsymbol{R})$ y $L^2(\boldsymbol{R})$.

Para $f, g \in S(R)$, definimos

$$\hat{f}(\gamma) = \int f(x)e^{-2\pi i n x\gamma}dx$$

$$\check{f}(x) = \int f(\gamma)e^{2\pi i\gamma x}d\gamma$$

La primera ecuación es llamada la transformada de Fourier de f, y la segunda ecuación la transformada inversa de Fourier de f.

La importancia de la transformada discreta de Fourier radica en su uso para descomponer las señales periódicas en senos y cosenos de diferentes frecuencias y amplitudes.

Y como alternativa a la transformada discreta, se utiliza el algoritmo de la transformada rápida de Fourier, método que realiza los mismos cálculos que la transformada discreta, pero debido a su recursividad, los realiza de manera más rápida.

3.1.1. La transformada rápida de Fourier

La transformada rápida de Fourier (FFT) es un algoritmo que reduce el tiempo de cálculo de n^2 pasos a $n\log_2(n)$. El único requisito es que el número de puntos en la serie tiene que ser una potencia de 2 (2^n puntos), por ejemplo 32, 1024, 4096, etc.

$$\left.\begin{array}{l} X(k) = \sum_{j=1}^{N} x(j)\omega_N^{(j-1)(k-1)} \\ x(j) = \frac{1}{N}\sum_{k=1}^{N} x(k)\omega_N^{-(j-1)(k-1)} \end{array}\right\} \omega_N = \exp\left(-\frac{2\pi i}{N}\right)$$

3.2. El teorema de Fourier

Expresa que toda función periódica de frecuencia ω_0, puede expresarse como la suma infinita de funciones seno o coseno que son múltiplos enteros n de ω_0. A esta se le denomina frecuencia fundamental y a cada término de seno o coseno se le conoce como armónica. La serie de Fourier hace

esta representación en forma de sumatoria.

$$f(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} \left[a_n \cos\left(n\omega_0 t\right) + b_n \sin\left(n\omega_0 t\right) \right]$$

Análogo a la serie, la transformada de Fourier descompone una señal periódica en senos y cosenos de diferentes frecuencias y amplitudes.

En las aplicaciones de ingenieria y tratamiento de señales, resulta más práctico considerar el proceso de manera discreta y no continua, ya que los sistemas de adquisición de datos no pueden obtener ni analizar la totalidad de la información.

$$x[k] = \sum_{n=0}^{N-1} x[n] e^{-\left(\frac{2\pi n k}{N}\right) j}$$

Donde:

$N =$ Número de muestras

$n =$ Enésima muestra original

$k =$ k-ésimo término de la transformada discreta de Fourier

3.3. Síntesis aditiva

Para entender el método de la sintesis aditiva, se considera el hecho de que todos los sonidos son una suma de distintos tonos sinusoidales y armónicos.

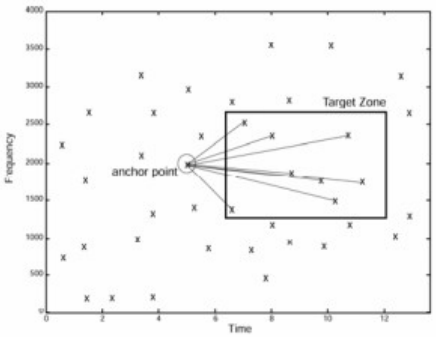
En la sintesis aditiva, partirás de cero y crearás un sonido mediante la combinación de varias ondas sinusoidales de distintos niveles y frecuencias. Según vayas combinando ondas sinusoidales, estas comenzarán a generar armónicos adicionales. En la mayoria de los sintetizadores aditivos, cada conjunto de ondas sinusoidales se considera y se utiliza de forma parecida a un oscilador.

