2024 edition

Deep Learning for Music Analysis and Generation

Fundamentals for Musical Audio

(audio → features)

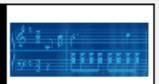
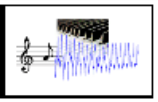
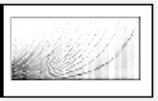
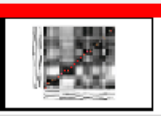
Yi-Hsuan Yang Ph.D.
yhyangtw@ntu.edu.tw

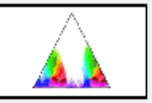
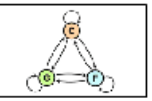
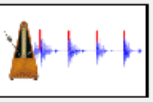
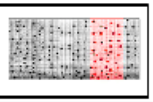
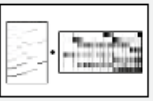
Outline

- Time and frequency representations for music
- Audio features for music
- Math in STFT
- DL 101

FMP Notebook

https://www.audiolabs-erlangen.de/resources/MIR/FMP/C1/C1.html
https://www.audiolabs-erlangen.de/resources/MIR/FMP/C2/C2.html

Part	Title	Notions, Techniques & Algorithms	HTML	IPYNB
B 	Basics	Basic information on Python, Jupyter notebooks, Anaconda package management system, Python environments, visualizations, and other topics	[html]	[ipynb]
0 	Overview	Overview of the notebooks (https://www.audiolabs-erlangen.de/FMP)	[html]	[ipynb]
1 	Music Representations	Music notation, MIDI, audio signal, waveform, pitch, loudness, timbre	[html]	[ipynb]
2 	Fourier Analysis of Signals	Discrete/analog signal, sinusoid, exponential, Fourier transform, Fourier representation, DFT, FFT, STFT	[html]	[ipynb]
3 	Music Synchronization	Chroma feature, dynamic programming, dynamic time warping (DTW), alignment, user interface	[html]	[ipynb]

Part	Title	Notions, Techniques & Algorithms	HTML	IPYNB
4 	Music Structure Analysis	Similarity matrix, repetition, thumbnail, homogeneity, novelty, evaluation, precision, recall, F-measure, visualization, scape plot	[html]	[ipynb]
5 	Chord Recognition	Harmony, music theory, chords, scales, templates, hidden Markov model (HMM), evaluation	[html]	[ipynb]
6 	Tempo and Beat Tracking	Onset, novelty, tempo, tempogram, beat, periodicity, Fourier analysis, autocorrelation	[html]	[ipynb]
7 	Content-Based Audio Retrieval	Identification, fingerprint, indexing, inverted list, matching, version, cover song	[html]	[ipynb]
8 	Musically Informed Audio Decomposition	Harmonic/percussive separation, signal reconstruction, instantaneous frequency, fundamental frequency (F0), trajectory, nonnegative matrix factorization (NMF)	[html]	[ipynb]

Outline

- **Time and frequency representations for music**
- Audio features for music
- Math in STFT
- DL 101

Audio Waveforms

https://www.audiolabs-erlangen.de/resources/MIR/FMP/C1/C1S3_Waveform.html

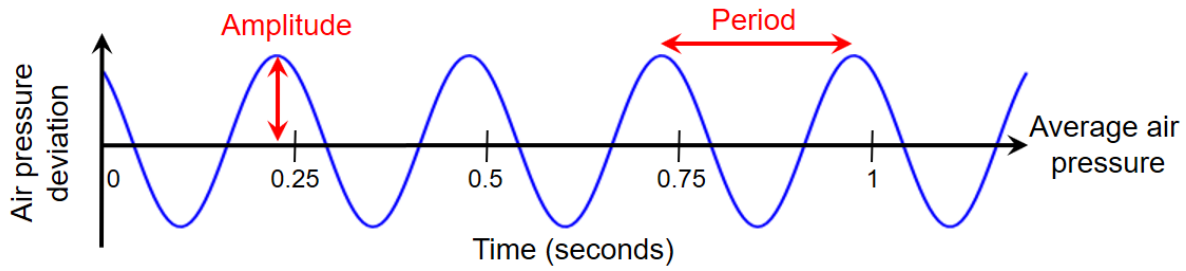
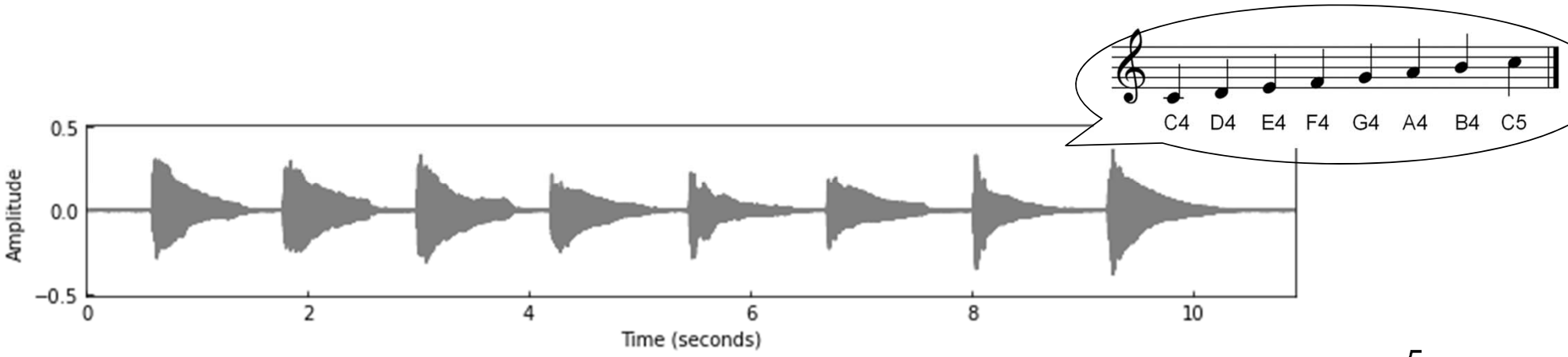


Figure 1.19 from [Müller, FMP, Springer 2015]



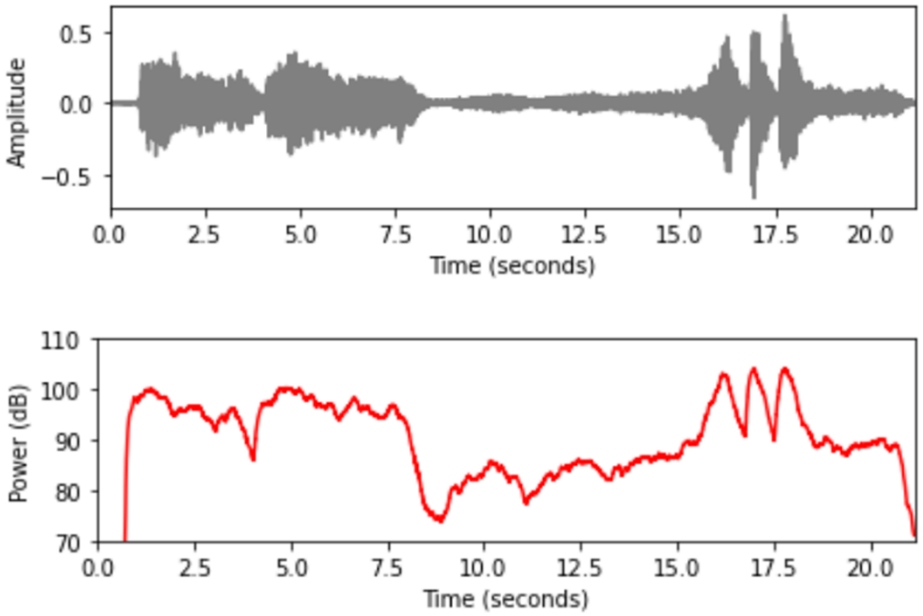
What Do We Perceive in Musical Audio?

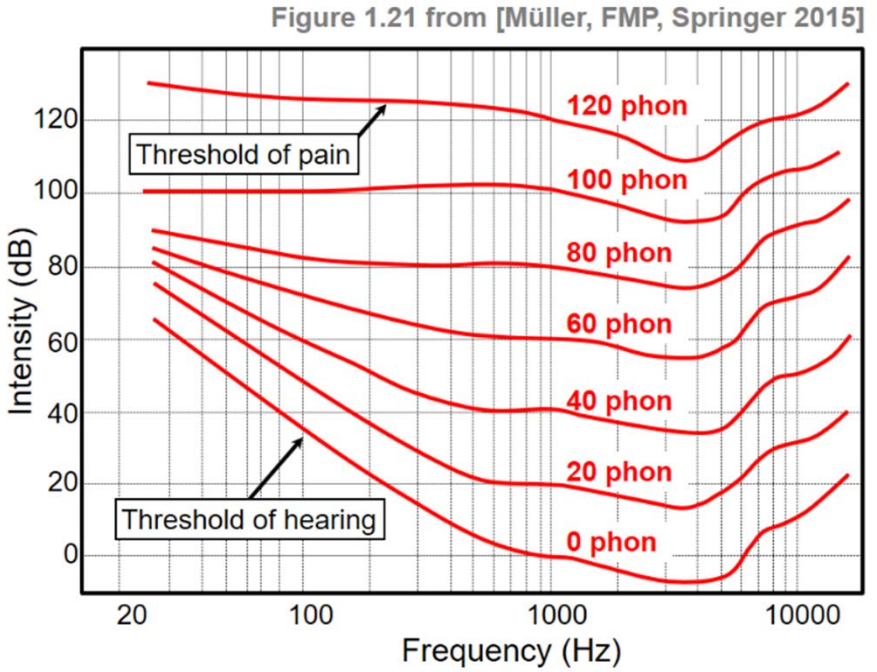
- Loudness
- Timbre / instrument / singer identity
- Pitch / melody
- Chord / harmony
- Tempo / rhythm
- Style
- Emotion
- Spatial information

Dynamics, Intensity, and Loudness

https://www.audiolabs-erlangen.de/resources/MIR/FMP/C1/C1S3_Dynamics.html

- **Intensity**: a physical property
  - Other names: power (dB); energy
- **Loudness**: a perceptual property
  - Less used in deep learning research





Timbre: Different Instruments

https://www.audiolabs-erlangen.de/resources/MIR/FMP/C1/C1S3_Timbre.html

- **Timbre** (tone color; tone quality)
  - Allows a listener to distinguish the musical tone of a violin, an oboe, or a trumpet even if the tone is played at the same **pitch** and with the same **loudness**
  - Hard to grasp & *subjective*
  - May be described with words such as *bright*, *dark*, *warm*, and *harsh*

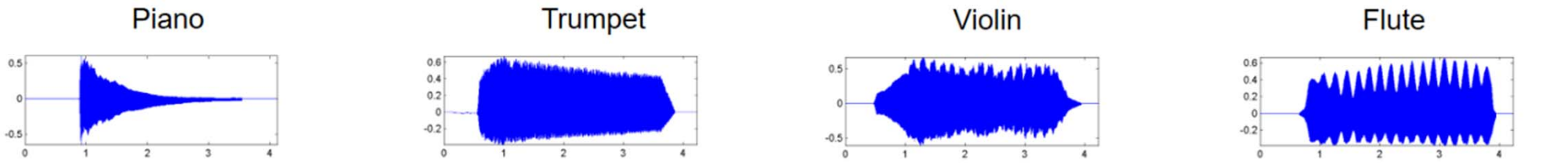
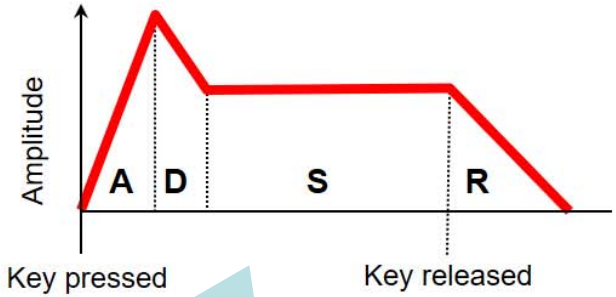


Figure 1.23 from [Müller, FMP, Springer 2015]

0:04 / 0:04 0:00 / 0:04 0:00 / 0:04 0:00 / 0:04

Temporal Characteristics

- **ADSR**: attack (A), decay (D), sustain (S), and release (R)
  - Hard to analyze when there are concurrent notes



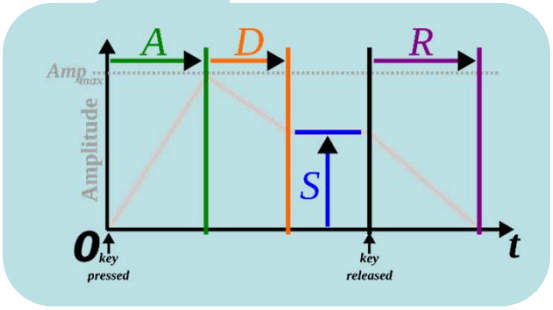
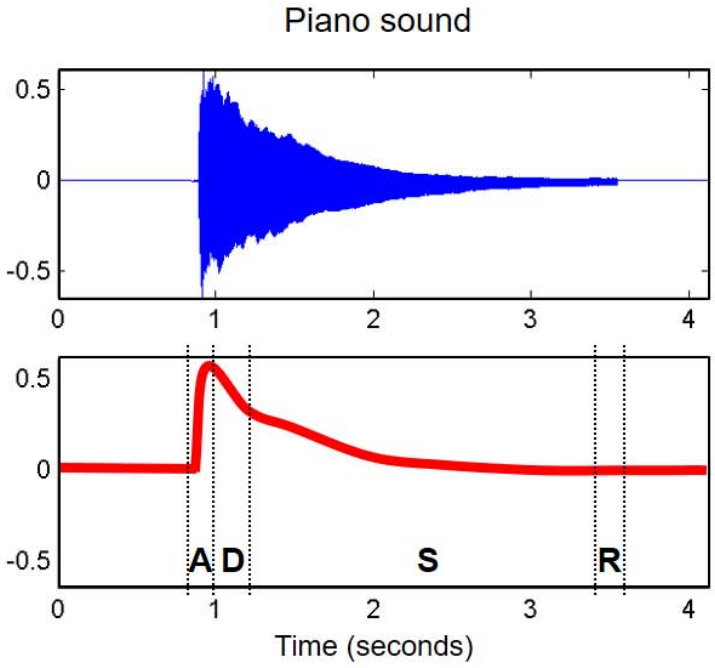


Figure from:
https://en.wikipedia.org/wiki/Envelope_(music)



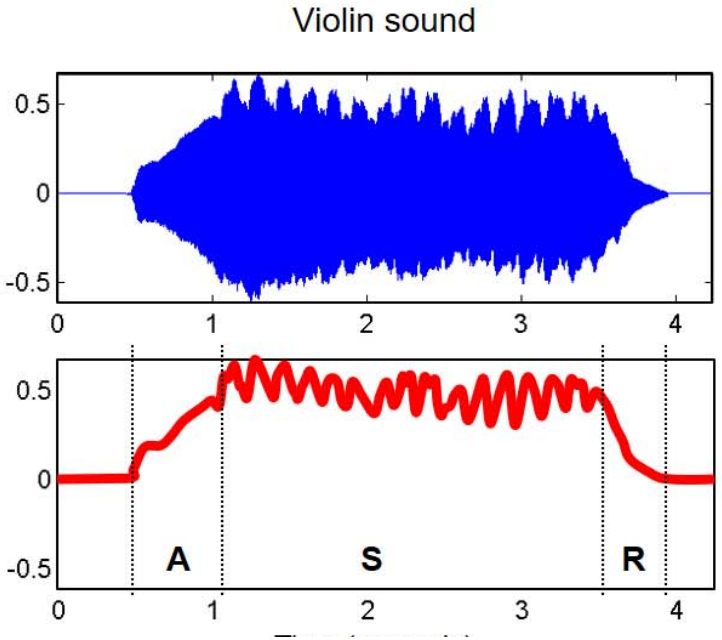
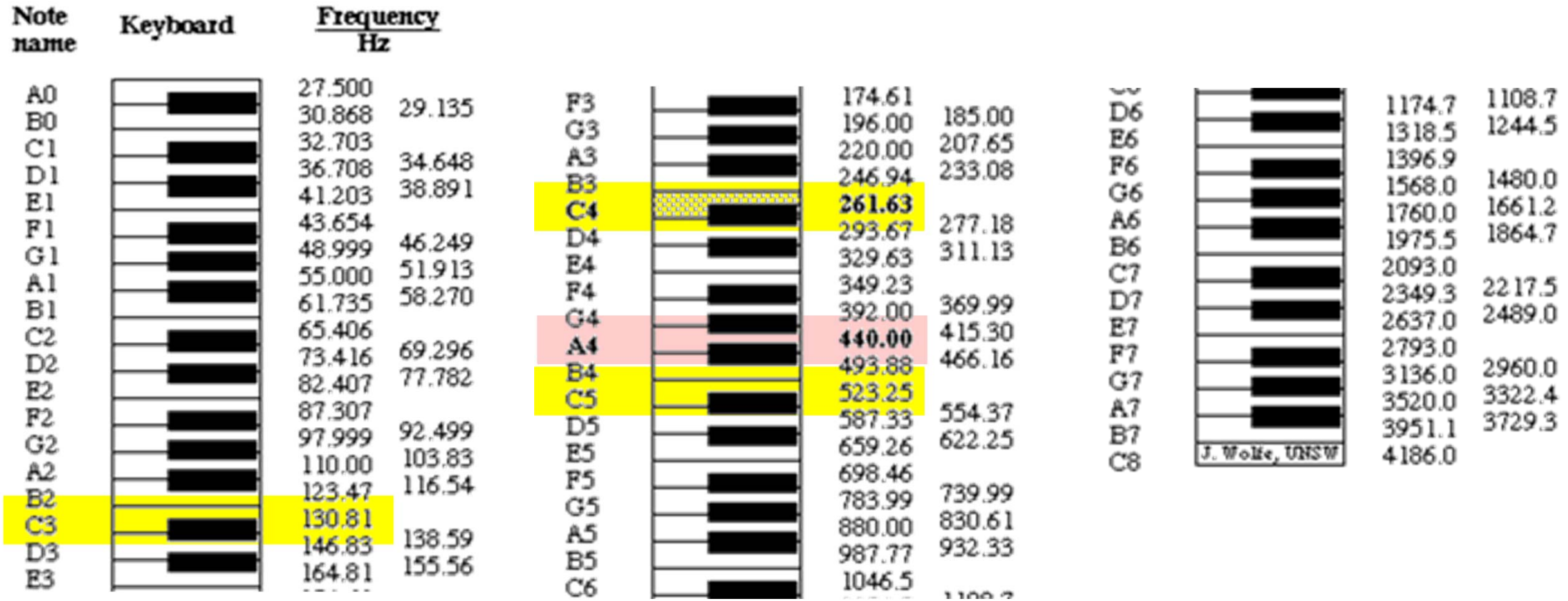


Figure 1.22b and Figure 1.23
from [Müller, FMP, Springer 2015]

Notes and Pitches

(Assuming A4=440 Hz; but there are *other tunings*)

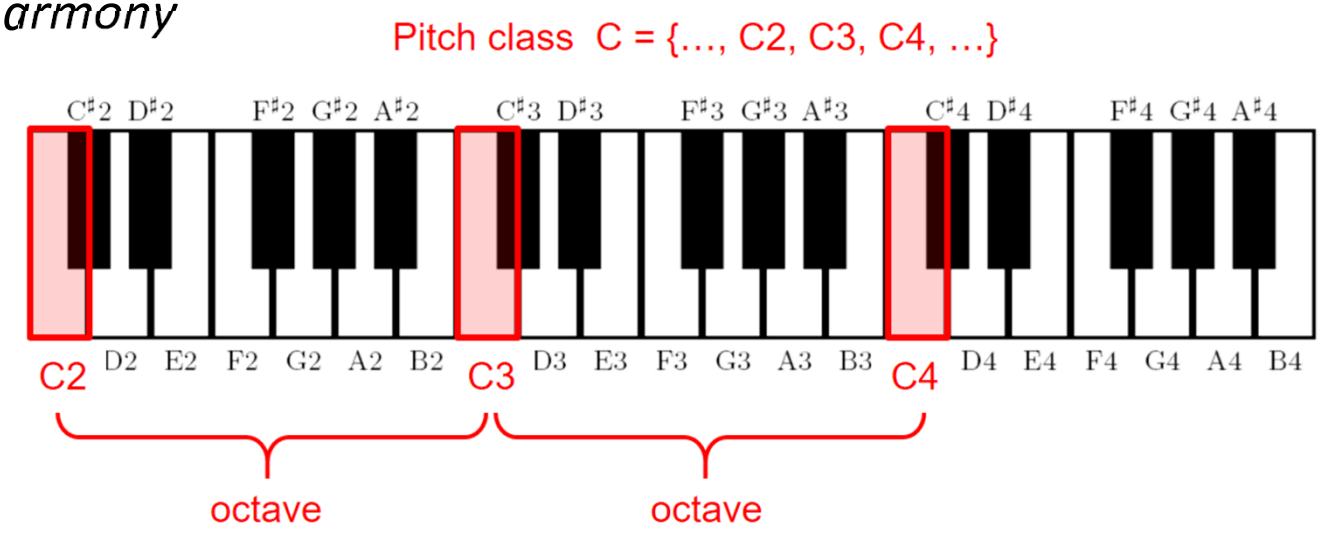
https://newt.phys.unsw.edu.au/jw/notes.html



Notes and Pitches

https://www.audiolabs-erlangen.de/resources/MIR/FMP/C1/C1S1_MusicalNotesPitches.html

