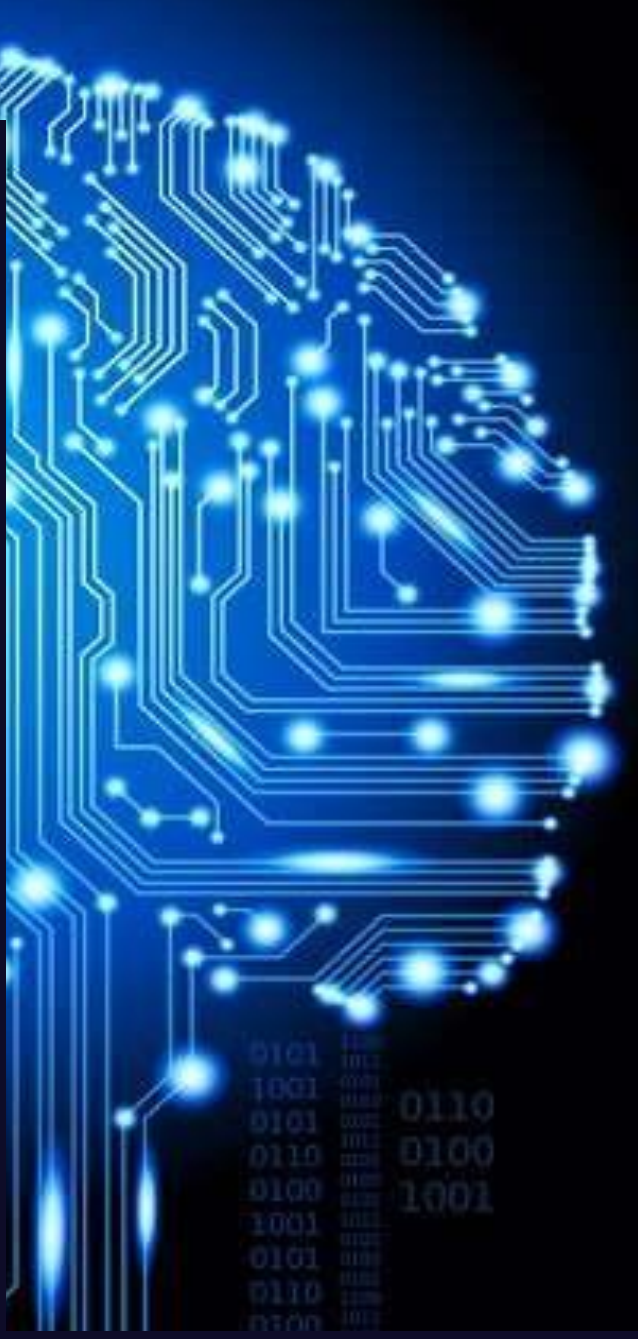
Music genre classification

University of Tehran
School of Mathematics, Statistics and Computer Science



FALL 2020

HOSSEIN ZANDINEJAD
SAMAN ARZAGHI
RAYAN FORSAT

Music genre classification

With Neural Networks

Introduction:

Undoubtedly, music and related fields have attracted a lot of attention in recent decades and it has been well-received. Following this, many genres have emerged and in some cases it is very difficult to distinguish these genres from each other. On the other hand, due to the expansion of neural networks and its popularity among programmers and the development of artificial intelligence, many tasks that are difficult or even impossible in some cases, can be left to machines.

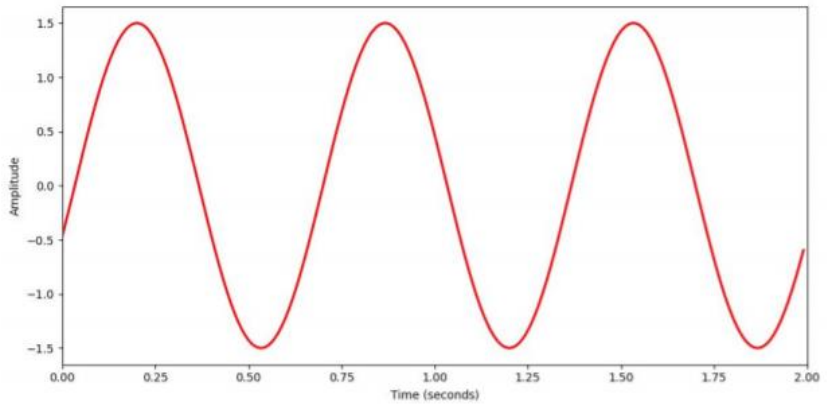
As a result of the problem we mentioned, which is the difficulty of distinguishing certain genres from one another, we decided to form a tool using python to help us classify the desired audio.