$$f'_N(t) = \sum_{n=-N}^{N} c_n e^{i\omega_0 n t}$$

3.4. Huella digital de sonido

El proceso de huella de audio consiste en identificar coincidencias entre un audio externo y otro audio que se encuentre en una base de datos. Para esto se deben seleccionar los máximos locales del espectrograma obtenido tras haber aplicado un análisis de transformada de Fourier y síntesis aditiva al audio externo. Al haber seleccionado los puntos, se obtienen las relaciones entre cada uno de estos. Algunas ventajas de este método para el reconocimiento de música es que las comparaciones suelen ser acertadas y las búsquedas son rápidas, sin embargo, se cuenta con probabilidades de error, condiciones de ruido y tiempo de espera.

El espectrograma es una representación visual de las variaciones de tiempo y frecuencia que tiene una muestra de sonido. Pero, para poder obtener este espectrograma necesitamos realizar una transformada discreta de Fourier a nuestra señal de audio.



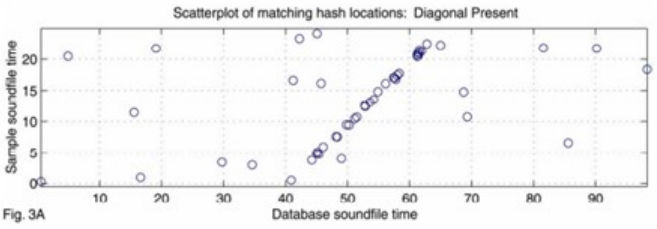


Fig. 3A

Cada relación entre dos puntos conforma un dato. Los datos están conformados por la frecuencia del primer punto, la frecuencia del segundo punto, y la distancia temporal entre ambos. Con estos datos, se puede comparar el audio externo con uno que se tenga en la base de datos.

En el eje horizontal de esta gráfica está la línea temporal del audio de la base de datos, mientras que en el vertical se encuentra la línea temporal de la muestra del audio grabado. Si las coincidencias forman una línea de tendencia parecida a la que se ve entre los valores 40 y 60 del eje x, esto representa un emparejamiento.

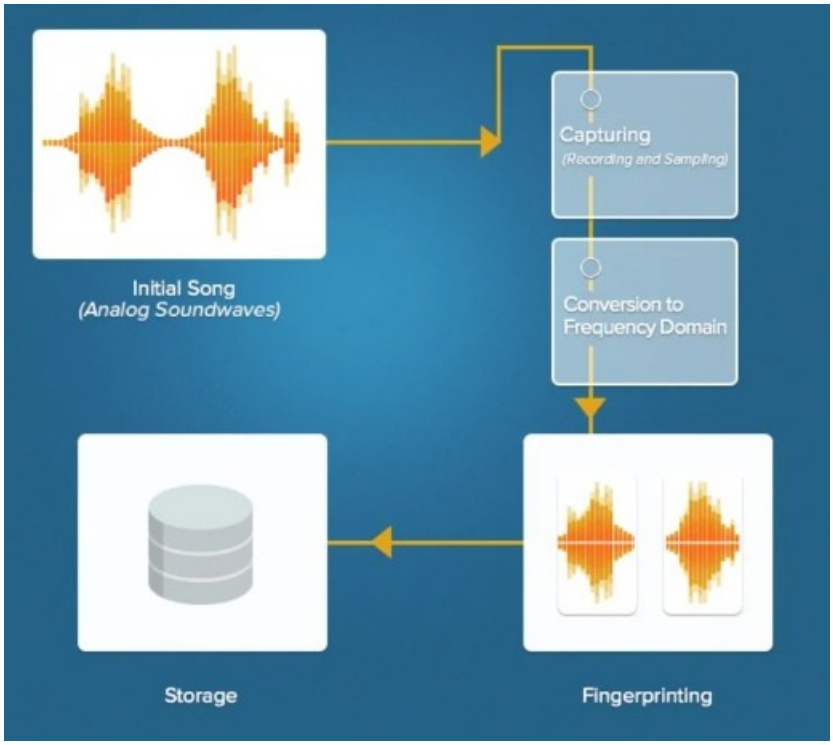
4. Proceso de reconocimiento musical

Primero, se captura o graba algunos segundos de la canción a reconocer. Esto es con el proposito de tener un audio de muestra para el match que se realizará al final.