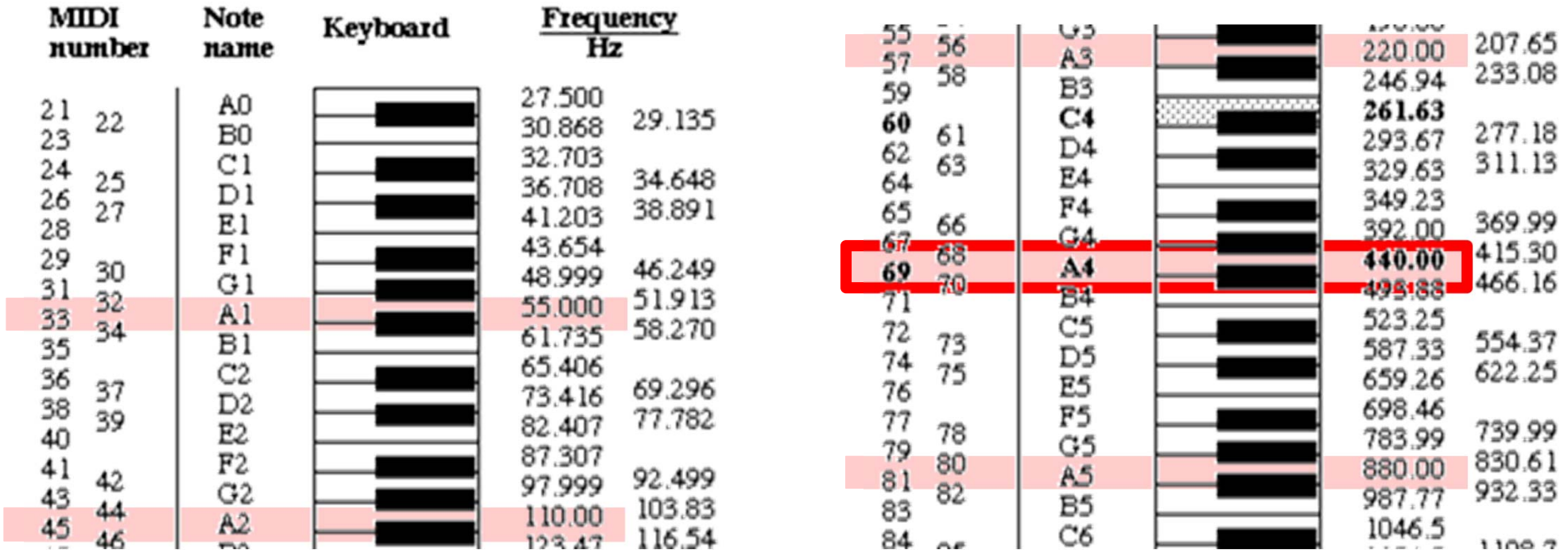
- **Pitch Class**
  - Two notes with *fundamental frequencies* in a ratio equal to *any power of* ***two*** (e.g., half, twice, or four times) are perceived as very **similar**
  - All notes with this kind of relation can be grouped under the same *pitch class*
  - Related to *chords/harmony*



MIDI Note Numbers

https://en.wikipedia.org/wiki/MIDI_tuning_standard

$$f = 2^{(d-69)/12} \cdot 440\ \text{Hz} \qquad d = 69 + 12\log_2\left(\frac{f}{440\ \text{Hz}}\right)$$

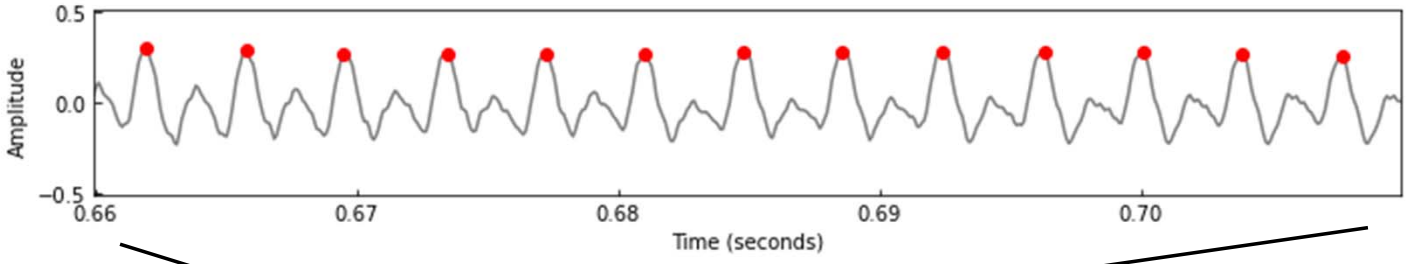


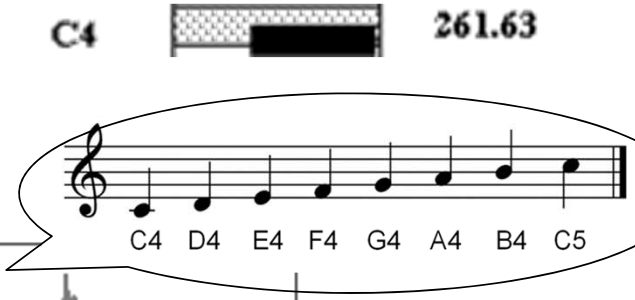
Audio Waveforms (Cont')

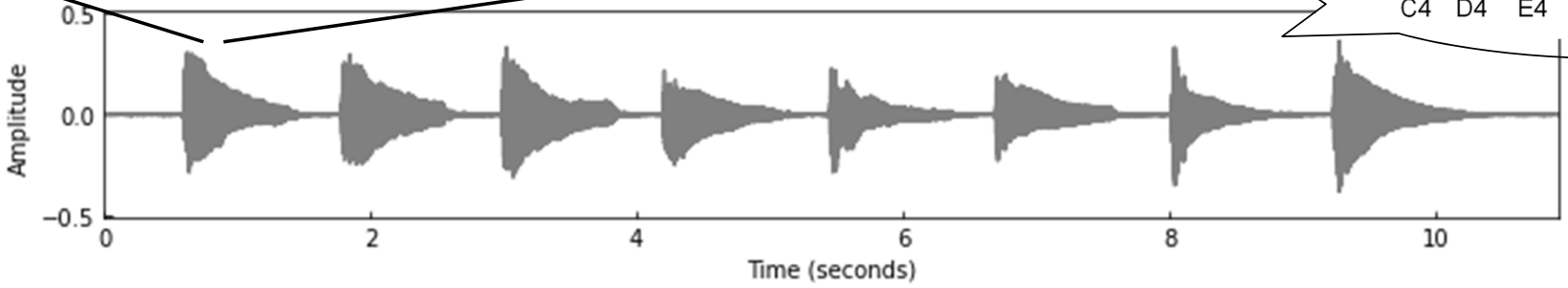
https://www.audiolabs-erlangen.de/resources/MIR/FMP/C1/C1S3_Waveform.html

Zoom-in of the section between 0.66 and 0.71 seconds

- Highly repetitive
- 13 high-pressure (red) points → 20*13=260 Hz

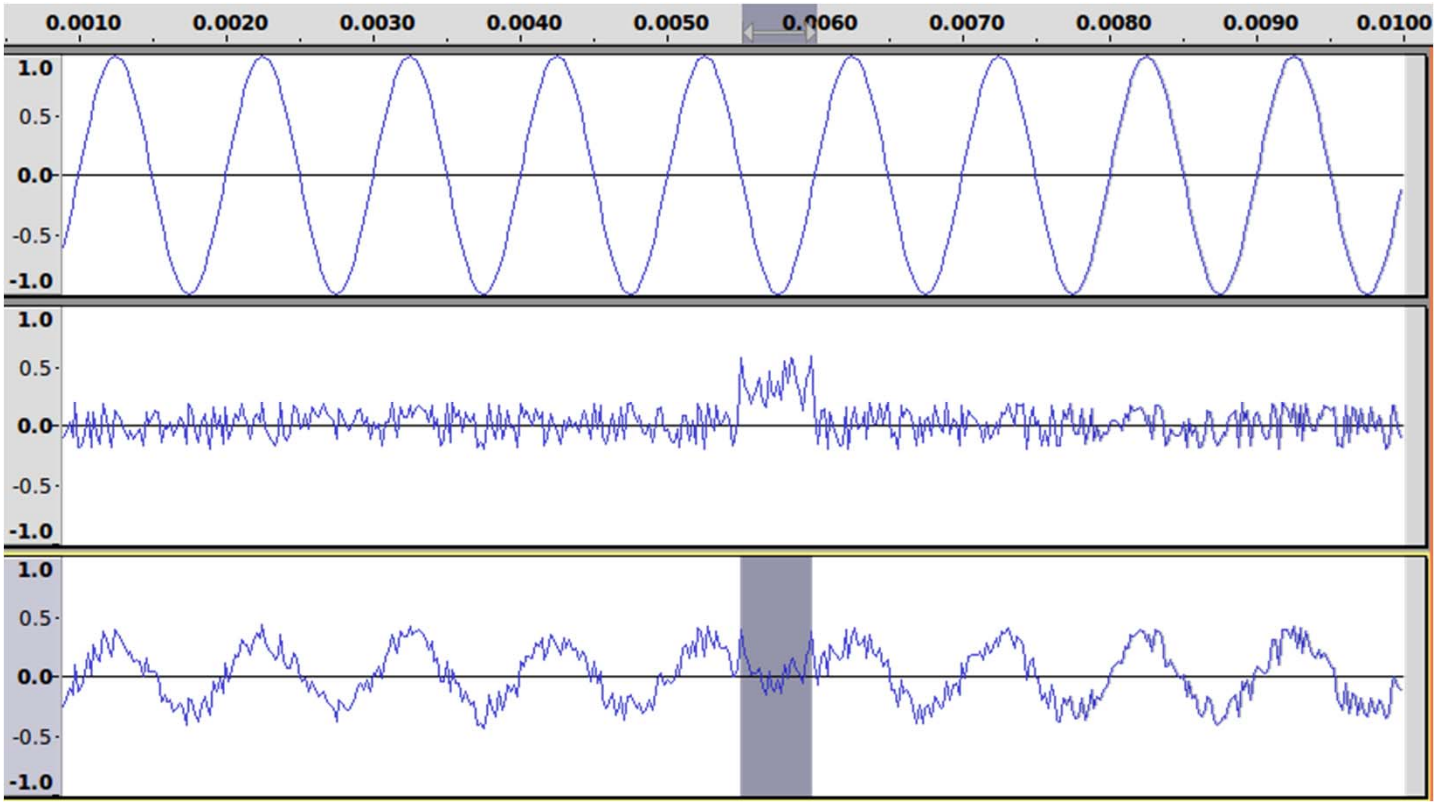






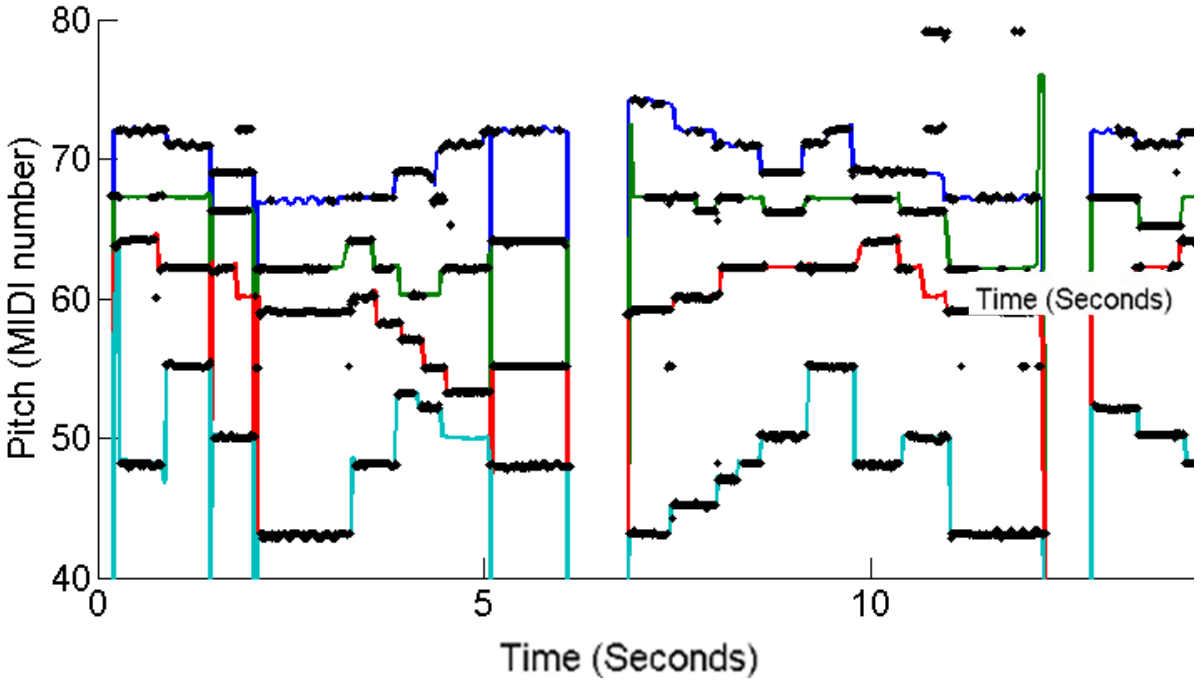
Temporal Characteristics

- **Zero-crossing rate**
  - How often a signal changes its sign
  - **Nosie**-like or not
  - Can be used in detecting the presence of vocals in speech signals
  - Cannot be used for estimating pitch due to noises and concurrent notes

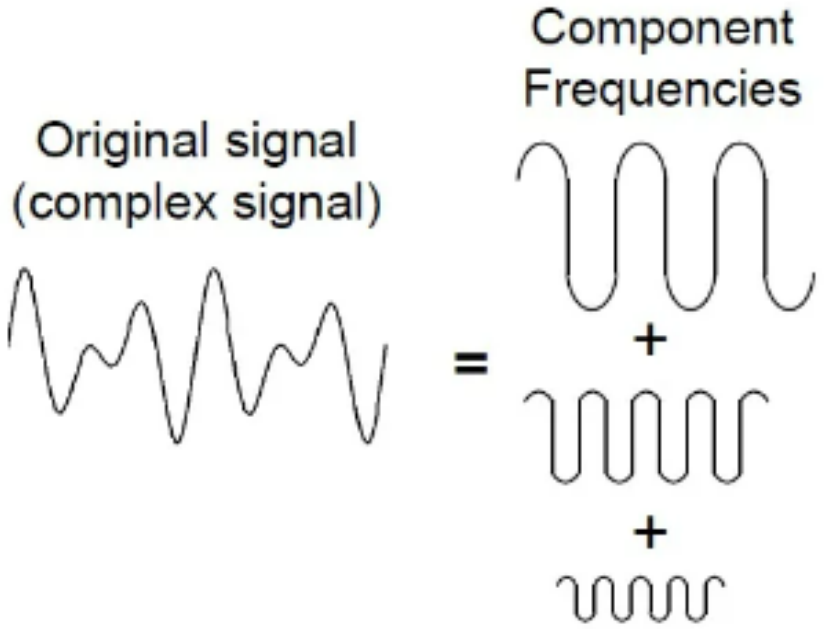


https://sandilands.info/sgordon/impact-of-noise-on-sine-wave-compared-to-square-wave

Multiple Concurrent Pitches



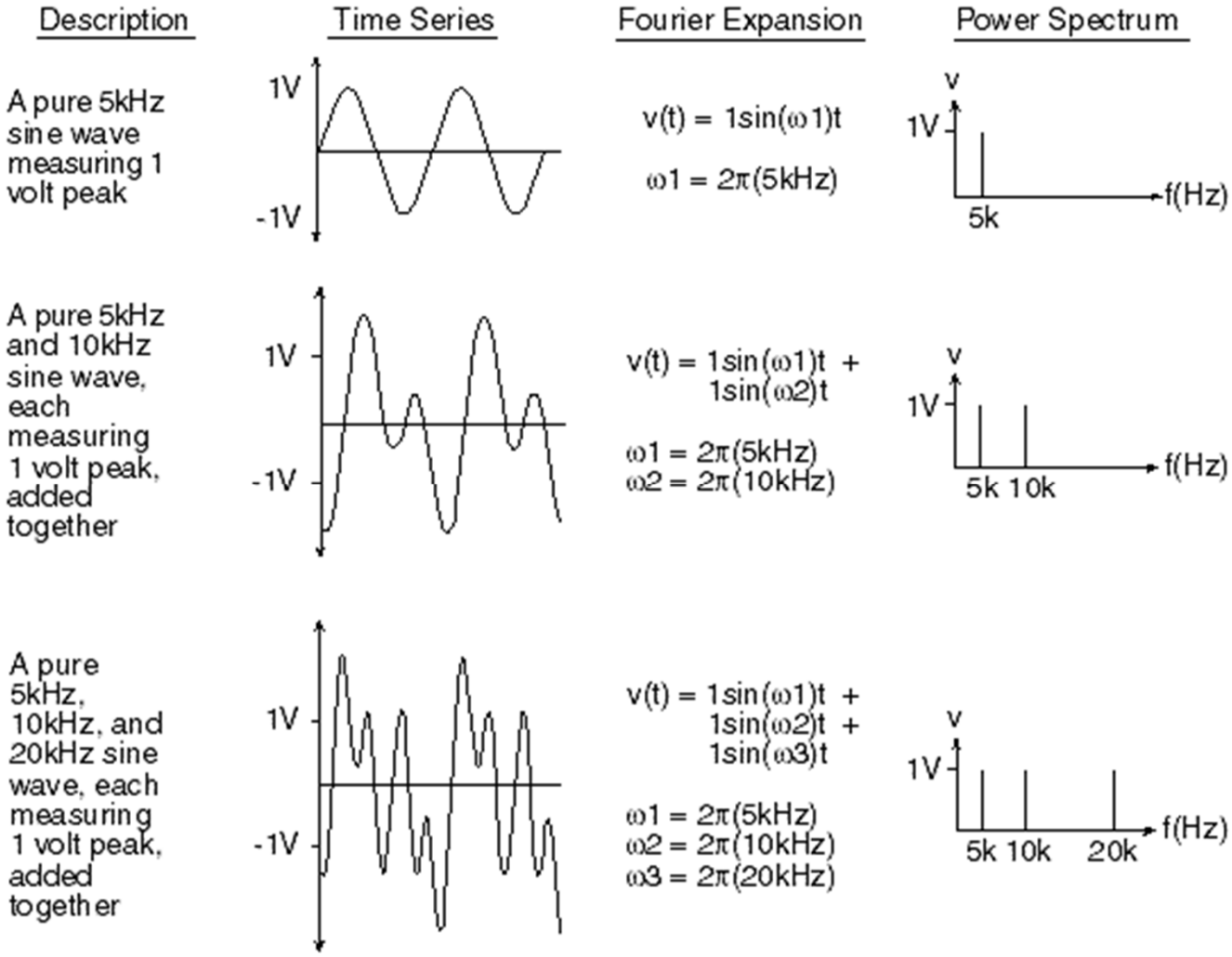
https://labsites.rochester.edu/air/projects/multipitch/multipitch.html



https://www.allaboutcircuits.com/technical-articles/an-introduction-to-filters/

Frequency-domain Analysis

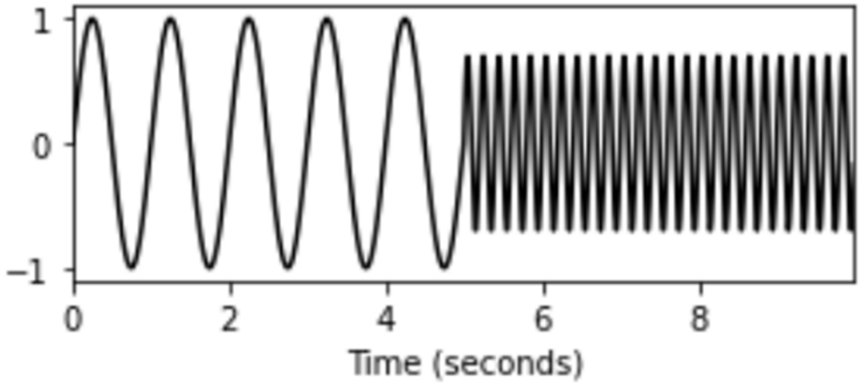
- Main subject of "Signals and Systems"
- **Fourier Transform**