Target:

Now to form our tool, we need to convert an analog audio to its binary form, so that the computer can store or process the audio, then by the use of machine learning and neural networks we can analyze the data and identify the desired music genre.

Audio Pre-Processing for DL Projects

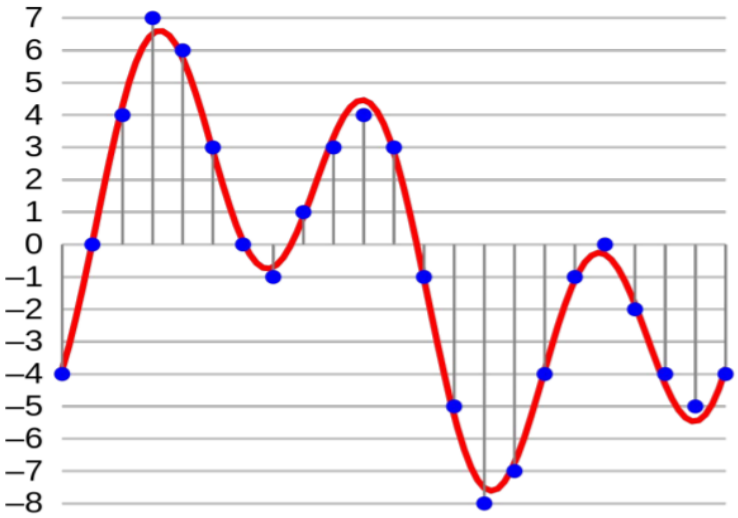
Well, sound produce when an object vibrates and those vibrations determine the oscillation of air molecules which basically creates an alternation of air pressure and this high pressure alternated with low pressure causes a wave and we can represent this wave using wave form.



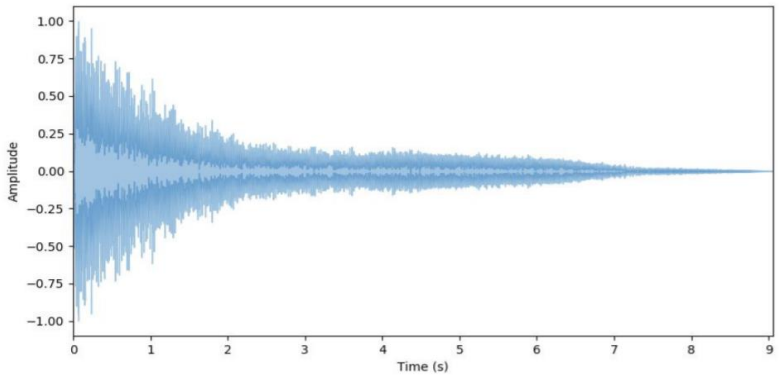
When we talk about sounds around us like our own voice or a sound of a piano playing, we are talking continuous waveforms and they are analog, but obviously, we can't really store analog waveforms we need an away of digitalizing them, and for doing that we can use analog-digital conversion (ADC) process and for that we must perform two steps:

1-Sampling → Sample the signal at specific time intervals.

2-Quantization → Quantize the amplitude given and represent with a limited number of bits. (note that more bits we store the amplitude and-the better quality of sound will be)