Segundo, se aplica la transformada discreta de Fourier a los valores obtenidos mediante la división de la curva análoga (proceso parecido a las sumas de Riemann). Con esto, se obtiene el espectrograma del sample.

Tercero, ya obtenido el espectrograma del sample se utiliza el método de la huella digital para comparar los máximos locales del espectrograma del sample, con los espectrogramas de las canciones guardadas en la base de datos.

Por último, se aplica la comparación, y aquel espectrograma que alinie sus máximos con los máximos del sample, es el match con más coincidencias, y por lo tanto, la canción que estamos buscando.



5. Programa de reconocimiento musical

Link al programa: https://drive.google.com/drive/folders/1AwBtphIdOLKMPy4nwk20-B_In7QJreW0

Las librerías necesarias para ejecutar el código son:

```
[119] import numpy as np
      import matplotlib.pyplot as plt
      import scipy
      from scipy import fft, signal
      from scipy.io.wavfile import read
      from scipy.fft import fft, fftfreq
      import glob
      from typing import List, Dict, Tuple
      import pickle
```

La siguiente función sirve para crear el mapa de constelaciones, es decir, los puntos más relevantes del espectrograma de la canción en donde el eje horizontal representa el tiempo y el vertical la frecuencia. La función comienza obteniendo parámetros para dividir la fracción en fragmentos o ventanas, esto se hace con el objetivo de que los cambios de sonido a lo largo de la canción, los cuales hacen que el espectro de frecuencias cambie, no afecte el emparejamiento. En este caso se opta por una ventana de muestreo de 0.5 segundo, lo que significa que se tomarán muestras cada 0.5 segundos. Además, también se opta por encontrar 15 picos por fragmento. Tras dividir la función en ventanas de forma equivalente, se obtiene la transformada rápida de Fourier, la cual es una matriz de Frecuencias x Tiempos. Para cada ventana se obtiene la parte positiva y se buscan los picos más prominentes, estos se manipulan con los parámetros de prominencia y distancia. Para obtenerlos, se utiliza la función argpartition la cual sirve para encontrar los n valores más pequeños dentro de un arreglo. Posteriormente se agregan al mapa de constelaciones los picos con sus respectivas frecuencia y tiempo.

```
[120] def crear_constelacion(audio, Fs):

          ventana_muestreo_longitud = 0.5
          ventana_muestreo_numero = int(ventana_muestreo_longitud * Fs)
          ventana_muestreo_numero += ventana_muestreo_numero % 2
          num_picos = 15

          cantidad_completar = ventana_muestreo_numero - audio.size % ventana_muestreo_numero
          cancion_entrada = np.pad(audio, (0, cantidad_completar))

          frecuencias, tiempos, stft = signal.stft(
              cancion_entrada, Fs, nperseg=ventana_muestreo_numero, nfft=ventana_muestreo_numero, return_onesided=True)

          mapa_constelacion = []
          for tiempo_indice, ventana in enumerate(stft.T):
              espectro = abs(ventana)
              picos, props = signal.find_peaks(espectro, prominence=0, distance=200)
              n_picos = min(num_picos, len(picos))
              picos_prominentes = np.argpartition(props["prominences"], -n_picos)[-n_picos:]
              for pico in picos[picos_prominentes]:
                  frecuencia = frecuencias[pico]
                  mapa_constelacion.append([tiempo_indice, frecuencia])
          return mapa_constelacion
```