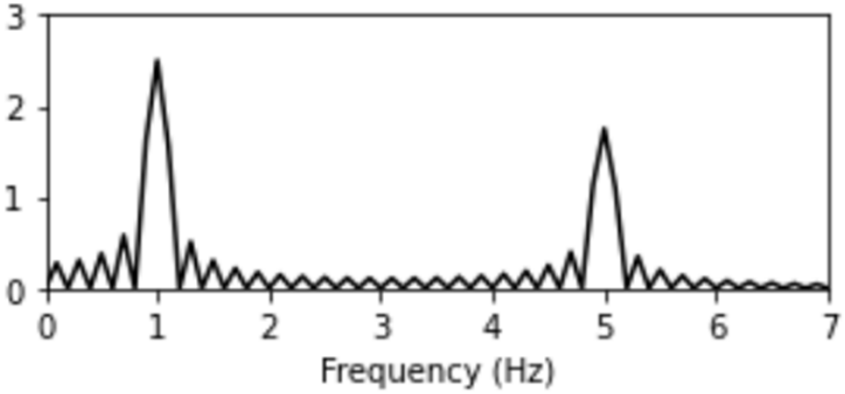


Frequency-domain Analysis

https://www.audiolabs-erlangen.de/resources/MIR/FMP/C2/C2_STFT-Basic.html

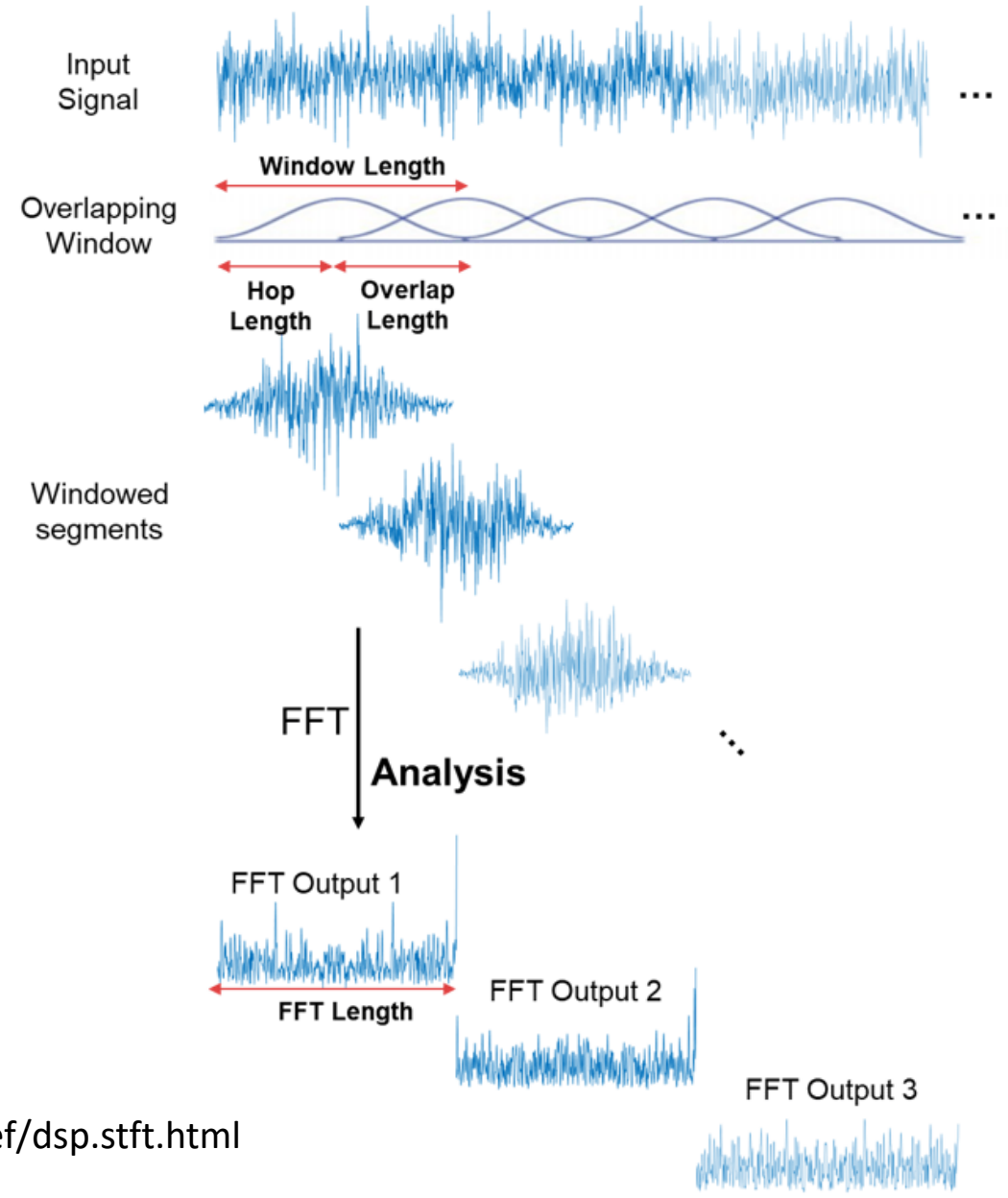
- **Fourier Transform cannot localize temporal events**





Frequency-domain Analysis

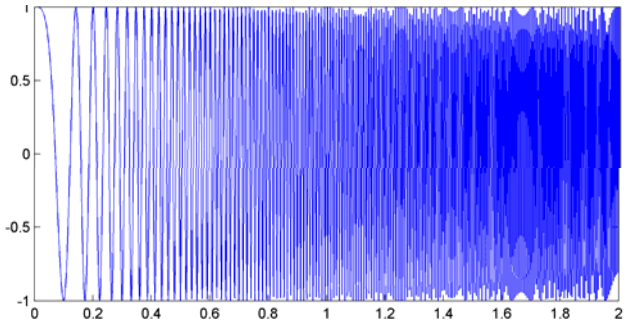
- Fourier Transform (FFT) cannot localize temporal events
- **Short-Time Fourier Transform** (STFT)
  - *Time-frequency representation* of signals
  - Slide **short windows** over the input signal and then compute the FFT locally for each windowed signal



https://ww2.mathworks.cn/help/dsp/ref/dsp.stft.html

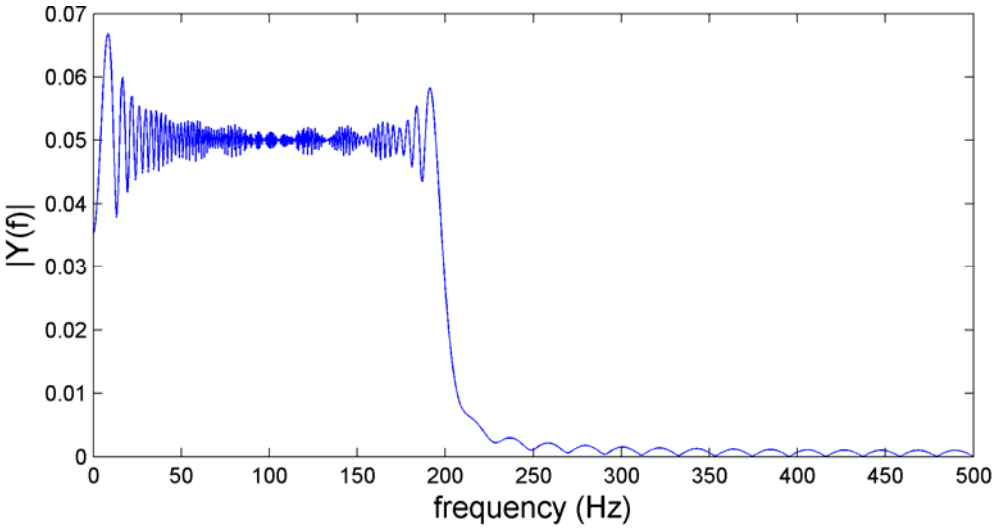
FFT vs STFT

Time-domain **waveform**

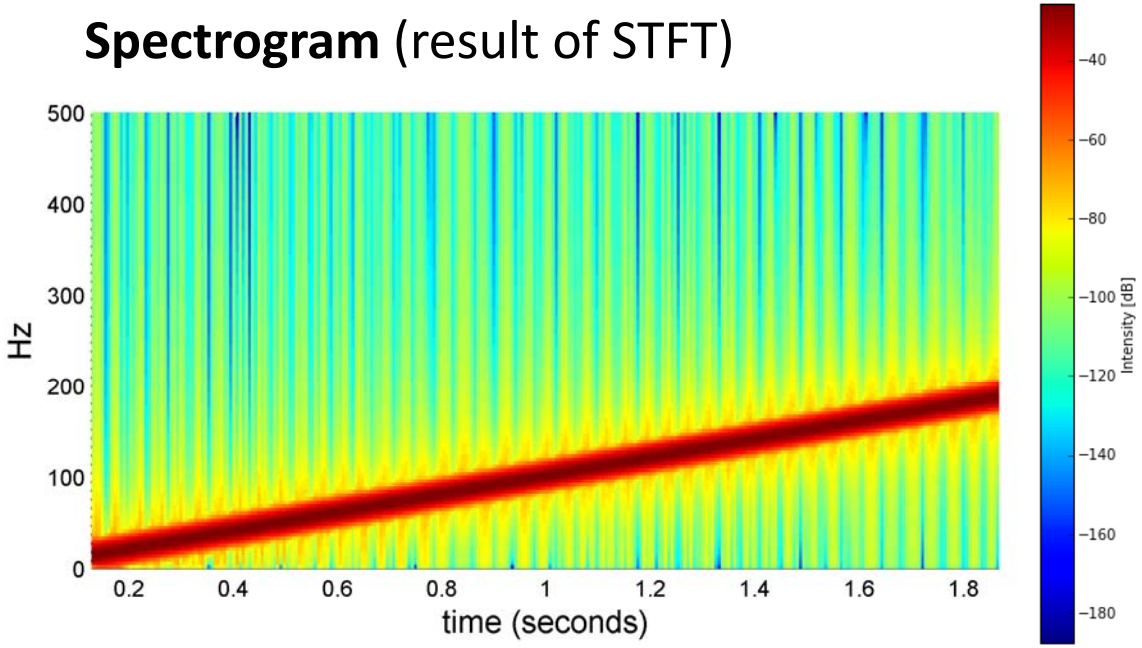


- **Waveform**: amplitude vs time
- **Spectrum**: amplitude vs frequency
- **Spectrogram**: time vs frequency; use *color* for amplitude

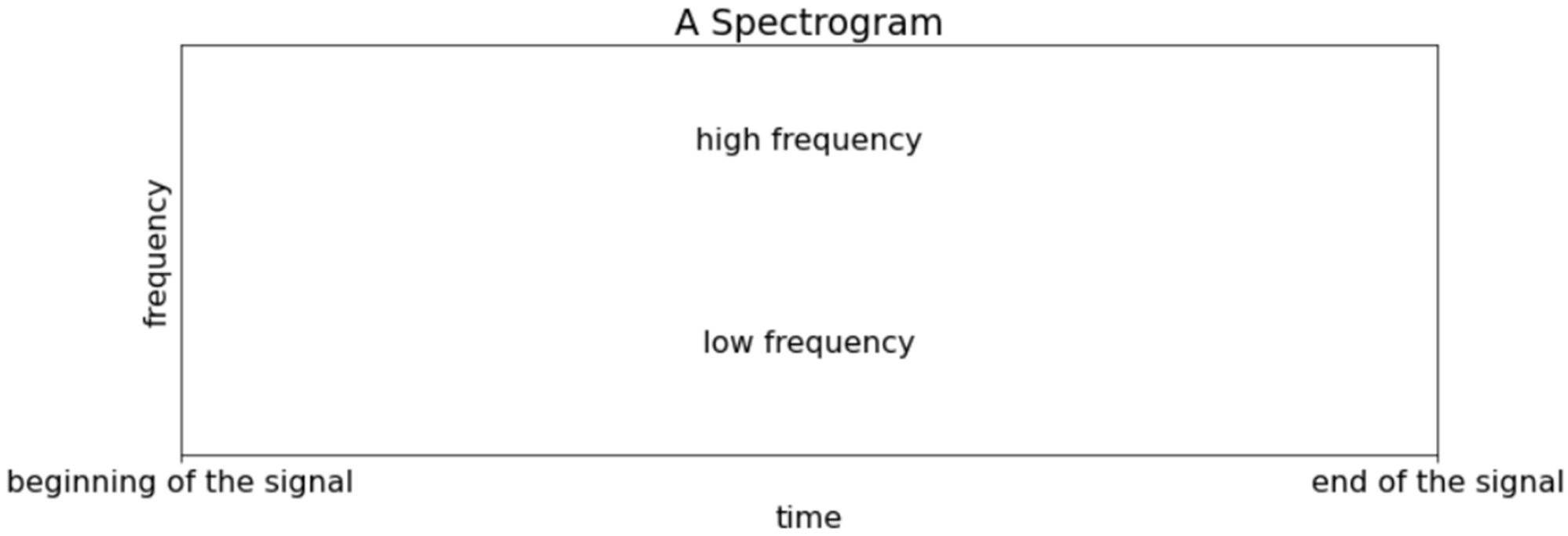
Spectrum (result of FFT)



Spectrogram (result of STFT)



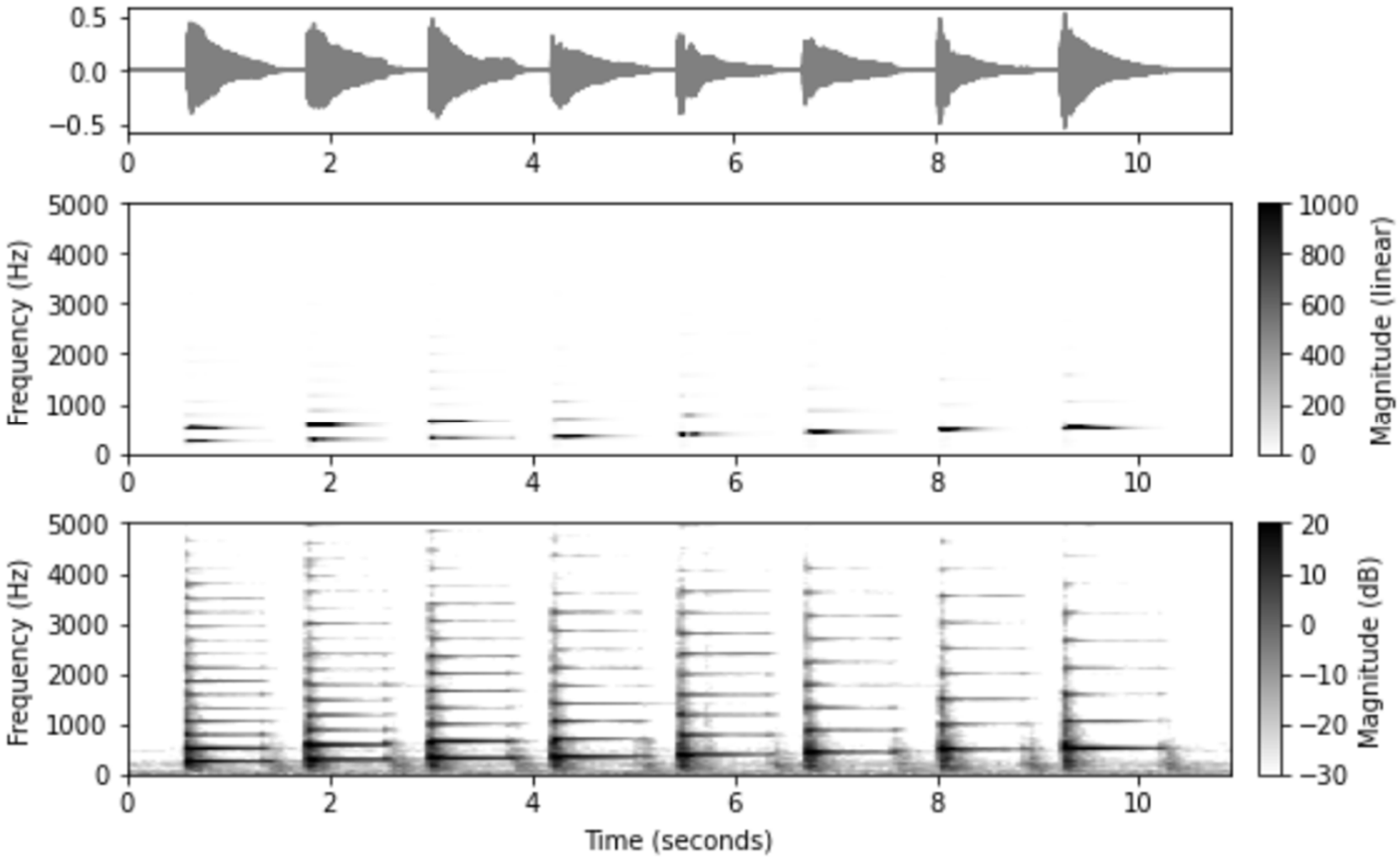
Time-Frequency Analysis via STFT



https://music-classification.github.io/tutorial/part2_basics/input-representations.html

Time-Frequency Analysis via STFT

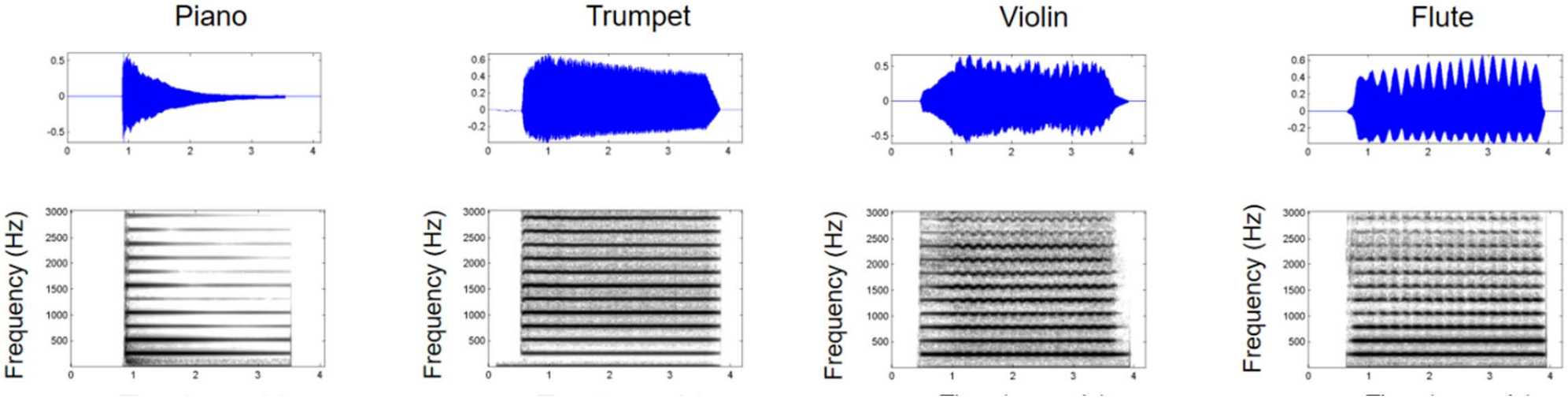
https://www.audiolabs-erlangen.de/resources/MIR/FMP/C2/C2_STFT-Basic.html



- **Time-frequency representation** of audio
  - Linear frequency, **linear** magnitude
  - Linear frequency, **logarithmic** magnitude (dB)

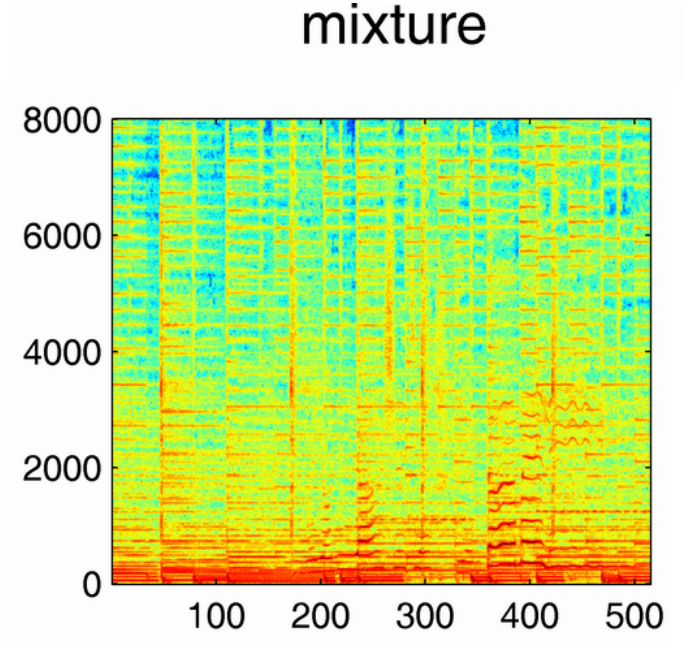
Spectral Characteristics

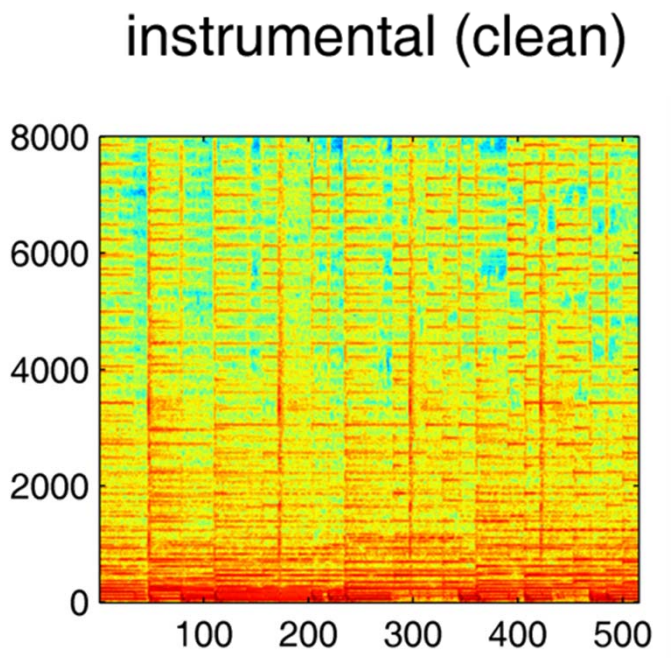
- Fundamental frequency (**F0**)
- **Partials** (harmonics): usually at frequencies which are **integer multiples** of F0
  - The partials in different instruments may exhibit **relative strengths**
  - The frequency may *deviate* from the ideal harmonics (this is called **inharmonicity**)