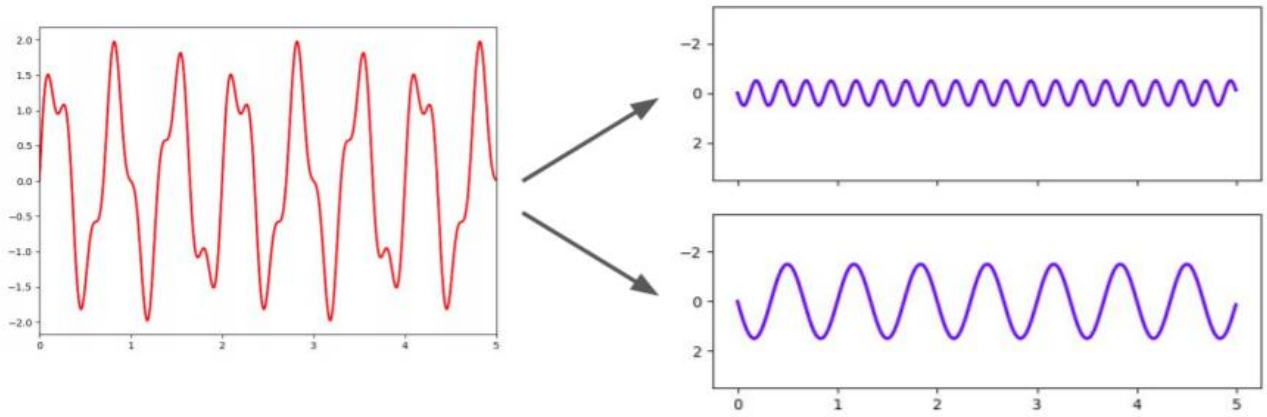
Ok, now let's take a look at piano key sound waves.



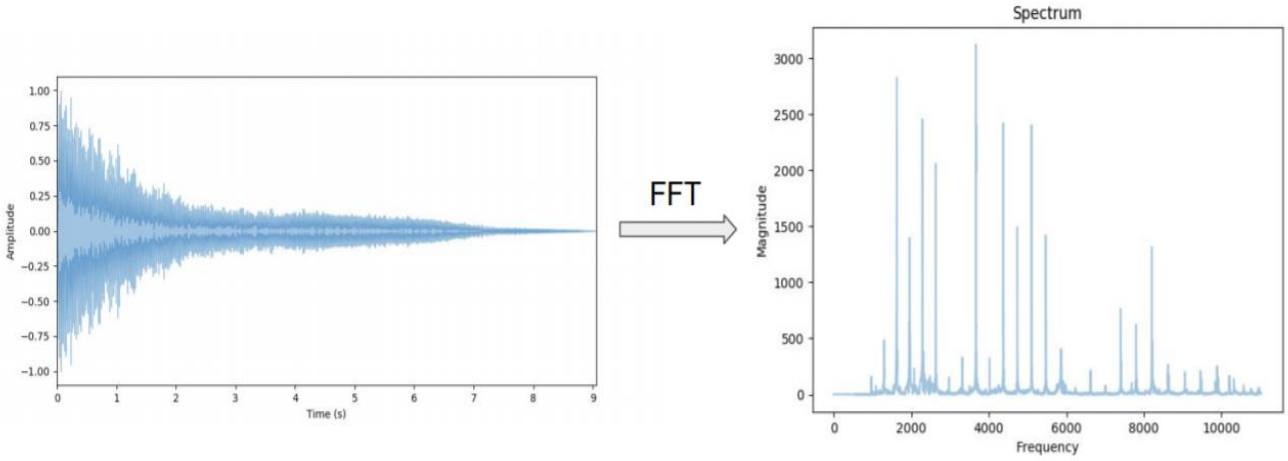
The following codes can be used to convert raw audio.wav to Waveform and display its chart:

```
import numpy as np
import librosa, librosa.display
import matplotlib.pyplot as plt

FIG_SIZE = (15,10)

file = "music.wav"

# load audio file with Librosa
signal, sample_rate = librosa.load(file, sr=22050)

# DISPLAY WAVEFORM
def waveform():
    plt.figure(figsize=FIG_SIZE)
    librosa.display.waveplot(signal, sample_rate, alpha=0.4)
    plt.xlabel("Time (s)")
    plt.ylabel("Amplitude")
    plt.title("Waveform")
    plt.show()
```

If you take look at the above figure, it looks so complex to understand, but nature has given us an incredible way of knowing quite a lot about complex sounds, and that's given through a Fourier transform (FFT). A Fourier transform is decomposing complex periodic sound into a sum of sine waves oscillating at different frequencies.