La siguiente función sirve para crear los hashes en base al mapa de constelación, un hash en Python es un dato entero que, en este caso, encripta la distancia entre dos picos. En primer lugar, se deben establecer los parámetros de frecuencia para el tipo de archivo, que en este caso es wav. Ahora se procede a iterar cada punto del mapa de constelación con los siguientes 100 puntos en el tiempo, tomando en cuenta únicamente valores con una diferencia mínima de tiempo. Posteriormente las dos frecuencias son convertidas en datos enteros de 10 bits y la distancia entre estas a un entero de 12 bits. Esto último da como resultado un hash de 32 bits el cual es almacenado en el diccionario creado al principio de la función.

```
[108] def crear_hashes(mapa_constelacion, cancion_id=None):

          hashes = {}
          frecuencia_superior = 23_000
          frequencia_bits = 10

          for idx, (tiempo, frec) in enumerate(mapa_constelacion):

              for otro_tiempo, otra_frec in mapa_constelacion[idx : idx + 100]:
                  dif_tiempo = otro_tiempo - tiempo

                  if dif_tiempo <= 1 or dif_tiempo > 10:
                      continue

                  frec_conv = frec / frecuencia_superior * (2 ** frequencia_bits)
                  otra_frec_conv = otra_frec / frecuencia_superior * (2 ** frequencia_bits)
                  hash = int(frec_conv) | (int(otra_frec_conv) << 10) | (int(dif_tiempo) << 20)

                  hashes[hash] = (tiempo, cancion_id)
          return hashes
```

Esta función sirve para determinar cuál canción de la base de datos tiene la mayor cantidad de hashes iguales que la grabación ingresada. El parámetro son los hashes de la grabación, los cuales son iterados a lo largo de todos los hashes de las canciones de la base de datos y se almacenan los coincidentes. Para obtener las puntuaciones de iteran los hashes iguales de cada canción, obteniendo los desfases de tiempo entre las coincidencias, obteniendo así el pico más alto en el tiempo de ambas grabaciones.

```python
def puntuacion_canciones(hashes):
    emparejamientos_por_cancion = {}
    for hash, (tiempo_muestreo, _) in hashes.items():
        if hash in base_de_datos:
            emparejamiento = base_de_datos[hash]
            for tiempo_ref, indice_cancion in emparejamiento:
                if indice_cancion not in emparejamientos_por_cancion:
                    emparejamientos_por_cancion[indice_cancion] = []
                emparejamientos_por_cancion[indice_cancion].append((hash, tiempo_muestreo, tiempo_ref))

    puntuaciones = {}
    for indice_cancion, matches in emparejamientos_por_cancion.items():
        cancion_puntuaciones_offset = {}
        for hash, tiempo_muestreo, tiempo_ref in matches:
            delta = tiempo_ref - tiempo_muestreo
            if delta not in cancion_puntuaciones_offset:
                cancion_puntuaciones_offset[delta] = 0
            cancion_puntuaciones_offset[delta] += 1

        max = (0, 0)
        for offset, score in cancion_puntuaciones_offset.items():
            if score > max[1]:
                max = (offset, score)

        puntuaciones[indice_cancion] = max

    puntuaciones = list(sorted(puntuaciones.items(), key=lambda x: x[1][1], reverse=True))

    return puntuaciones
```

Para crear la base de datos primero se obtiene la dirección de la carpeta en donde se encuentra la base de datos. Se recorren todos los archivos wav en la carpeta, obteniendo sus respectivos mapa de constelación y hashes. Por una parte, las canciones y sus índices (por orden alfabético) se almacenan en un diccionario, mientras que los hashes se almacenan en otro. Posteriormente, tanto los hashes como la lista de canciones son guardados como pickles, es decir, cadenas de bytes.

```python
songs = glob.glob('/content/drive/MyDrive/Shazam/data/*.wav')
cancion_indice = {}
base_de_datos: Dict[int, List[Tuple[int, int]]] = {}

for indice, archivo in enumerate(sorted(songs)):
    cancion_indice[indice] = archivo
    Fs, audio_entrada = read(archivo)
    audio_entrada = audio_entrada[:,0]
    constelacion = crear_constelacion(audio_entrada, Fs)
    hashes = crear_hashes(constelacion, indice)

    for hash, par_indice_tiempo in hashes.items():
        if hash not in base_de_datos:
            base_de_datos[hash] = []
        base_de_datos[hash].append(par_indice_tiempo)

with open("base_de_datos.pickle", 'wb') as db:
    pickle.dump(base_de_datos, db, pickle.HIGHEST_PROTOCOL)
with open("cancion_indice.pickle", 'wb') as songs:
    pickle.dump(cancion_indice, songs, pickle.HIGHEST_PROTOCOL)
```