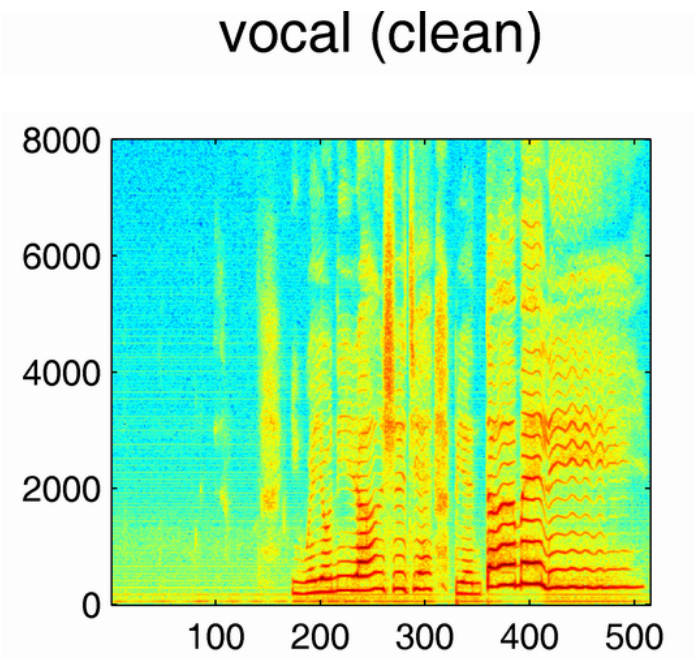


Time-Frequency Analysis via STFT

- For tasks such as classification/detection and source separation

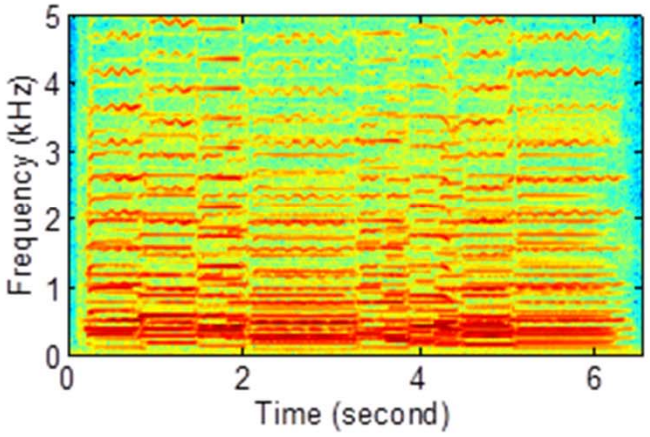




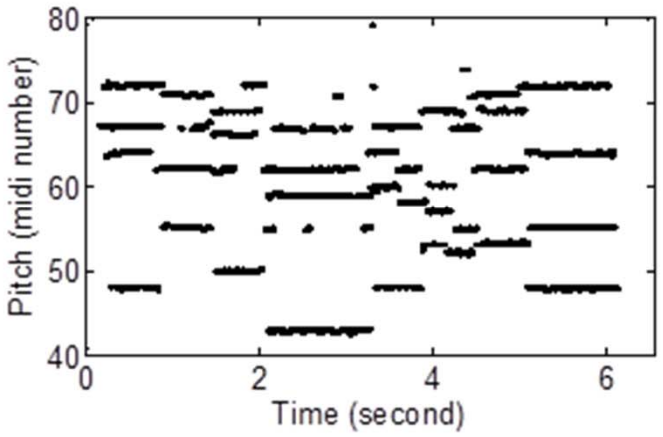


Time-Frequency Analysis via STFT

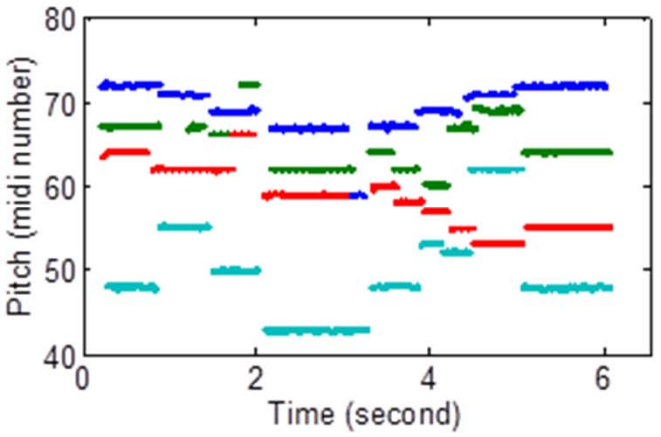
- For tasks such as multi-pitch estimation (MPE) and transcription



Spectrogram



Multi-pitch estimation results

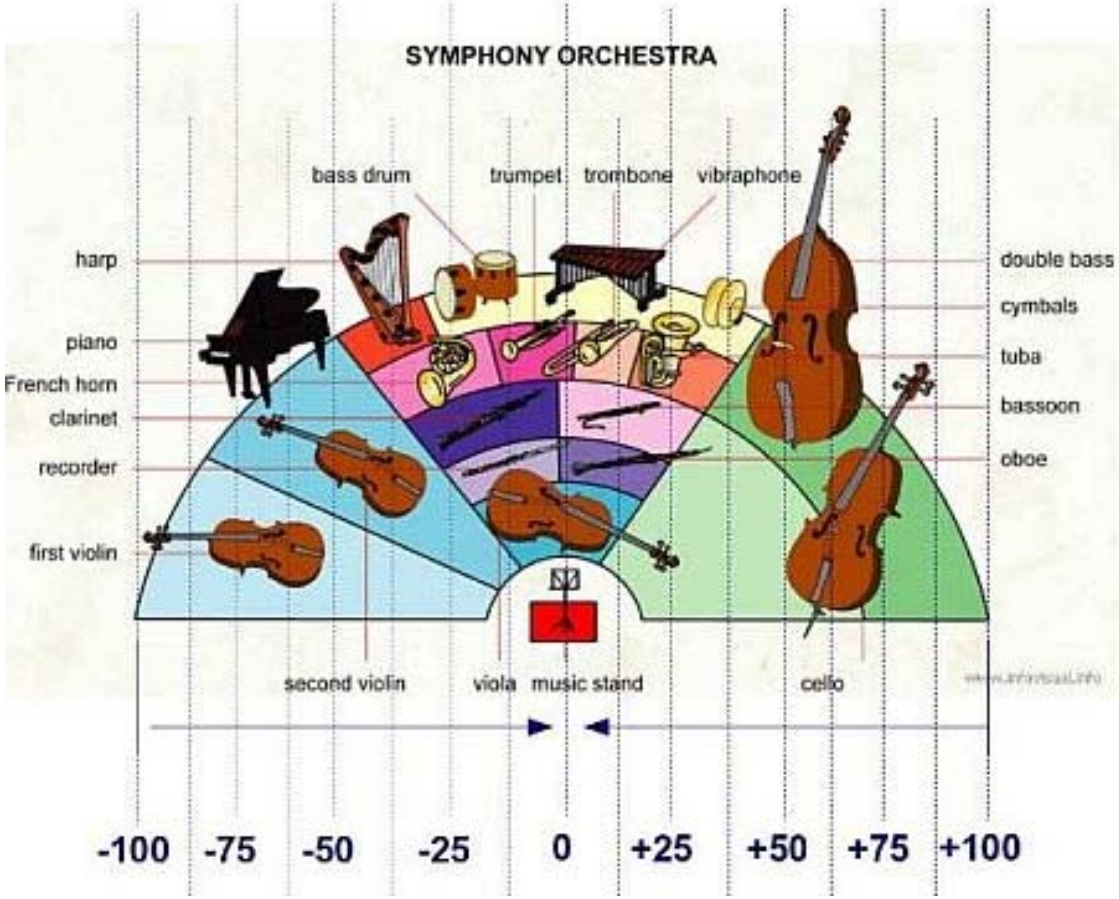


MPE + instrument detection

https://labsites.rochester.edu/air/projects/multipitch/multipitch.html

Spatial Information

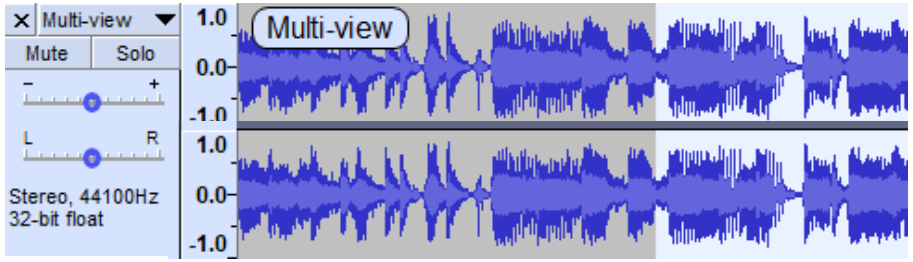
https://www.renegadeproducer.com/audio-panning.html

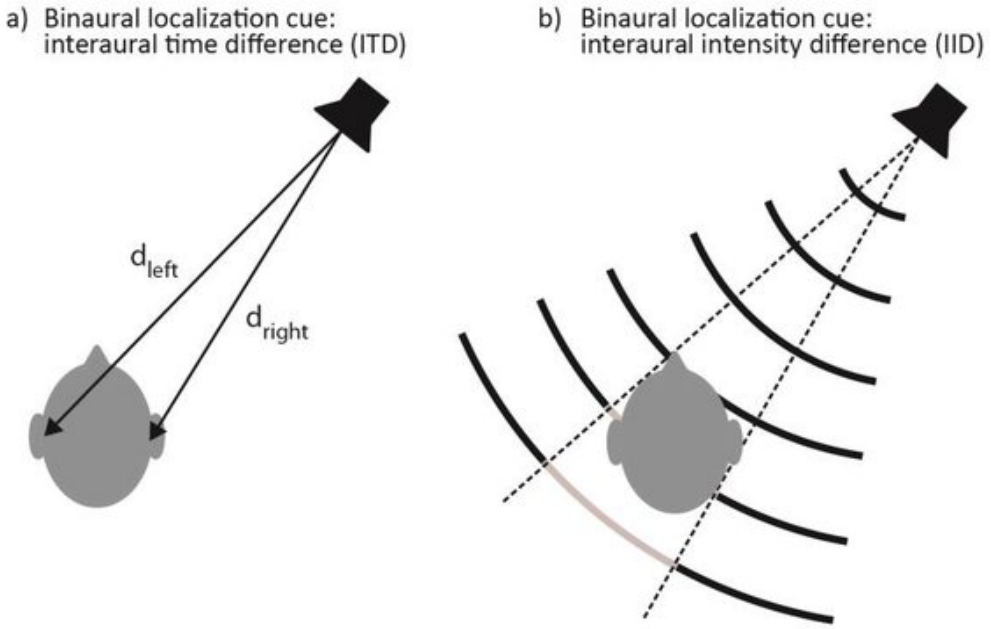


https://www.audiorecording.me/symphony-orchestra-panning-and-reverb-settings.html

Spatial Information: Mono vs Stereo

- Monaural
- Binaural





Source: https://www.researchgate.net/figure/Binaural-and-monaural-cues-used-for-sound-localization-a-For-sound-sources-off-the_fig1_299281975

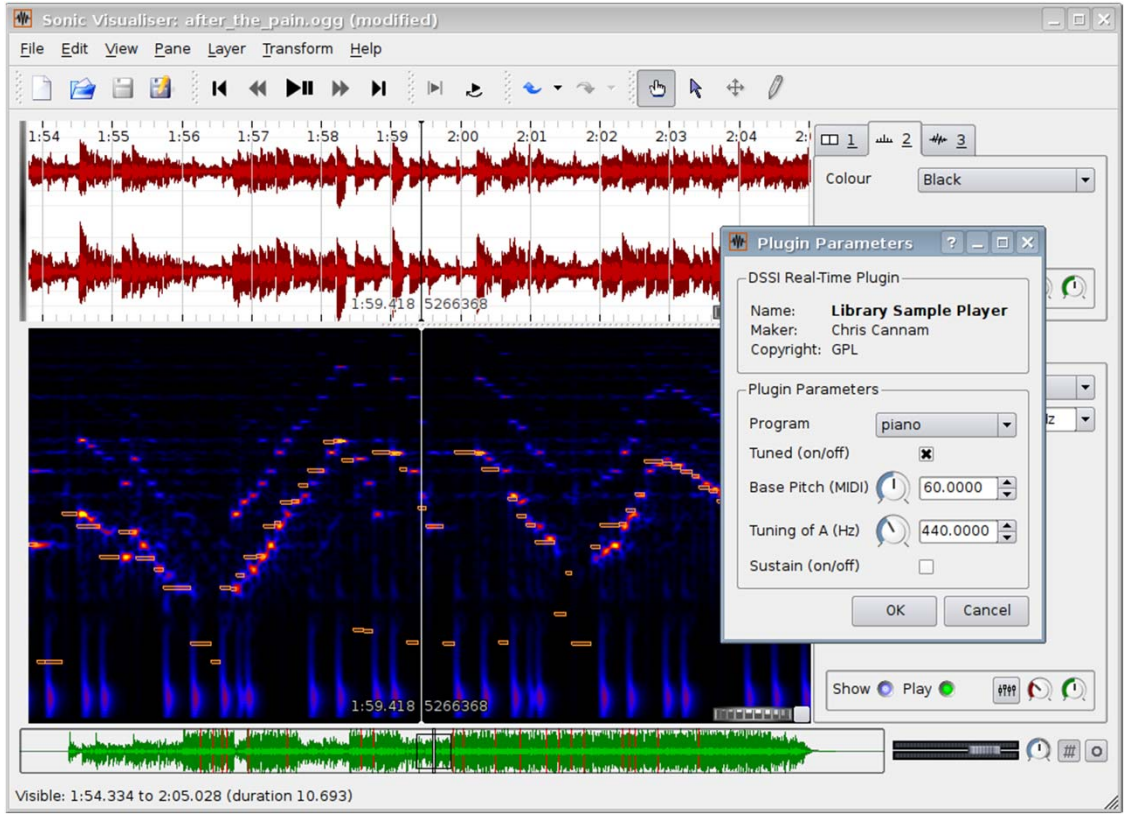
- Stereo-to-mono conversion
  - By taking the average
- Mono-to-stereo conversion
  - Need research
    https://www.youtube.com/watch?v=aWxmQKm_s8Q

What Do We Perceive in Musical Audio?

- **Loudness**
- **Timbre** / instrument / singer identity
- **Pitch** / melody
- Chord / harmony → lecture 3
- Tempo / rhythm → lecture 3
- Style → lecture 3
- Emotion → lecture 3
- **Spatial information**

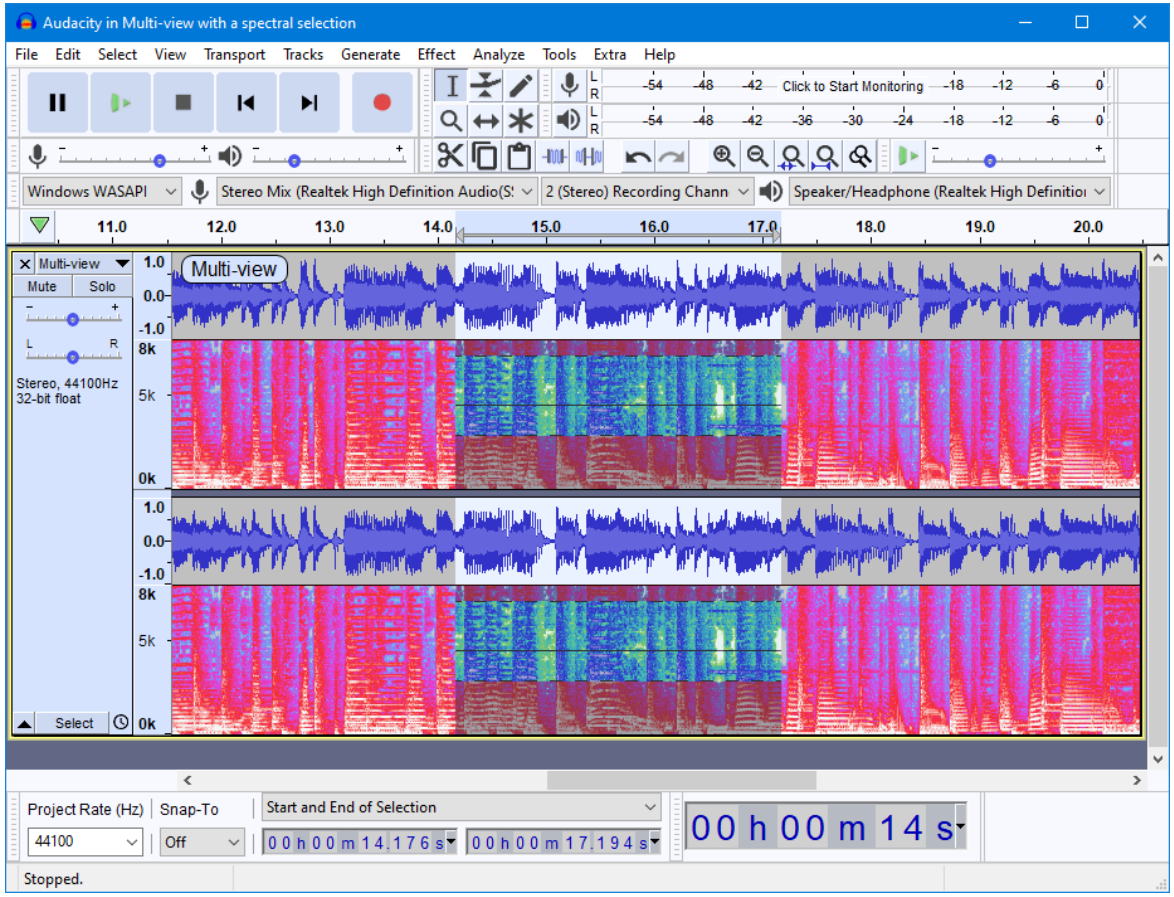
Software: Sonic Visualiser

http://www.sonicvisualiser.org/



Software: Audacity

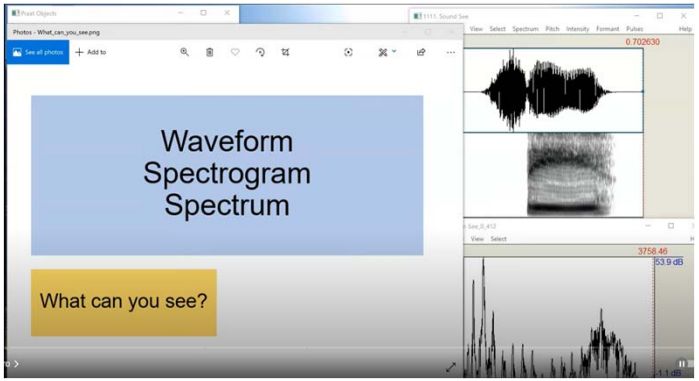
https://www.audacityteam.org/



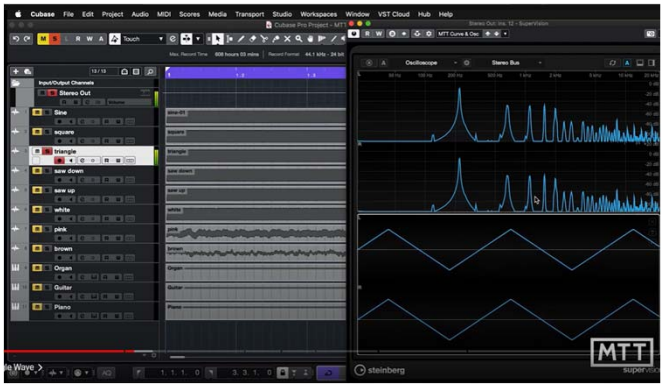
Let’s Find Some Audio Recordings

- https://www.freesound.org/
  - https://www.freesound.org/people/acclivity/sounds/22347/
  - https://www.freesound.org/people/Rudmer_Rotteveel/sounds/316915/
  - https://www.freesound.org/people/Jaylew1987/sounds/321112/
  - https://www.freesound.org/people/mickel11/sounds/90803/