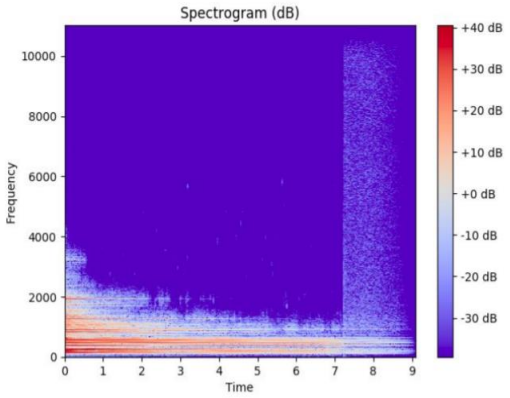
It is great because now we can decomposed complex sounds into simpler ones and analyze them. When we do a Fourier transform, we'll get some data like the below figure.



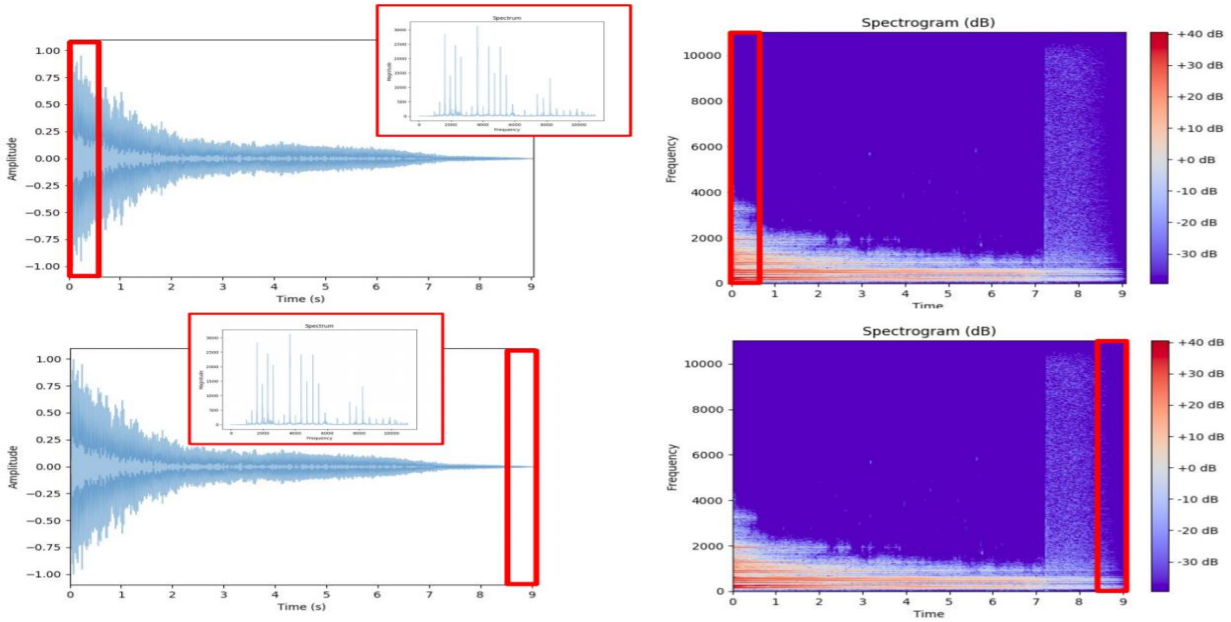
The following codes can be used to convert raw audio.wav to FFT and display its chart:

```
import numpy as np
import librosa, librosa.display
import matplotlib.pyplot as plt

FIG_SIZE = (15,10)
file = "music.wav"
signal, sample_rate = librosa.load(file, sr=22050)

# DISPLAY FFT TO POWER SPECTRUM
def fft():
    # perform Fourier transform
    fft = np.fft.fft(signal)
    # calculate abs values on complex numbers to get magnitude
    spectrum = np.abs(fft)
    # create frequency variable
    f = np.linspace(0, sample_rate, len(spectrum))
    # take half of the spectrum and frequency
    left_spectrum = spectrum[:int(len(spectrum)/2)]
    left_f = f[:int(len(spectrum)/2)]
    # plot spectrum
    plt.figure(figsize=FIG_SIZE)
    plt.plot(left_f, left_spectrum, alpha=0.4)
    plt.xlabel("Frequency")
    plt.ylabel("Magnitude")
    plt.title("Power spectrum")
    plt.show()
```

Note that when we do Fourier transform basically, we move from the time domain to the frequency domain and because of it we lose information about time. at first, it seems we lost a lot of information but there is a solution to that, and it's called short-time Fourier transform (STFT) given through a Fourier transform. A STFT looks like below figure.