Antes de leer nuestra grabación, cargamos nuestros archivos con los hashes y la lista de canciones. Después de leer la grabación se obtiene su mapa de constelación y sus hashes. Por último, se compara la cantidad de hashes iguales con cada canción de la base de datos y se obtienen las puntuaciones.

```python
[124] base_de_datos = pickle.load(open('base_de_datos.pickle', 'rb'))
    direccion_cancion = pickle.load(open("cancion_indice.pickle", "rb"))
    Fs, audio_entrada = read("/content/drive/MyDrive/Shazam/pruebas/prueba1.wav")
    audio_entrada = audio_entrada[:,0]
    constelacion = crear_constelacion(audio_entrada, Fs)
    hashes = crear_hashes(constelacion, None)
    puntuaciones = puntuacion_canciones(hashes)

    for cancion_indice, puntuacion in puntuaciones:
        print(f"{direccion_cancion[cancion_indice]}=: Score of {puntuacion[1]} at {puntuacion[0]}")
```

Ejemplo del correcto funcionamiento del programa.

6. Conclusión

Nuevamente recalcamos la importancia y versatilidad del teorema de Fourier, y como este nos ayudó a crear un programa capaz de comparar grandes cantidades de huellas digitales para encontrar las más "parecidasz decirnos qué canción estamos escuchando. Este sería solo uno de los tantos usos de Fourier en la ingeniería y las ciencias.

Referencias

Cetta, P. (2005). ANÁLISIS ESPECTRAL APLICADO A LA MÚSICA. https://repositorio.uca.edu.ar/bitstream/123456789/9191/1/analisis-espectral-aplicado-musica.pdf

González, G. (1997). Series de Fourier, Transformadas de Fourier y Aplicaciones. https://www2.math.ethz.ch/EMIS/journals/DM/v5/art6.pdf

Pacheco, N. (2020). ANALISIS Y DISEÑO DE UN ALGORITMO DE RECONOCIMIENTO MUSICAL A TRAVES DE LA HUELLA DIGITAL DE SONIDO. http://repositorio.ug.edu.ec/bitstream/redug/51597/1/ARELLANO%20PACHECO%20NICOLAS%20IVAN.pdf

Osorio, J., Chaves, J. Knott, A. (2012). Aproximación a la síntesis de la música a través del análisis de Fourier. https://Dialnet-AproximacionALaSintesisDeLaMusicaATravesDelAnalisi-4271767.pdf

Asinten, J. (s. f.). El Sonido. https://d1wqtxts1xzle7.cloudfront.net/34957678/sonido-with-cover-page-v2.pdf?Expires=1669244116&Signature=VbH7PiLfYRXfYaaLfuRQtCcRN7~yWIYqd5gXEQ0QCnxu9BHA7tUYWCR7qvrQIjY4m9~u_&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA

Jovanovic, J. (2015, 2 febrero). Shazam! Reconocimiento de Algoritmos de Música, Huellas Dactilares y Procesamiento. Toptal Engineering Blog. https://www.toptal.com/algorithms/shazam-reconocimiento-de-alg

How Shazam Works - An explanation in Python. (s. f.). Michael Strauss. https://michaelstrauss.dev/shazam-in-python/