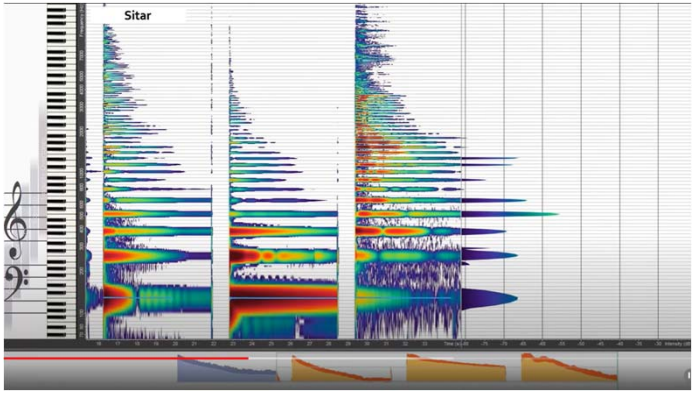
Related Videos



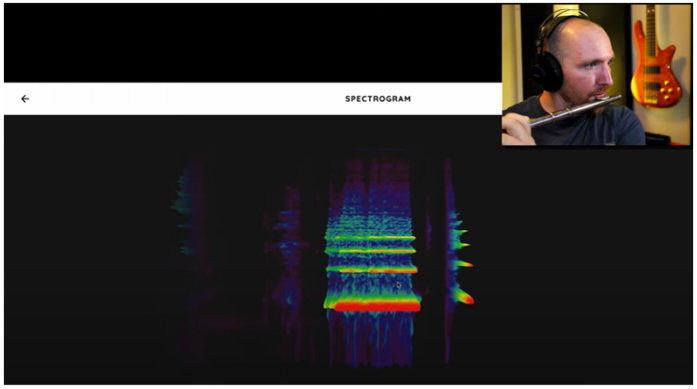
https://www.youtube.com/watch?v=2Hj1kAWVjLo



https://www.youtube.com/watch?v=fZ18C5RXDcc



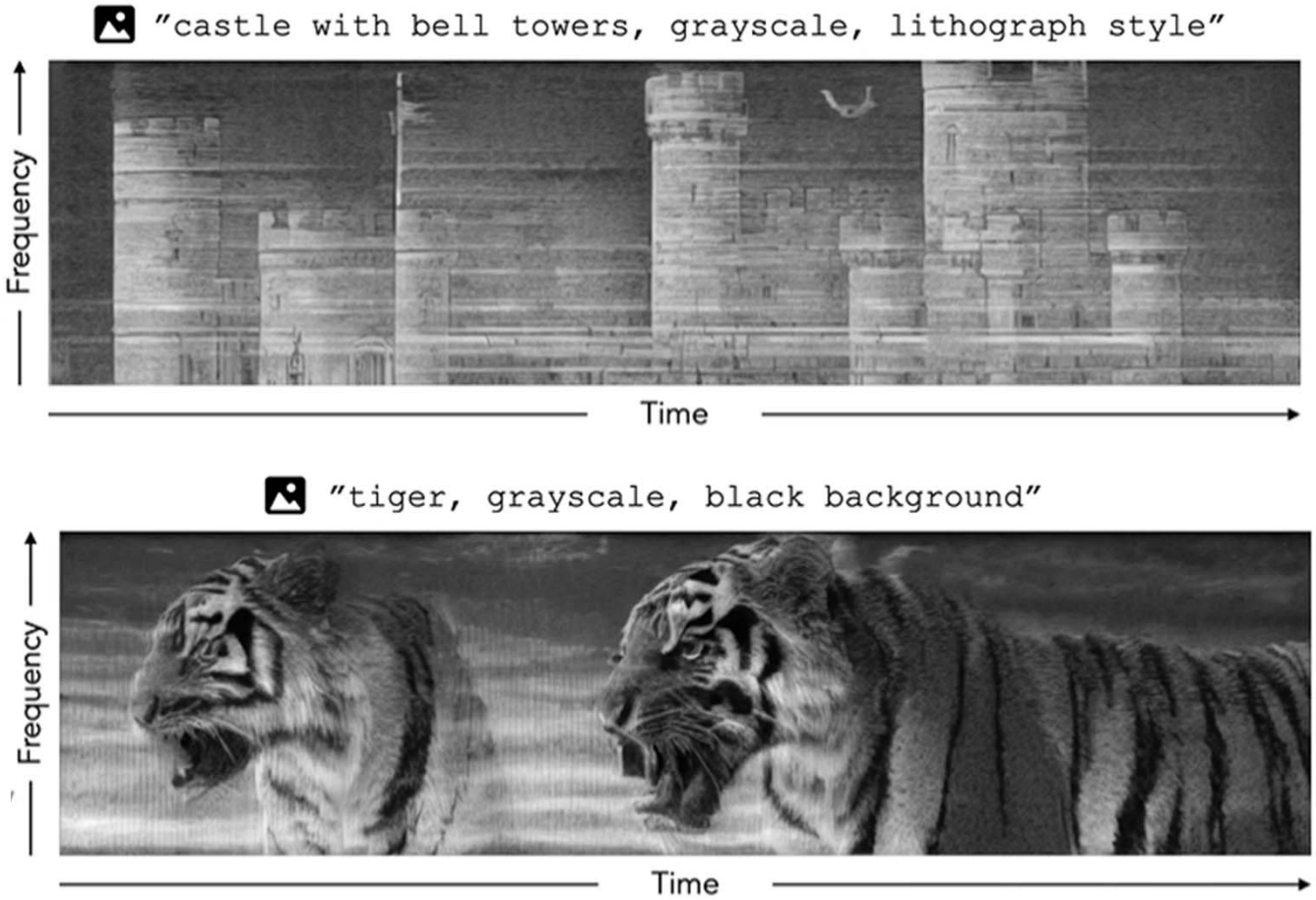
https://www.youtube.com/watch?v=VRAXK4QKJ1Q

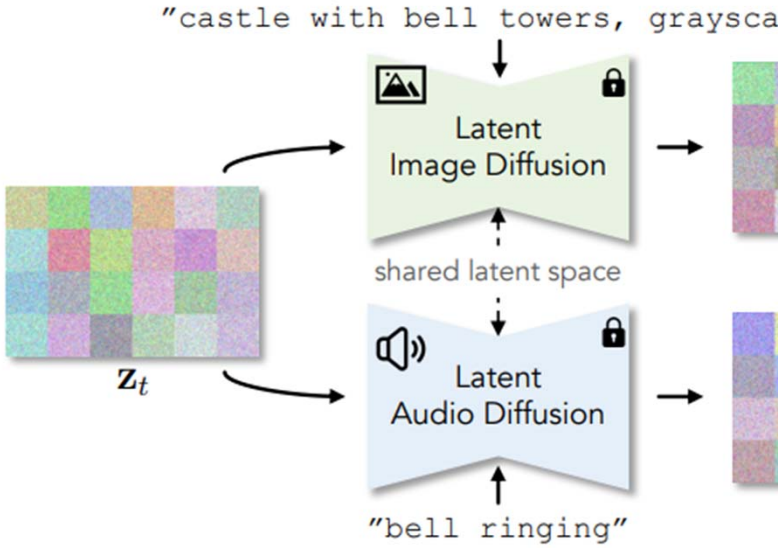


https://www.youtube.com/watch?v=jEO0GHU3xsU

Fun Stuff: “Images that Sound”

https://ificl.github.io/images-that-sound/



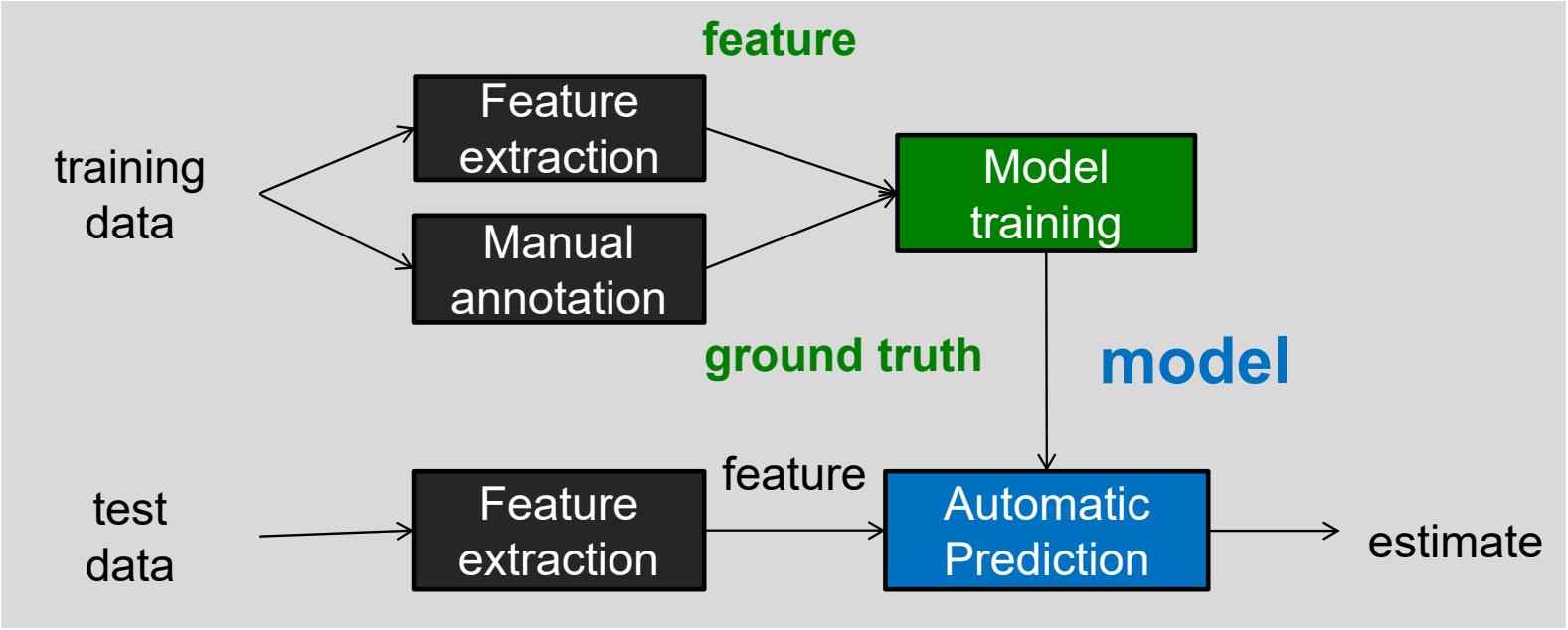


Outline

- Time and frequency representations for music
- **Audio features for music**
- Math in STFT
- DL 101

Representing the Audio as Features is Needed in ML/DL

- Given: $\{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, yN)\}$
  - $\boldsymbol{x}_i \in \mathbb{R}^M$: feature representation
  - $y_i \in \{-1, +1\}$: class label



Library: torchaudio

https://pytorch.org/audio/0.11.0/tutorials/audio_feature_extractions_tutorial.html

```
waveform, sample_rate = get_speech_sample()

n_fft = 1024
win_length = None
hop_length = 512

# define transformation
spectrogram = T.Spectrogram(
    n_fft=n_fft,
    win_length=win_length,
    hop_length=hop_length,
    center=True,
    pad_mode="reflect",
    power=2.0,
)
# Perform transformation
spec = spectrogram(waveform)

print_stats(spec)
plot_spectrogram(spec[0], title="torchaudio")
```

Library: LibROSA

https://librosa.org/doc/latest/index.html

https://colab.research.google.com/github/stevetjoa/musicinformationretrieval.com/blob/gh-pages/ipython_audio.ipynb

```
[ ]  import librosa
     x, sr = librosa.load('audio/simple_loop.wav')
```

```
[ ]  X = librosa.stft(x)
     Xdb = librosa.amplitude_to_db(abs(X))
     plt.figure(figsize=(14, 5))
     librosa.display.specshow(Xdb, sr=sr, x_axis='time', y_axis='hz')
```

Library: LibROSA

https://librosa.org/doc/latest/index.html

Audio loading

- `load` (path, *[, sr, mono, offset, duration, ...])
- `stream` (path, *, block_length, frame_length, ...)
- `to_mono` (y)
- `resample` (y, *, orig_sr, target_sr[, ...])
- `get_duration` (*[, y, sr, S, n_fft, ...])
- `get_samplerate` (path)

Time-domain processing

- `autocorrelate` (y, *[, max_size, axis])
- `lpc` (y, *, order[, axis])
- `zero_crossings` (y, *[, threshold, ...])
- `mu_compress` (x, *[, mu, quantize])
- `mu_expand` (x, *[, mu, quantize])

Signal generation

- `clicks` (*[, times, frames, sr, hop_length, ...])
- `tone` (frequency, *[, sr, length, duration, phi])
- `chirp` (*, fmin, fmax[, sr, length, duration, ...])

Magnitude scaling

- `amplitude_to_db` (S, *[, ref, amin, top_db])
- `db_to_amplitude` (S_db, *[, ref])
- `power_to_db` (S, *[, ref, amin, top_db])
- `db_to_power` (S_db, *[, ref])
- `perceptual_weighting` (S, frequencies, *[, kind])
- `frequency_weighting` (frequencies, *[, kind])
- `multi_frequency_weighting` (frequencies, *[, ...])
- `A_weighting` (frequencies, *[, min_db])
- `B_weighting` (frequencies, *[, min_db])

Library: LibROSA

https://librosa.org/doc/latest/index.html

Time unit conversion

- `frames_to_samples` (frames, *[, hop_length, n_fft])
- `frames_to_time` (frames, *[, sr, hop_length, ...])
- `samples_to_frames` (samples, *[, hop_length, ...])
- `samples_to_time` (samples, *[, sr])
- `time_to_frames` (times, *[, sr, hop_length, n_fft])
- `time_to_samples` (times, *[, sr])
- `blocks_to_frames` (blocks, *, block_length)
- `blocks_to_samples` (blocks, *, block_length, ...)
- `blocks_to_time` (blocks, *, block_length, ...)

Frequency unit conversion

- `hz_to_note` (frequencies, **kwargs)
- `hz_to_midi` (frequencies)
- `midi_to_hz` (notes)
- `midi_to_note` (midi, *[, octave, cents, key, ...])
- `note_to_hz` (note, **kwargs)
- `note_to_midi` (note, *[, round_midi])

Music notation

- `key_to_notes` (key, *[, unicode])
- `key_to_degrees` (key)

Library: LibROSA

https://librosa.org/doc/latest/index.html

Spectral representations

- `stft` (y, *[, n_fft, hop_length, win_length, ...])
- `istft` (stft_matrix, *[, hop_length, ...])
- `cqt` (y, *[, sr, hop_length, fmin, n_bins, ...])
- `icqt` (C, *[, sr, hop_length, fmin, ...])

Harmonics

- `interp_harmonics` (x, *, freqs, harmonics[, ...])
- `salience` (S, *, freqs, harmonics[, weights, ...])
- `f0_harmonics` (x, *, f0, freqs, harmonics[, ...])
- `phase_vocoder` (D, *, rate[, hop_length, n_fft])

Spectral features

- `chroma_stft` (*[, y, sr, S, norm, n_fft, ...])
- `chroma_cqt` (*[, y, sr, C, hop_length, fmin, ...])
- `chroma_cens` (*[, y, sr, C, hop_length, fmin, ...])
- `chroma_vqt` (*[, y, sr, V, hop_length, fmin, ...])
- `melspectrogram` (*[, y, sr, S, n_fft, ...])
- `mfcc` (*[, y, sr, S, n_mfcc, dct_type, norm, ...])
- `rms` (*[, y, S, frame_length, hop_length, ...])
- `spectral_centroid` (*[, y, sr, S, n_fft, ...])

Pitch and tuning

- `pyin` (y, *, fmin, fmax[, sr, frame_length, ...])
- `yin` (y, *, fmin, fmax[, sr, frame_length, ...])
- `estimate_tuning` (*[, y, sr, S, n_fft, ...])

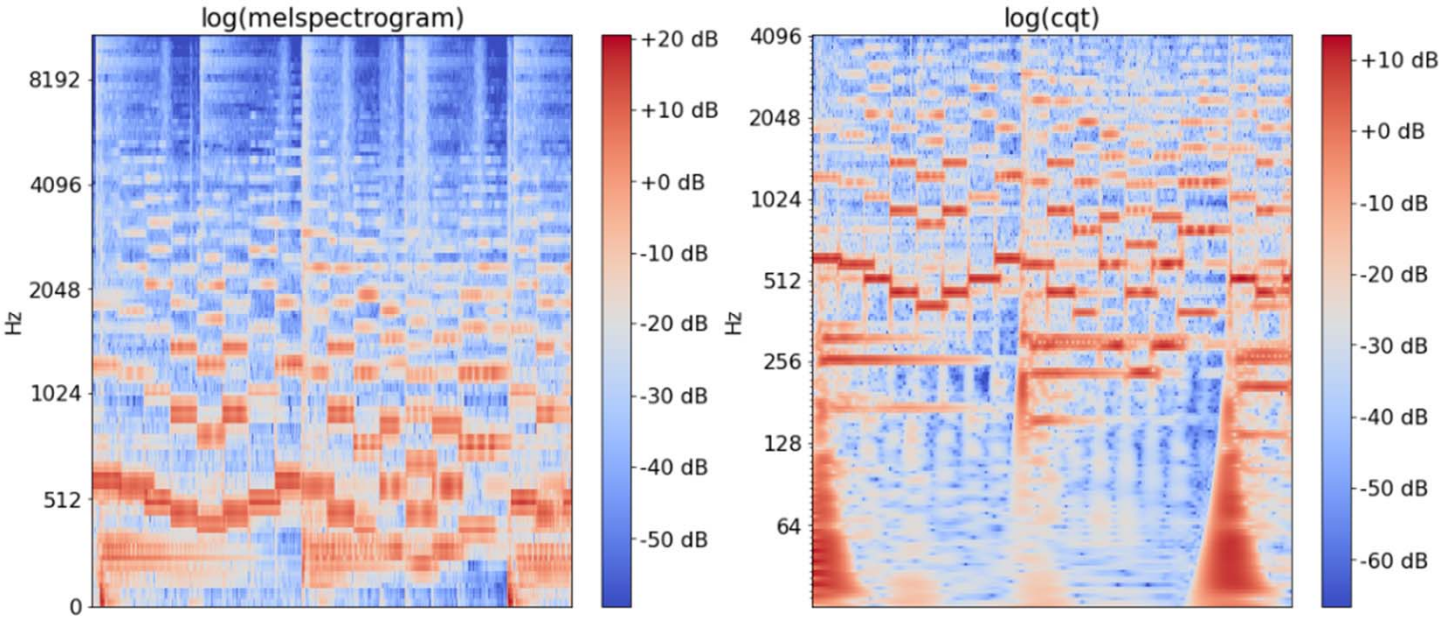
Rhythm features

- `tempo` (*[, y, sr, onset_envelope, tg, ...])
- `tempogram` (*[, y, sr, onset_envelope, ...])

Constant-Q Transform (CQT)

https://music-classification.github.io/tutorial/part2_basics/input-representations.html

- **STFT** (*linearly*-spaced frequencies)
- **CQT** (*logarithmically*-spaced, closer to human auditory perception)

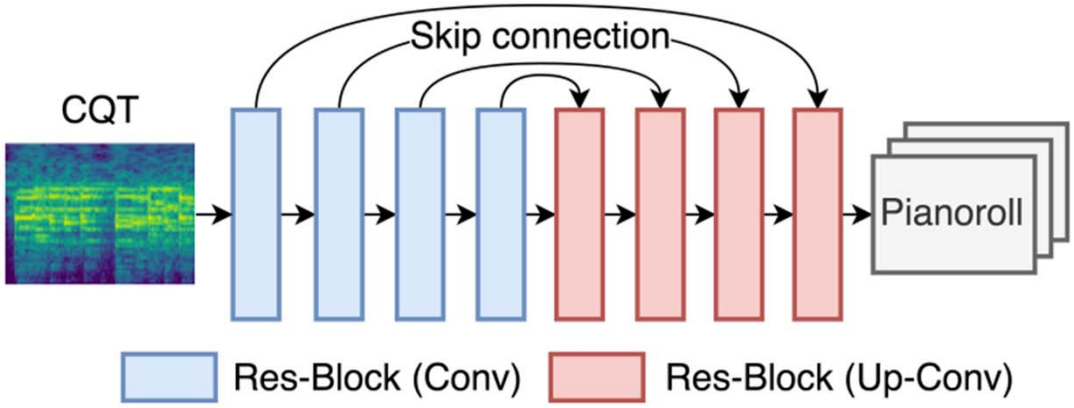


- **Logarithmic** frequency, **logarithmic** magnitude

Constant-Q Transform (CQT)

- Good for **pitch-related** tasks (e.g., multi-pitch estimation or transcription)

Ref: Hung et al, “Multitask learning for frame-level instrument recognition,” ICASSP 2019



CQT - PyTorch

```
from cqt_pytorch import CQT

transform = CQT(
    num_octaves = 8,
    num_bins_per_octave = 64,
    sample_rate = 48000,
    block_length = 2 ** 18
)
```

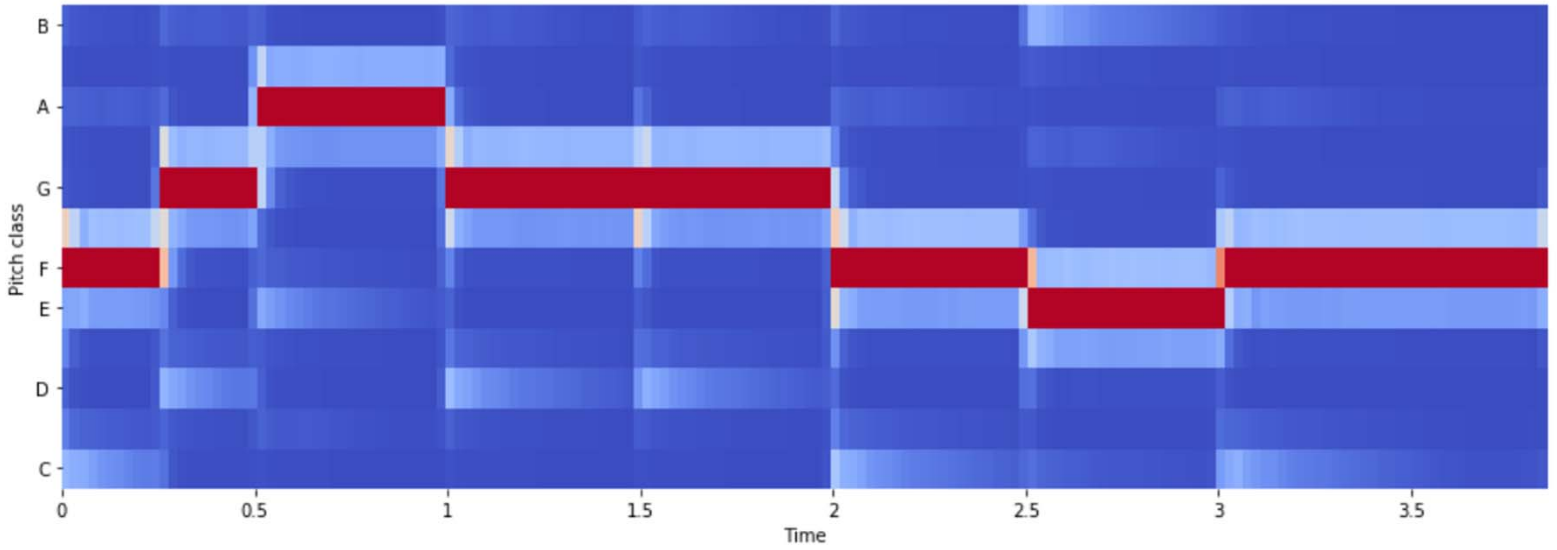
https://github.com/archinetai/cqt-pytorch

https://github.com/eloimoliner/CQT_pytorch

Pitch Class Profile / Chromagram

https://musicinformationretrieval.com/chroma.html

- "A **chroma vector** (Wikipedia) is a typically a 12-element feature vector indicating how much energy of each pitch class, {C, C#, D, D#, E, ..., B}, is present in the signal"
  - i.e., ignore octaves