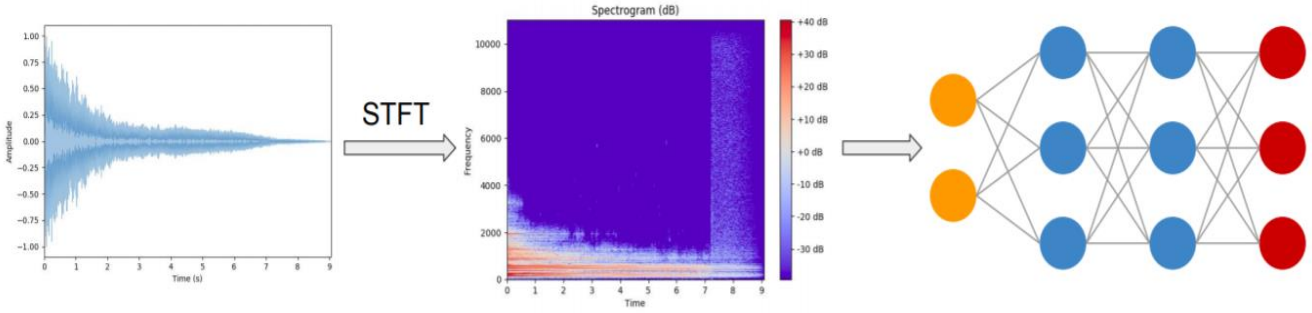
But let's say how does it work and how we can build STFT. It computes several Fourier transforms at different intervals and in doing so it preserves information about time and the way sounds evolved it's like over time, So the different intervals at which we perform the Fourier transform is given by the frame size and so a frame is a bunch of samples and so we fix he number of samples and we are giver spectrogram.



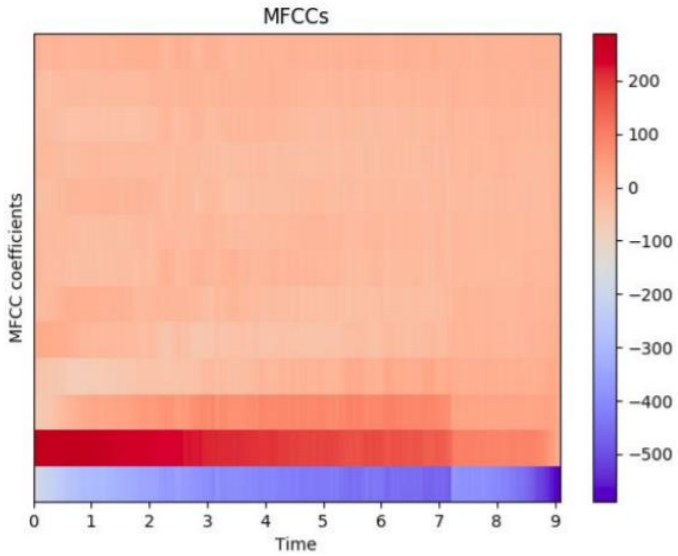
The following codes can be used to convert raw audio.wav to STFT and display its chart:

```
import numpy as np
import librosa, librosa.display
import matplotlib.pyplot as plt

FIG_SIZE = (15,10)
file = "music.wav"
signal, sample_rate = librosa.load(file, sr=22050)

# DISPLAY STFT TO SPECTROGRAM
def stft():
    hop_length = 512 # in num. of samples
    n_fft = 2048 # window in num. of samples
    # calculate duration hop length and window in seconds
    hop_length_duration = float(hop_length)/sample_rate
    n_fft_duration = float(n_fft)/sample_rate
    print("STFT hop length duration is: {}s".format(hop_length_duration))
    print("STFT window duration is: {}s".format(n_fft_duration))
    # perform stft
    stft = librosa.stft(signal, n_fft=n_fft, hop_length=hop_length)
    # calculate abs values on complex numbers to get magnitude
    spectrogram = np.abs(stft)
    # display spectrogram
    plt.figure(figsize=FIG_SIZE)
    librosa.display.specshow(spectrogram, sr=sample_rate, hop_length=hop_length)
    plt.xlabel("Time")
    plt.ylabel("Frequency")
    plt.colorbar()
    plt.title("Spectrogram")
    plt.show()
    # apply logarithm to cast amplitude to Decibels
    log_spectrogram = librosa.amplitude_to_db(spectrogram)
    plt.figure(figsize=FIG_SIZE)
    librosa.display.specshow(log_spectrogram, sr=sample_rate, hop_length=hop_length)
    plt.xlabel("Time")
    plt.ylabel("Frequency")
    plt.colorbar(format="%+2.0f dB")
    plt.title("Spectrogram (dB)")
    plt.show
```