Pitch Class Profile / Chromagram

- Good for tasks such as **chord recognition** or cover song identification

https://www.audiolabs-erlangen.de/resources/MIR/FMP/C5/C5S2_ChordRec_Templates.html

Ref: Cho et al, “On the relative importance of individual components of chord recognition systems,” TASLP 2014

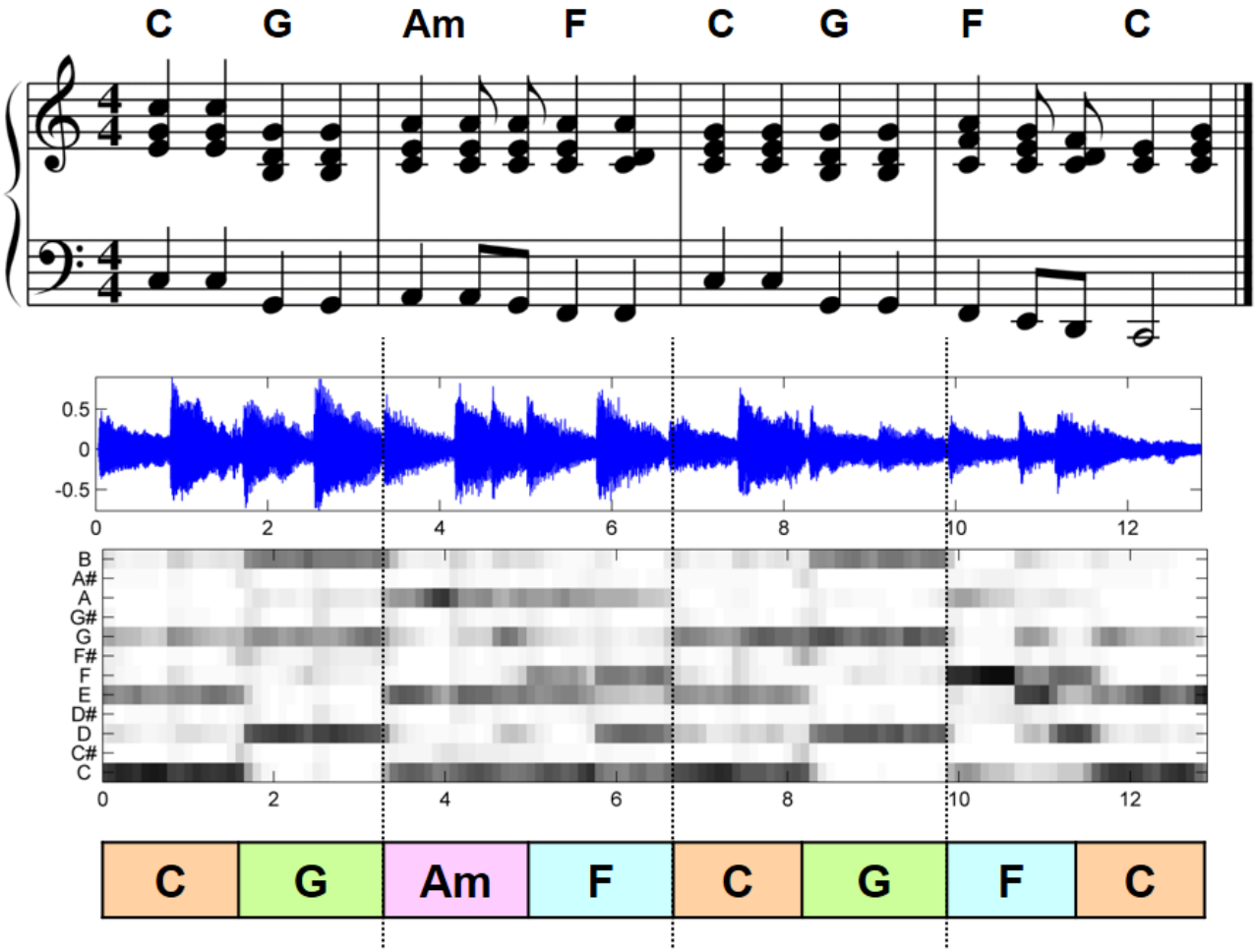
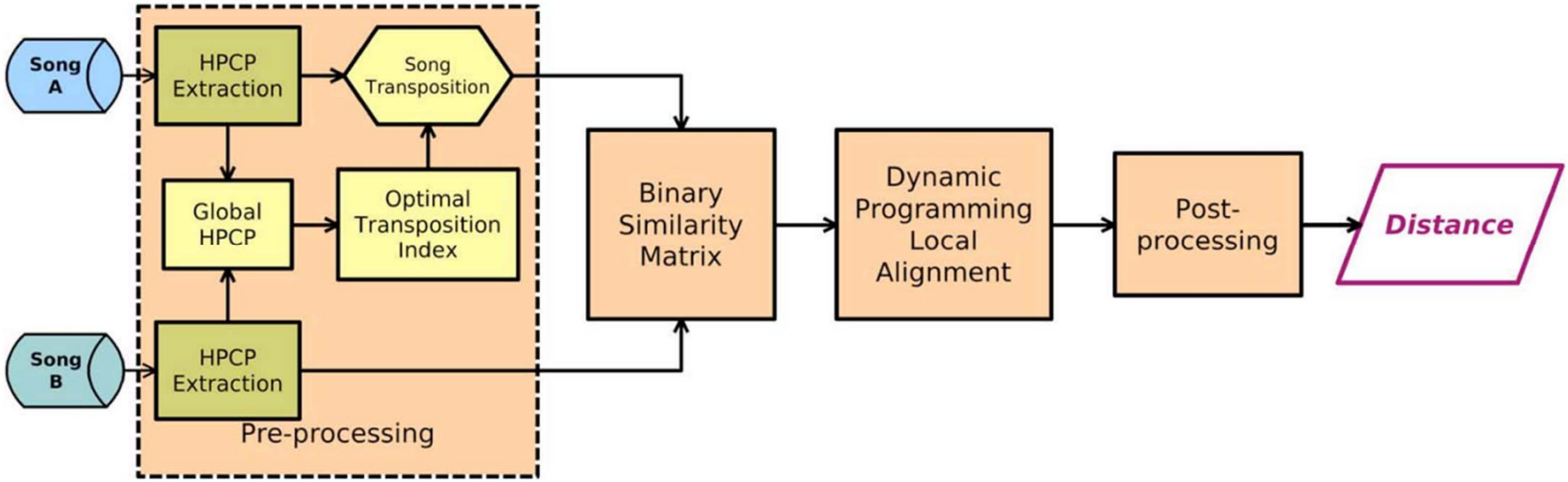


Figure 5.1 from [Müller, FMP, Springer 2015]

Pitch Class Profile / Chromagram

- Good for tasks such as chord recognition or **cover song identification**

https://essentia.upf.edu/tutorial_similarity_cover.html

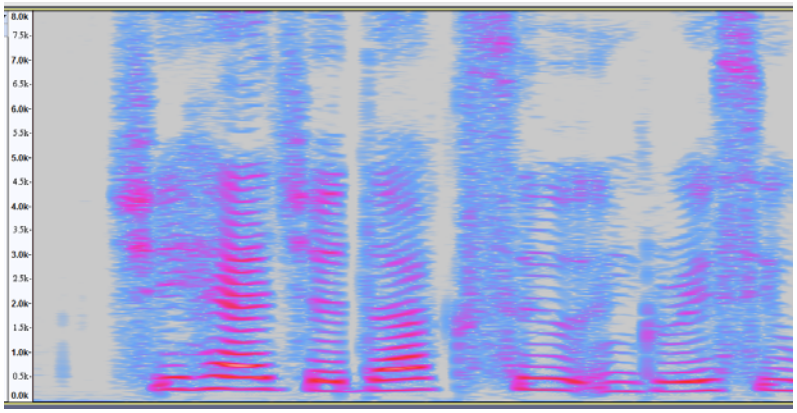


Ref: Serra et al, “Chroma binary similarity and local alignment applied to cover song identification,” TASLP 2008

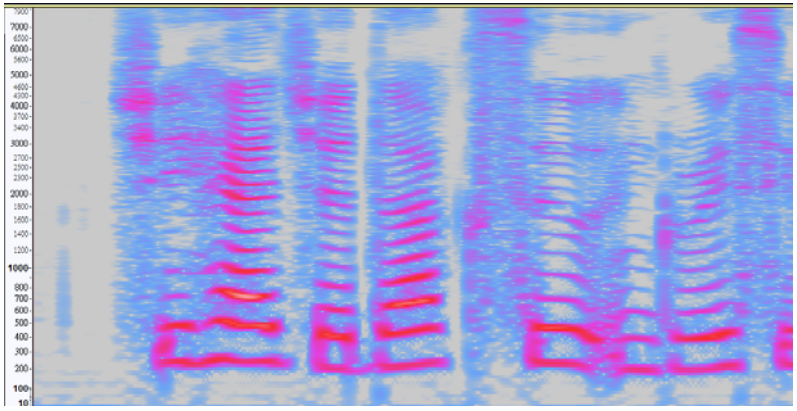
Mel-Spectrogram

- The Mel scale is a perceptual scale of pitches judged by listeners to be equal in distance from one another
- **Finer** resolution in the **low-frequency** range (NOT exactly logarithmic scale)
- Dimension reduction

linear scale

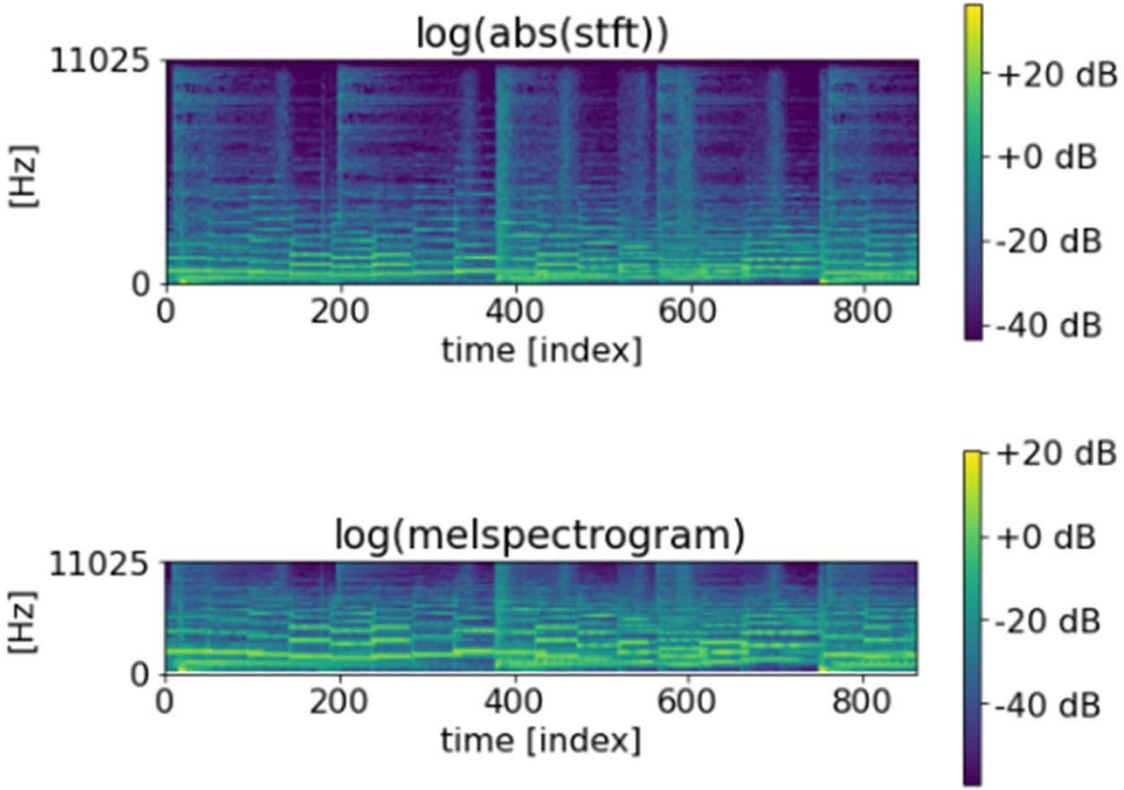


mel scale



Mel-Spectrogram

https://music-classification.github.io/tutorial/part2_basics/input-representations.html



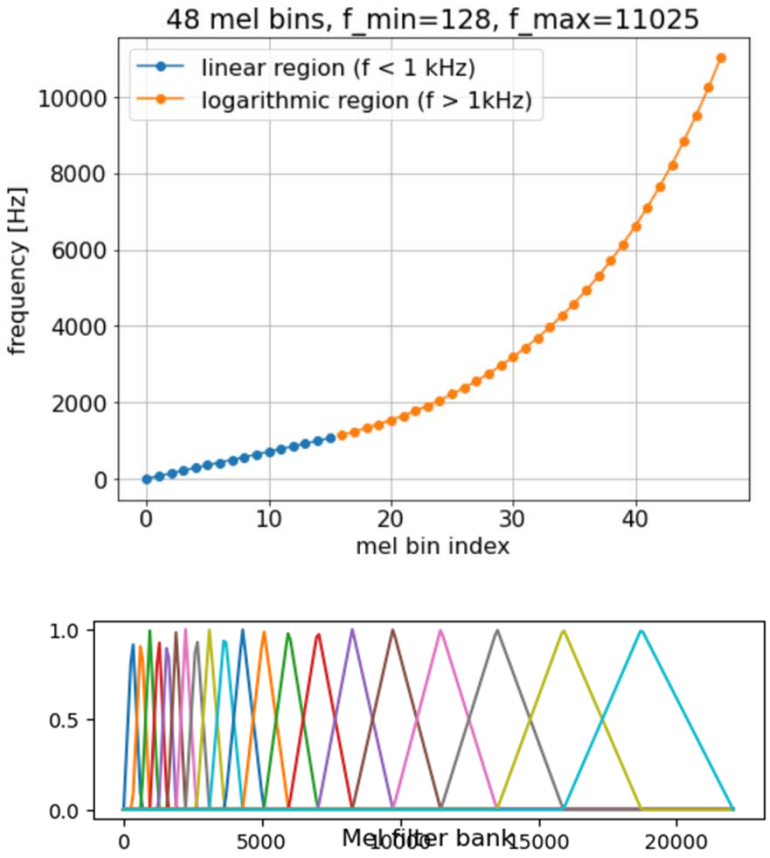
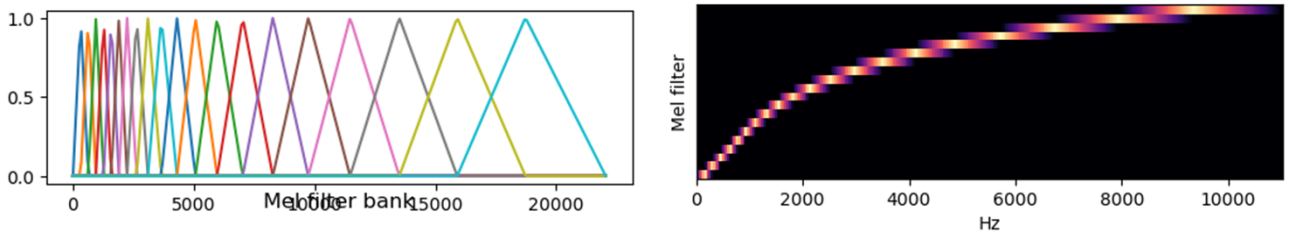


Figure from librosa

Mel-Spectrogram

- Usually treated as the "**default**" feature representation for musical audio, especially in the DL era
  - Easy to compute and understand
  - Reasonably rich information
  - Reasonable size
  - Can be used as input to computer vision (CV) models
  - Possible to go back from mel-spectrograms to waveforms via a "vocoder" (see lecture 5)
- Used in all types of tasks, for both music analysis and generation

Mel-frequency cepstral coefficients (MFCC)



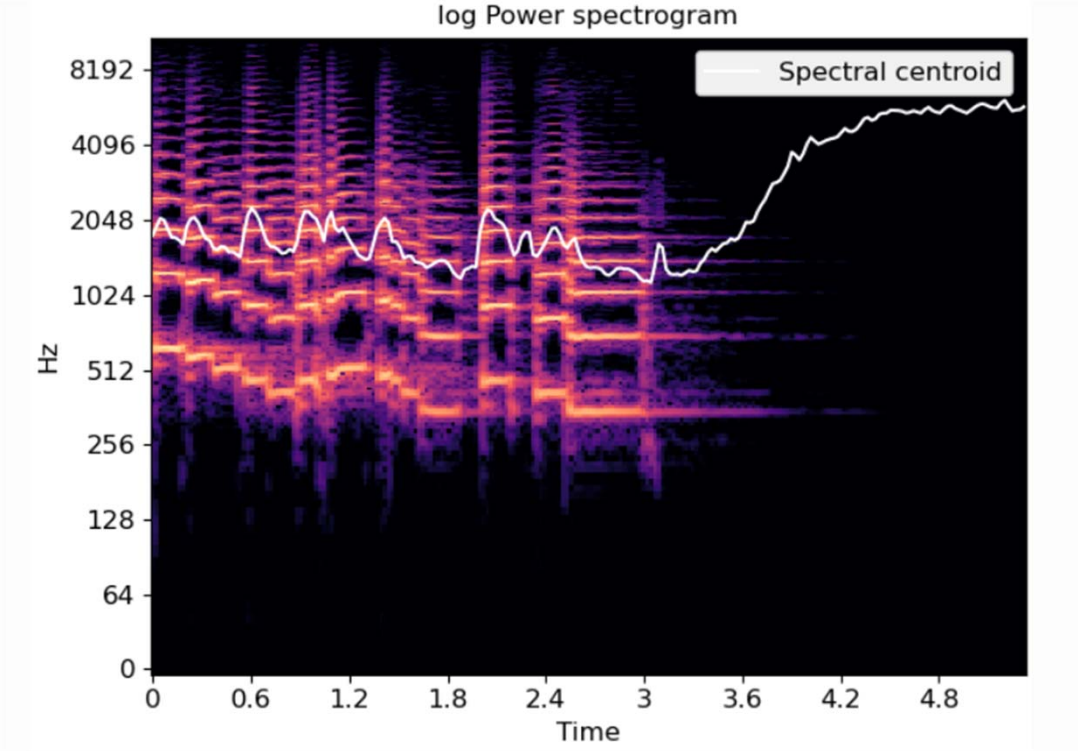
- Procedure
  1. Compute the spectrogram
  2. Grouping the FFT bins according to the *perceptually motivated* Mel-filter bank
     - **Linear** till 1,000 Hz and **logarithmic** above it
     - Usually with triangular overlapping windows
  3. Taking logs and **DCT** for uncorrelating the resulting features

- **Compact representation** of the spectrum (1,024-dim → 128 → 13 coefficients)
  - Somehow capture the energy distribution in the spectrum
  - Less interpretable

Ref: McFee et al., "librosa: Audio and music signal analysis in python," 2015

Spectral Centroid

- Each frame of a magnitude spectrogram is normalized and treated as a distribution over frequency bins, from which the mean (centroid) is extracted per frame
- A measure of spectral shape
- Higher centroid values correspond to "**brighter**" textures with more high frequencies
- Other statistics can also be used
  - bandwidth, skewness, kurtosis



Ref: Tzanetakis and Cook, "Musical genre classification of audio signals," TASLP 2002

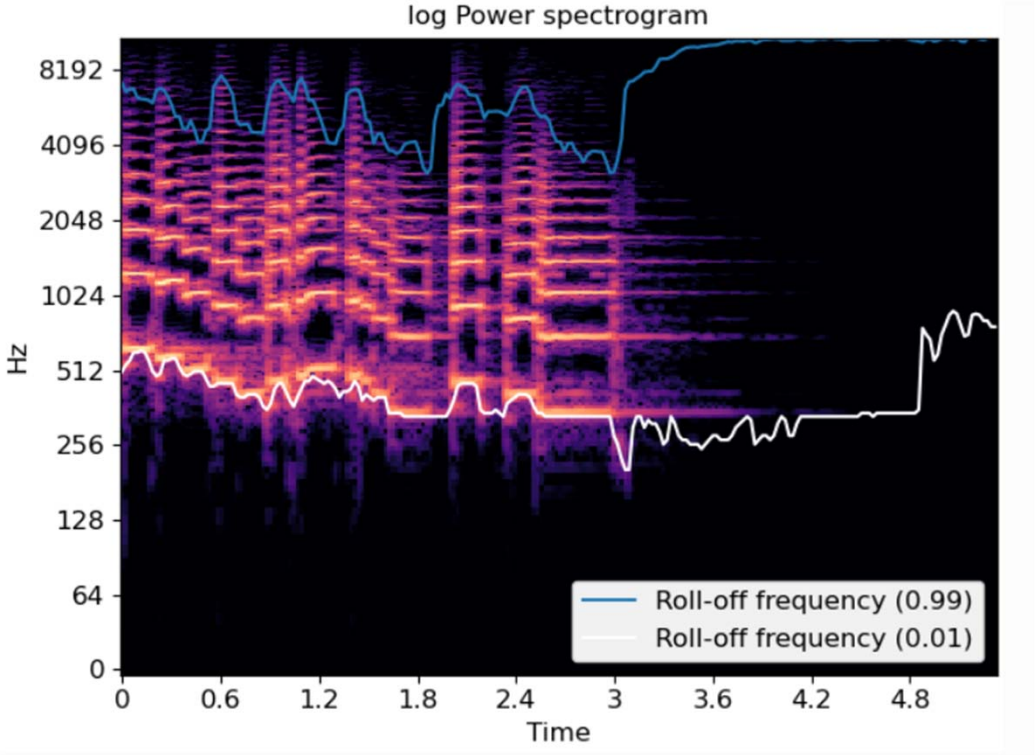
Spectral Contrast

- Each frame of a spectrogram is divided into sub-bands. For each sub-band, the energy contrast is estimated by comparing the mean energy in the top quantile (peak energy) to that of the bottom quantile (valley energy). High contrast values generally correspond to **clear, narrow-band signals**, while low contrast values correspond to **broad-band noise**

Ref: McFee et al., “librosa: Audio and music signal analysis in python,” 2015

Spectral Rolloff

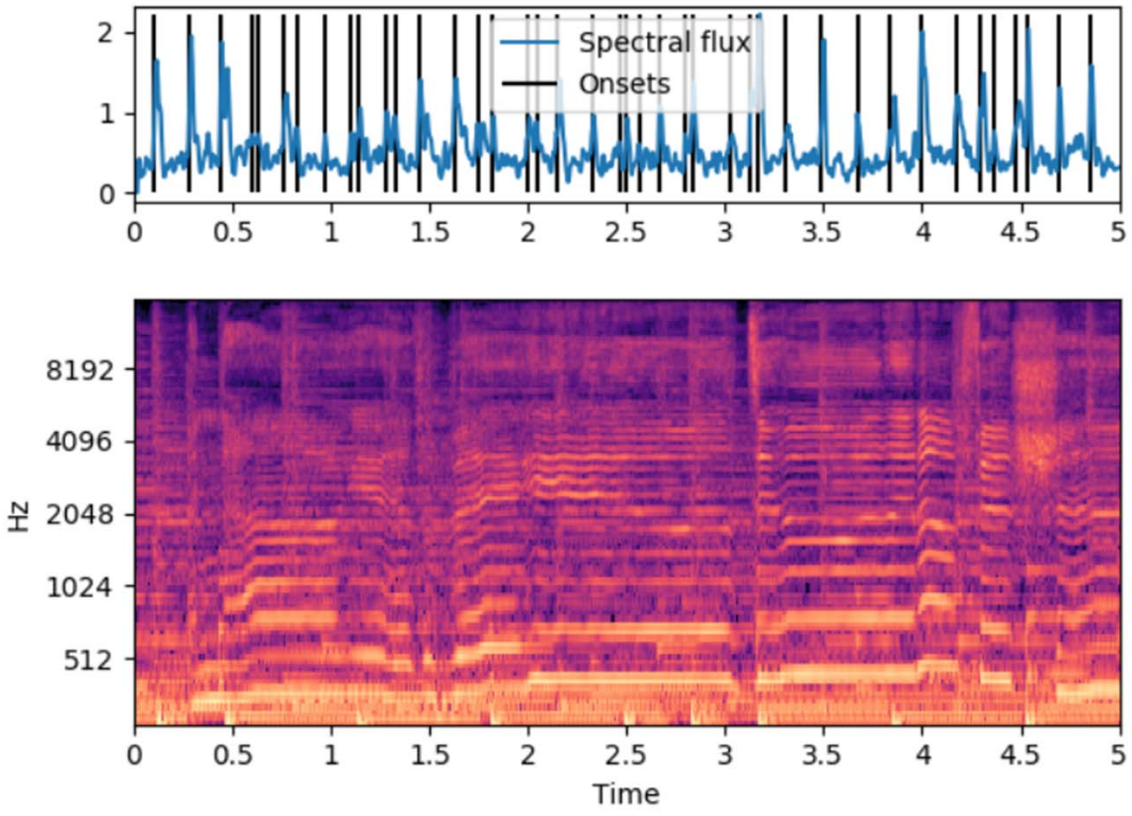
- The frequency for a spectrogram bin such that at least *roll_percent* (0.85 by default) of the **energy** of the spectrum in a frame is contained **in this bin and the bins below**
- Can be used to approximate the **maximum** (or **minimum**) frequency by setting roll_percent to a value close to 1 (or 0)
- Another measure of spectral shape



Ref: McFee et al., "librosa: Audio and music signal analysis in python," 2015

Spectral Flux

- How quickly the power spectrum of a signal is changing over time
- Usually calculated as the L2-norm between two adjacent normalized spectra
- Usually used for musical onset detection (more related to rhythm than to timbre)

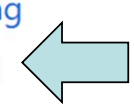


Ref1: https://en.wikipedia.org/wiki/Spectral_flux
Ref2: https://librosa.org/librosa_gallery/auto_examples/plot_superflux.html

Demonstrations 1

https://colab.research.google.com/github/stevetjoa/musicinformationretrieval.com/blob/gh-pages/spectral_features.ipynb

Signal Analysis and Feature Extraction



Demonstrations 2

http://www.ifs.tuwien.ac.at/~schindler/lectures/MIR_Feature_Extraction.html

```
def spectral_centroid(wavedata, window_size, sample_rate):

    magnitude_spectrum = stft(wavedata, window_size)

    timebins, freqbins = np.shape(magnitude_spectrum)

    # when do these blocks begin (time in seconds)?
    timestamps = (np.arange(0,timebins - 1) * (timebins / float(sample_rate)))

    sc = []

    for t in range(timebins-1):

        power_spectrum = np.abs(magnitude_spectrum[t])**2

        sc_t = np.sum(power_spectrum * np.arange(1,freqbins+1)) / np.sum(power_spectrum)

        sc.append(sc_t)


    sc = np.asarray(sc)
    sc = np.nan_to_num(sc)

    return sc, np.asarray(timestamps)
```

Library 1: LibROSA

Spectral features

`melspectrogram` (*[, y, sr, S, n_fft, ...])	Compute a mel-scaled spectrogram.
`mfcc` (*[, y, sr, S, n_mfcc, dct_type, norm, ...])	Mel-frequency cepstral coefficients (MFCCs)
`rms` (*[, y, S, frame_length, hop_length, ...])	Compute root-mean-square (RMS) value for each frame, either from t or from a spectrogram `S`.
`spectral_centroid` (*[, y, sr, S, n_fft, ...])	Compute the spectral centroid.
`spectral_bandwidth` (*[, y, sr, S, n_fft, ...])	Compute p'th-order spectral bandwidth.
`spectral_contrast` (*[, y, sr, S, n_fft, ...])	Compute spectral contrast
`spectral_flatness` (*[, y, S, n_fft, ...])	Compute spectral flatness
`spectral_rolloff` (*[, y, sr, S, n_fft, ...])	Compute roll-off frequency.

Ref: McFee et al., “librosa: Audio and music signal analysis in python,” 2015

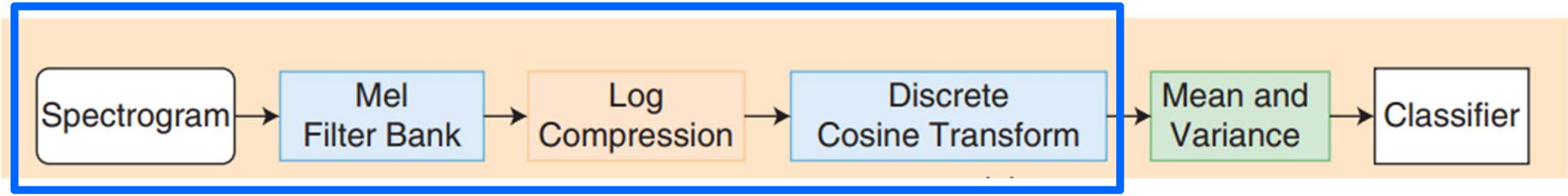
Library 2: Audio Commons Audio Extractor

https://github.com/AudioCommons/ac-audio-extractor

JSON output example


```
"duration": 6.0,
"lossless": 1.0,
"codec": "pcm_s16le",
"bitrate": 705600.0,
"samplerate": 44100.0,
"channels": 1.0,
"audio_md5": "8da67c9c2acbd13998c9002aa0f60466",
"loudness": -28.207069396972656,
"dynamic_range": 0.6650657653808594,
"temporal_centroid": 0.5078766345977783,
"log_attack_time": 0.30115795135498047,
"filesize": 529278,
"single_event": false,
"tonality": "G# minor",
"tonality_confidence": 0.2868785858154297,
"loop": true,
"tempo": 120,
"tempo_confidence": 1.0,
"note_midi": 74,
"note_name": "D5",
"note_frequency": 592.681884765625,
"note_confidence": 0.0,
"brightness": 50.56954356039029,
"depth": 13.000903137777897,
"metallic": 0.4906048209174263,
"roughness": 0.7237051954207928,
"genre": "Genre B",
"mood": "Mood B"
```

Spectral Features Can be Used to Build a Classifier in ML



- Procedure
  1. Compute the features per STFT frame (e.g., 13-D frame-level features)
  2. **Temporal pooling** over time for each audio clip (e.g., by taking the **mean and variance**; leading to 26-D clip-level features)
  3. Use that as input to a **classifier** (e.g., random forest, or support vector machine)
- Limits
  - *Clear* physical meaning but not sophisticated enough and *limited* **semantic** meaning
  - Only the classifier is *trainable*; the features are hand-crafted

“Feature Learning” in DL

(the blocks inside the black lines are learned)

(a) Feature engineering

(b) Low-level feature learning

(c) Convolution neural networks

(d) End-to-end learning

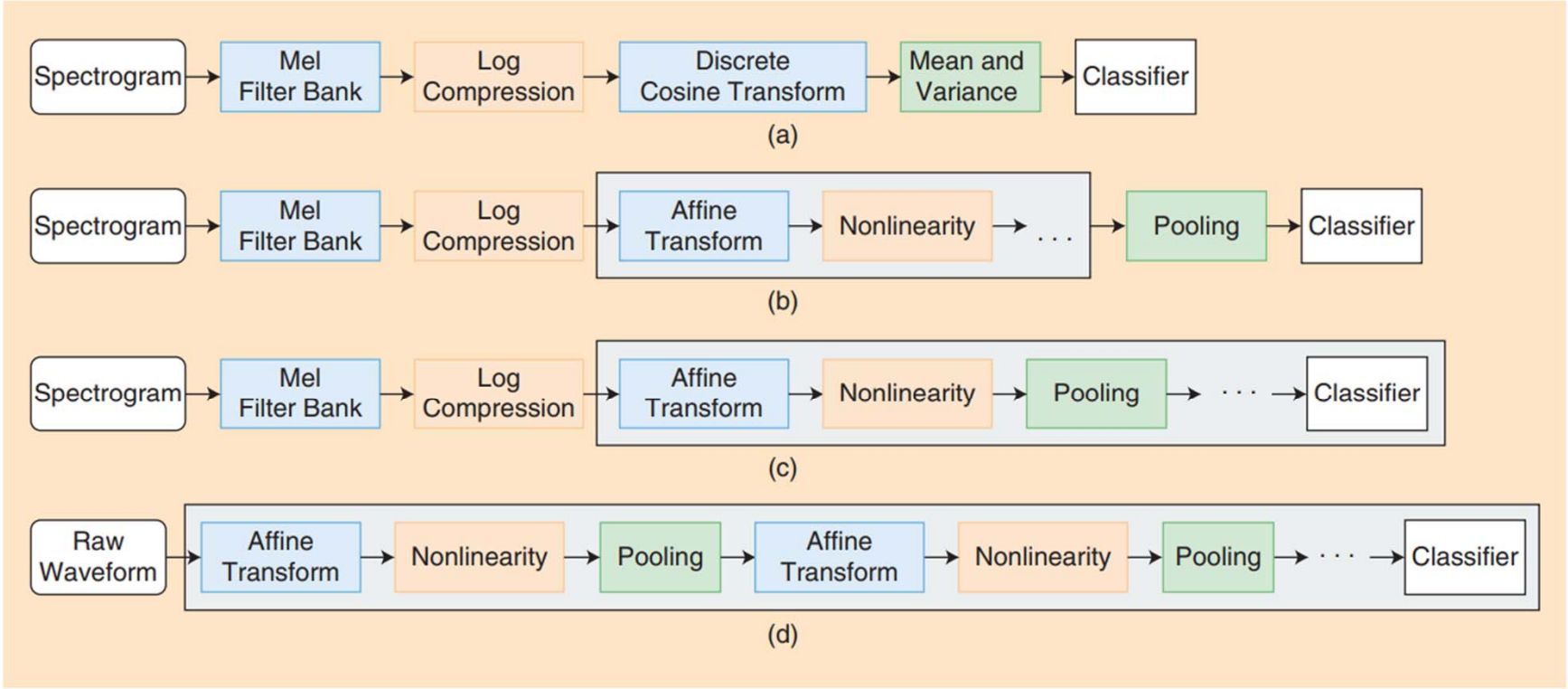
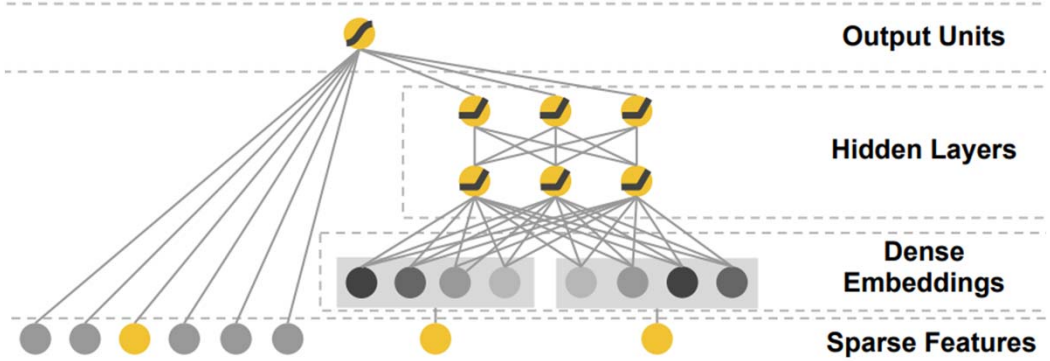


FIGURE 1. The transition of feature representation for music classification: (a) feature engineering [mel-frequency cepstral coefficients (MFCCs)], (b) low-level feature learning, (c) convolutional neural networks, and (d) end-to-end learning. The blocks inside the black lines indicate that they are learned by the algorithms.

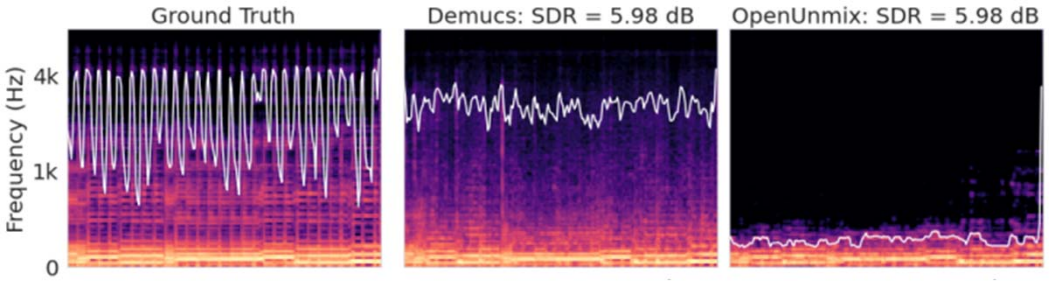
Ref: Nam et al., “Deep learning for audio-based music classification and tagging,” IEEE Signal Processing Magazine, 2019

The Use of Hand-crafted Features in DL

- Hand-crafted features
  - Clear physical meaning
  - Interpretable
- Used as input to music classifiers in the early days
- Can be used alongside learned features (though not popular)
  - The “deep & wide” architecture
- Can be used as objective metrics or loss functions for DL models



Ref: Cheng et al., “Wide & deep learning for recommender systems,” DLRS 2016



Ref: Schaffer et al., “Music separation enhancement with generative modeling,” ISMIR 2022

Different Features are Needed for Different Tasks

- **Timbre representation**: Spectrogram → **mel-spectrogram** → MFCC
- **Harmonic representation**: Spectrogram → **CQT** → chroma feature

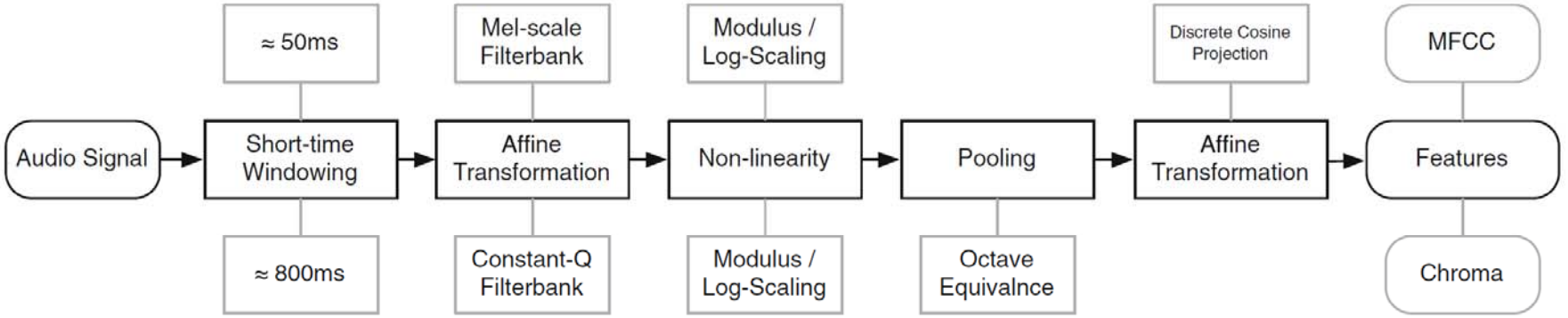
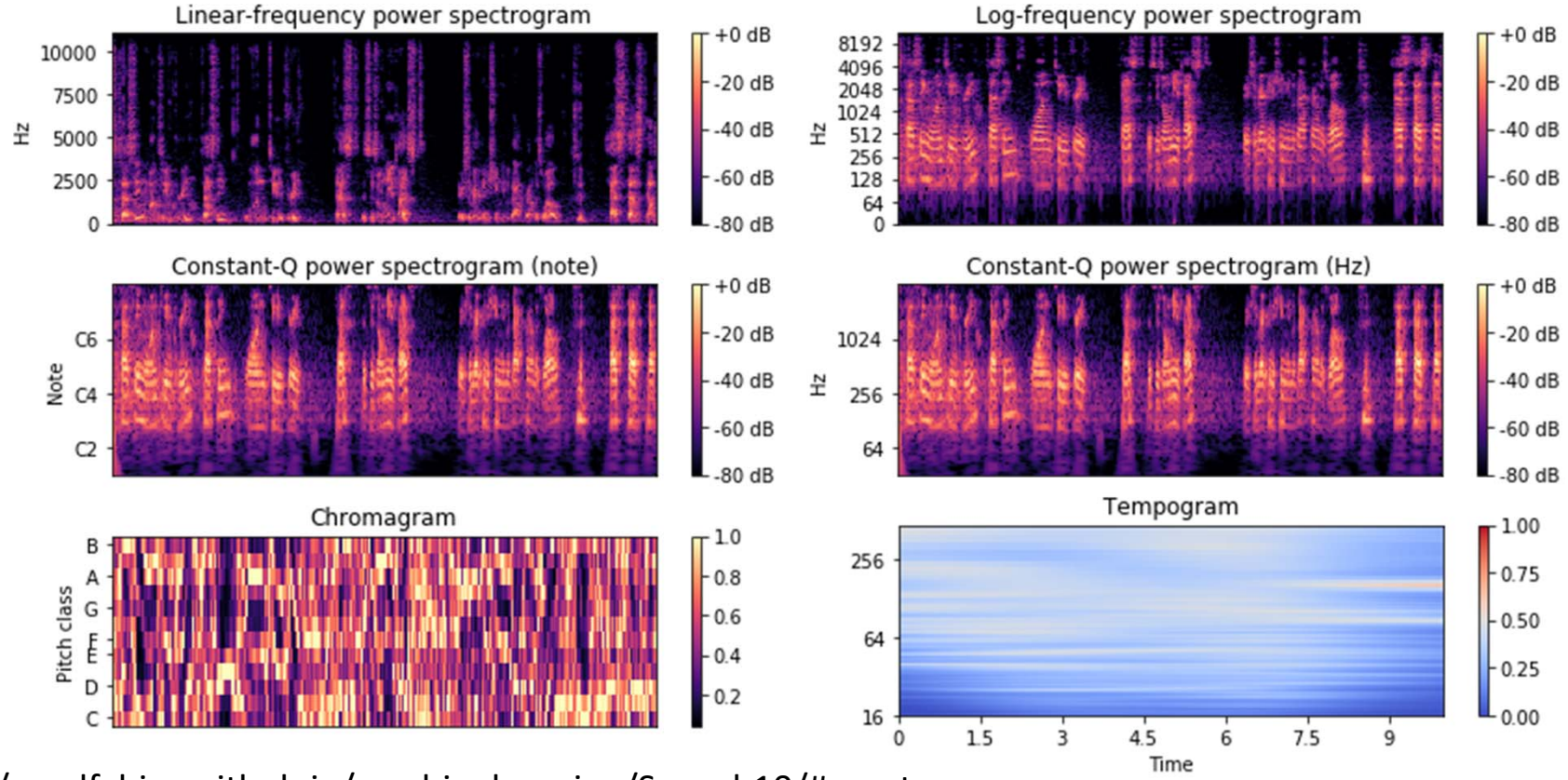


Fig. 3 *State of the art*: standard approaches to feature extraction proceed as the cascaded combination of a few simpler operations; on closer inspection, the main difference between chroma and MFCCs is the parameters used

Ref: Humphrey et al, "Feature learning and deep architectures: new directions for music informatics," JIIS 2013

Different Features are Needed for Different Tasks

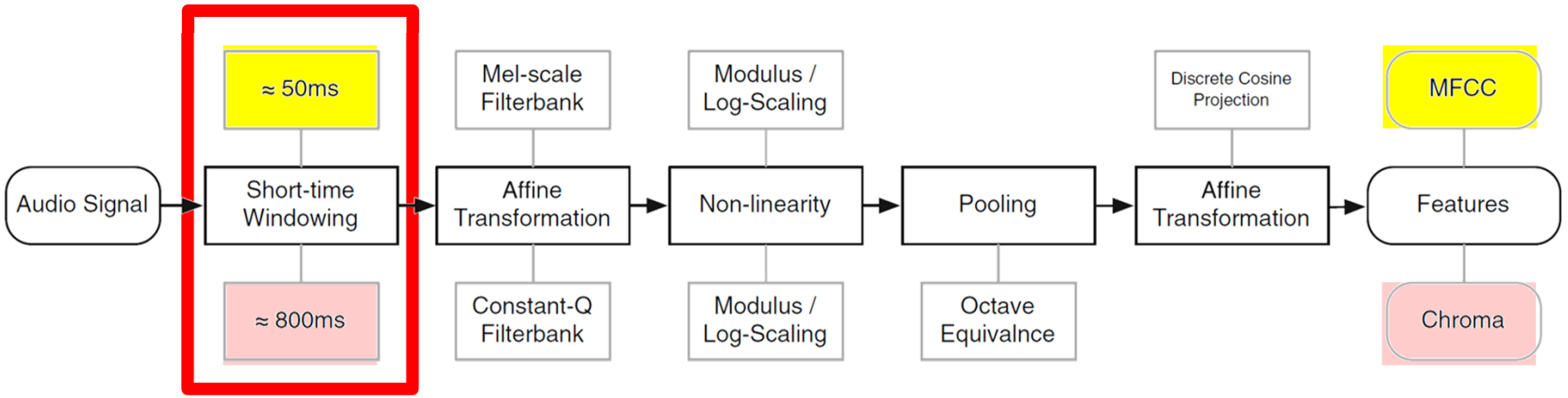
- We can **combine** different features to train a classifier for tasks such as music *genre* classification or music *emotion* classification



https://maelfabien.github.io/machinelearning/Speech10/#spectrogram

Different Features are Needed for Different Tasks

- **Timbre representation**: Spectrogram → **mel-spectrogram** → MFCC
- **Harmonic representation**: Spectrogram → **CQT** → chroma feature



- Wait… why the window size is different?