Now we may be wondering why did we have to learn about spectrogram because spectrograms are fundamental for performing like deep learning like applications like on audio data the whole pre-processing pipeline for audio data for deep learning is based on spectrograms, so we can pass wave form audios into state and we get spectrogram and we use spectrogram as an input for our deep learning model.



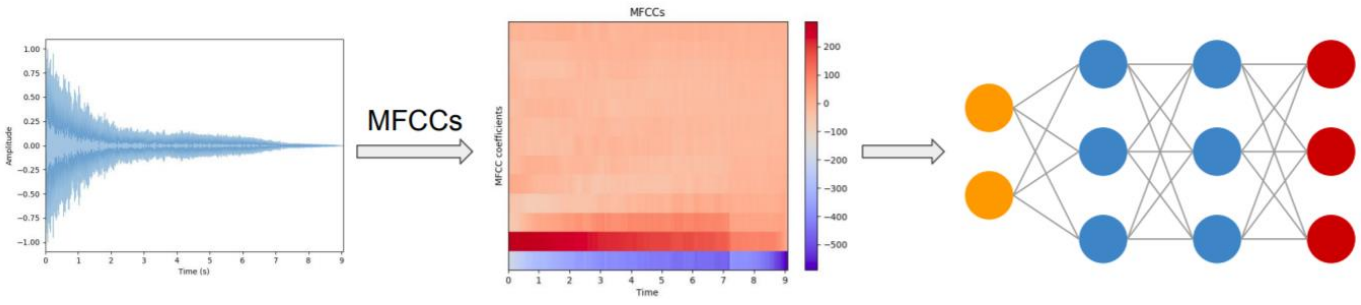
Let's introduce another feature that is fundamental and as important as spectrogram for deep learning, it called Mel Frequency Cepstral Coefficient (MFCC's). MFCC's capture of timbral/textural aspects of sound. it means if you have for example a piano and a violin playing the same melody you would have potentially like the same peach context and the same frequency but what would change is the quality of sound but MFCC's are capable of capturing the information and the differences. if you are interested in how MFCC's seem, look at the below figure.



The following codes can be used to convert raw audio.wav to MFCC's and display its chart:

```
import numpy as np
import librosa, librosa.display
import matplotlib.pyplot as plt

FIG_SIZE = (15,10)
file = "music.wav"
signal, sample_rate = librosa.load(file, sr=22050)

# DISPLAY MFCCs
def mfccs():
    hop_length = 512 # in num. of samples
    n_fft = 2048 # window in num. of samples
    # extract 13 MFCCs
    MFCCs = librosa.feature.mfcc(signal, sample_rate, n_fft=n_fft, hop_length=hop_len
gth, n_mfcc=13)
    # display MFCCs
    plt.figure(figsize=FIG_SIZE)
    librosa.display.specshow(MFCCs, sr=sample_rate, hop_length=hop_length)
    plt.xlabel("Time")
    plt.ylabel("MFCC coefficients")
    plt.colorbar()
    plt.title("MFCCs")
    plt.show()
```

And the way we use MFCC's is the same as how we use STFT.



WHAT DID WE USE TO DO? (optional)

In the past, traditional machine learning pre-processing for audio used to be so much different. let's take a look at that:

we can take a lot of information from waveforms, for example, we could use waveforms extracting time-domain features and use spectrogram extracting frequency domain features, so we start from the waveform and extract those features we wanted and we combine these features and use it in machine learning algorithms like logistic regression or super vector machine.

fortunately, with advanced deep learning, the whole process becomes a little bit more straight forward. After all, we don't need to see that much feature engineering because we use spectrogram. this is why deep learning models like in this case for audio is called end-to-end model cause you just use some basic information without worrying to much about extracting specific features.