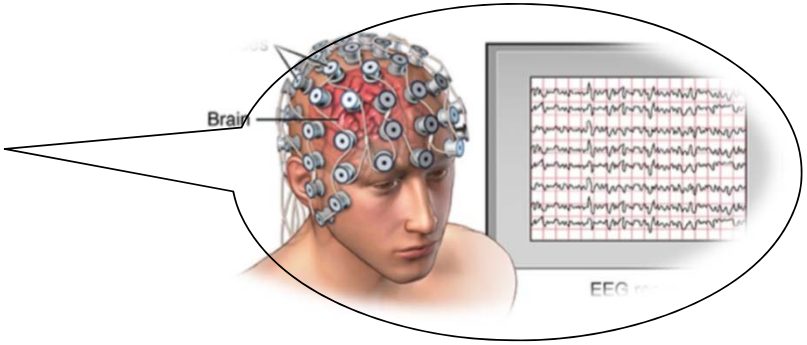
Ref: Humphrey et al, "Feature learning and deep architectures: new directions for music informatics," JIIS 2013

Outline

- Time and frequency representations for music
- Audio features for music
- **Math in STFT**
- DL 101

Sampling Rate

- Definition: number of samples per second
- Why: analog to digital
- Examples
  - EEG signal: **128** Hz
  - Telephone audio: **8k** Hz
  - Music audio: **44k** HZ

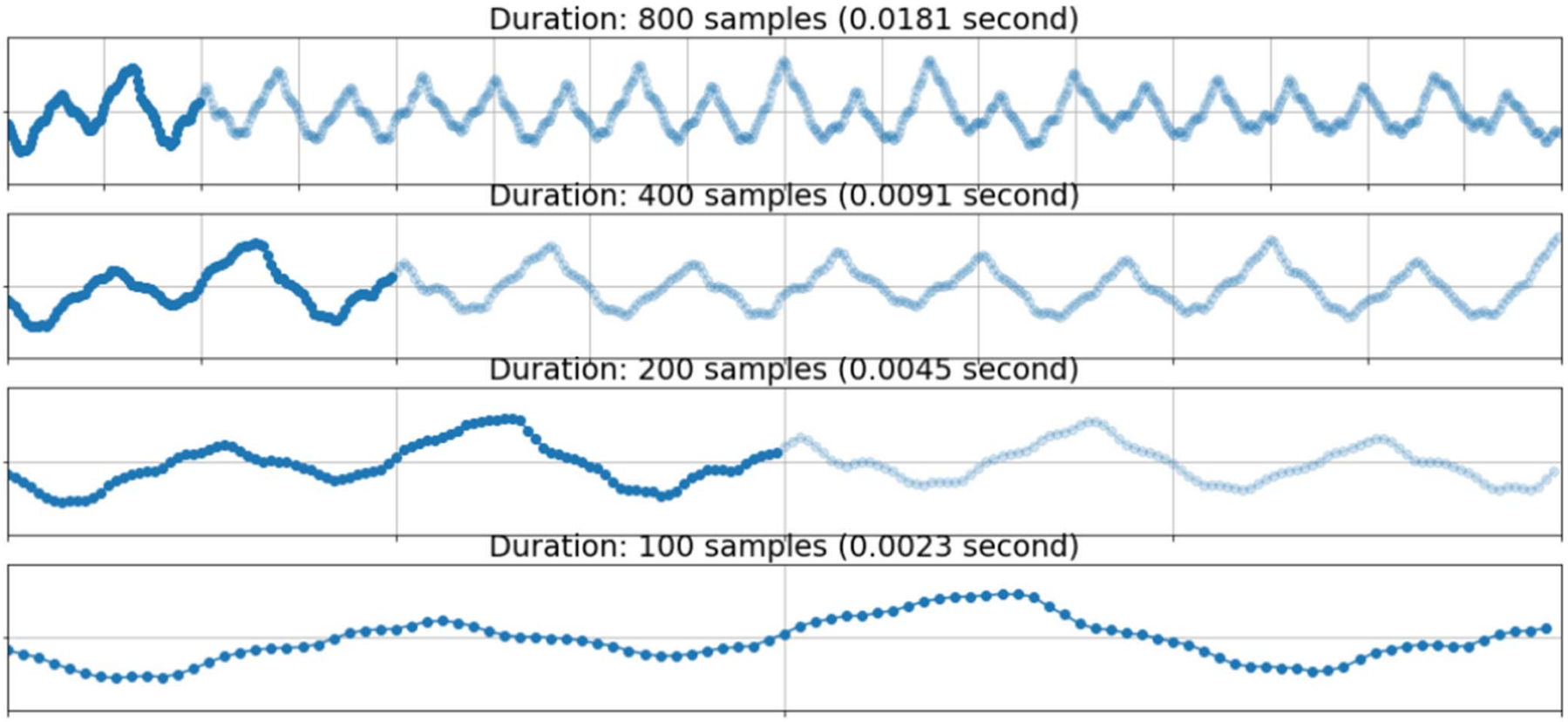


https://www.brightbraincentre.co.uk/electroencephalogram-eeg-brainwaves/

```
[ ]  import  librosa
     x,  sr  =  librosa.load('audio/simple_loop.wav')
```

```
waveform, sample_rate = get_speech_sample()
```

Sampling Rate



https://music-classification.github.io/tutorial/part2_basics/input-representations.html

Sampling Rate

- Usually, a model considered signals with a **certain and fixed** sampling rate
  - The “x”s with different sampling rates are not on the same time basis

```
x, sr = librosa.load('audio/simple_loop.wav')
```

  - Make sure your data are under the same sampling rate
  - Double check the sampling rate assumed by the models you sourced from GitHub
    - Speech: 16k Hz
    - Music: 22k, 44k, or 48k Hz
  - If we want to ML/DL model trained on signals with one sampling rate to process a signal under a different sampling rate, “**resampling**” of that signal is needed

Resampling

Library	Time on CPU (ms)
soxr (HQ)	7.2
scipy.signal.resample	13.4
soxr (VHQ)	15.8
torchaudio	19.2

https://github.com/dofuuz/python-soxr

```
y = soxr.resample(
    x,          # 1D(mono) or 2D(frames, channels) array input
    48000,      # input samplerate
    16000       # target samplerate
)
```

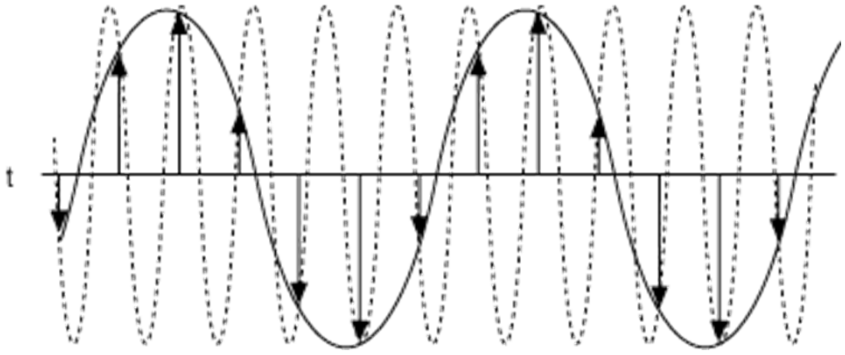
Sample Rate independent Models

- Usually, a model considered signals with a certain and fixed sampling rate
- Building a **sampling-rate agnostic** model that can take signals sampled at arbitrary sampling rates is an advanced topic

Ref 1: Saito et al, “Sampling-frequency-independent audio source separation using convolution layer based on impulse invariant method,” EUSIPCO 2021
Ref 2: Carson et al, “Sample rate independent recurrent neural networks for audio effects processing,” DAFx 2024

Nyquist–Shannon Sampling Theorem

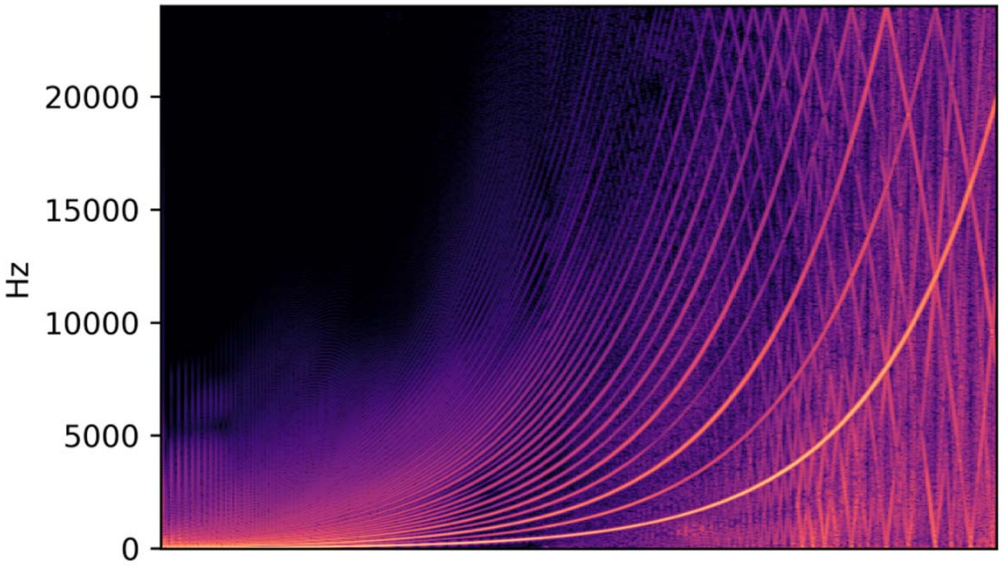
- A signal must be sampled at least **twice** as fast as the bandwidth of the signal to accurately reconstruct the waveform; otherwise, the high-frequency content will alias at a frequency inside the spectrum of interest
- **Sampling freq (Fs) > 2* the highest freq in the signal**



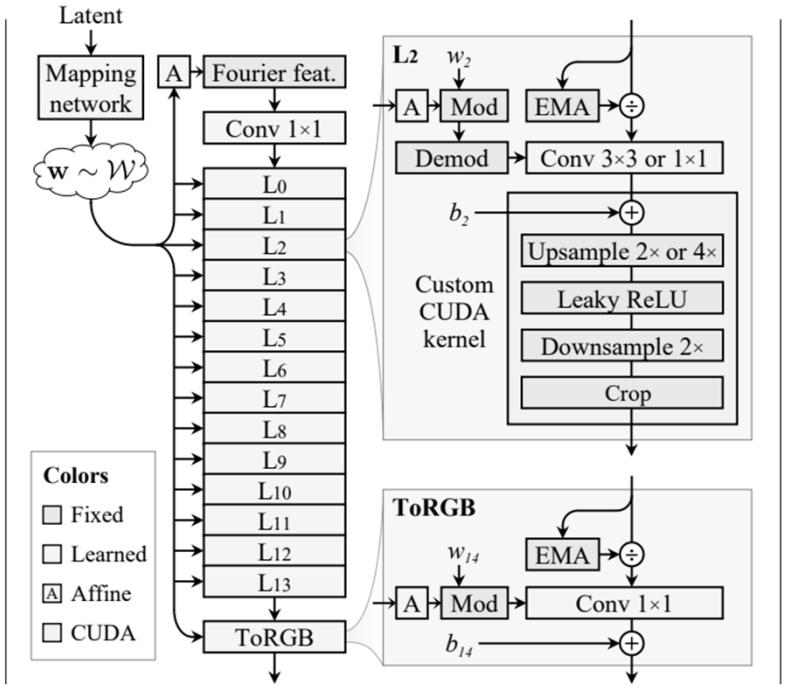
http://zone.ni.com/reference/en-XX/help/370524T-01/siggenhelp/fund_nyquist_and_shannon_theorems/

Aliasing

- High-frequency components fold back



Ref 1: Yeh et al, "PyNeuralFx: A Python package for neural audio effect modeling," ISMIR-LBD 2024



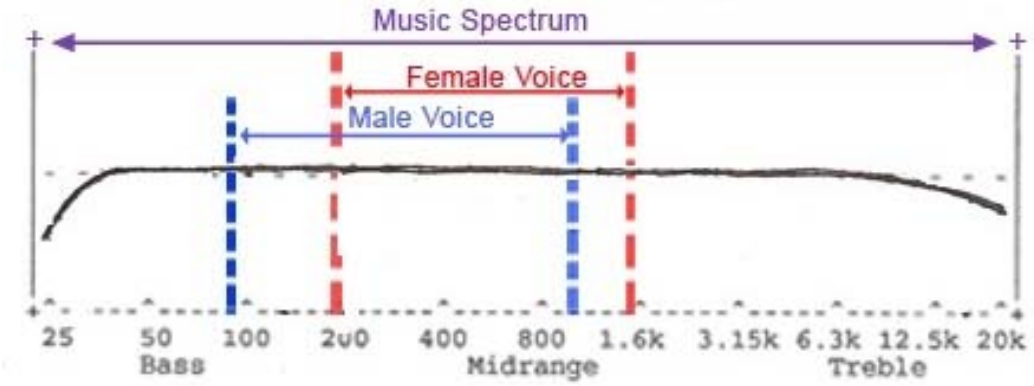
(b) Our alias-free StyleGAN3 generator architecture

https://nvlabs.github.io/stylegan3/

Ref 2: Karras et al, "Alias-free generative adversarial networks," NeurIPS 2021

Nyquist–Shannon Sampling Theorem

- Telephone audio: 8k Hz
  - Via phone, we cannot hear frequency higher than 4k Hz



https://www.quora.com/How-do-HRT-sex-reassignment-and-other-such-procedures-affect-vocal-production-particularly-the-singing-voice

- **Question**: With Fs=128 Hz, we assume that we don't need to care about frequency higher than __ Hz in brain waves

Nyquist–Shannon Sampling Theorem

Brainwaves, Frequencies and Functions

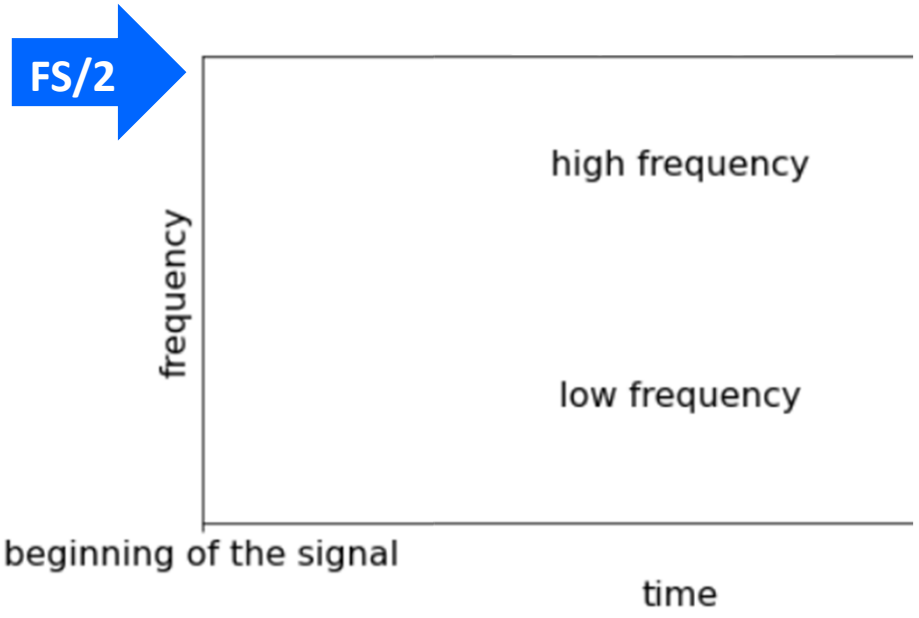
Unconscious		Conscious		
Delta	**Theta**	**Alpha**	**Beta**	**Gamma**
0,5 – 4 Hz	4 – 8 Hz	8 – 13 Hz	13 – 30 Hz	30-42 Hz
Instinct	**Emotion**	**Consciousness**	**Thought**	**Will**
Survival Deep sleep Coma	Drives Feelings Trance Dreams	Awareness of the body Integration of feelings	Perception Concentration Mental activity	Extreme focus Energy Ecstasy

http://altered-states.net/barry/update236/

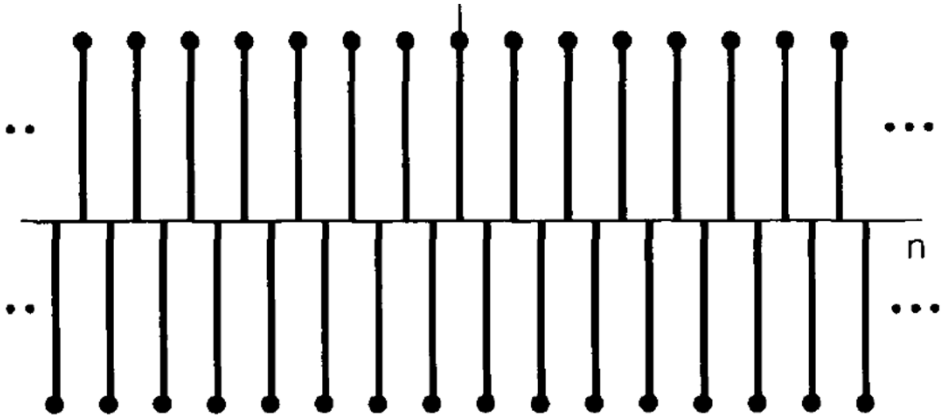
- **Question**: With Fs=128 Hz, we assume that we don't need to care about frequency higher than 64 Hz in brain waves

Nyquist–Shannon Sampling Theorem

- Sampling freq (Fs) > 2* the highest freq in the signal
- **Highest freq bin in STFT = Fs/2**



Sinusoid with the highest freq given a fixed FS



DTFS: Fourier Series Representation for Discrete-time Periodic Signals

- For **periodic** signals with fundamental **period L**, i.e., $x[n] = x[n+L]$
- The Fourier series coefficients a_k can be interpreted as the inner product of the signal $x[n]$ and a basis function $e^{jk\frac{2\pi}{L}n}$, which is complex-valued
- a_k are **complex-valued**
- a_k are also **periodic with period L**, i.e., $a_k = a_{k+L}$

Synthesis equation: $x[n] = \sum_{k=\langle L \rangle} a_k e^{jk\frac{2\pi}{L}n}$

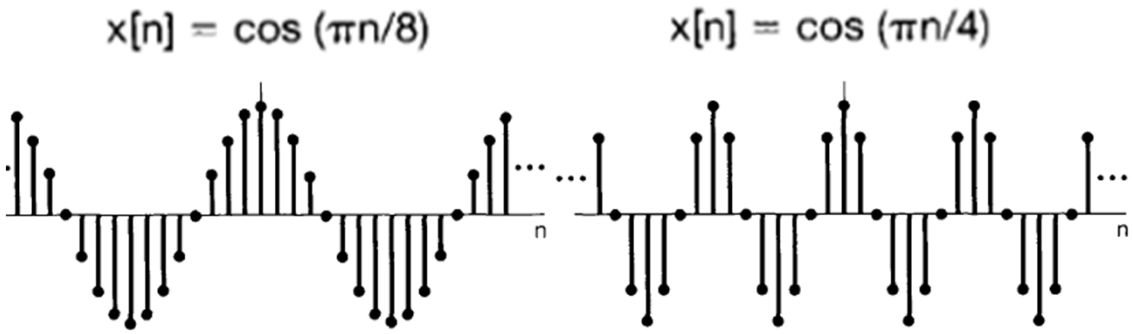
Analysis equation: $a_k = \frac{1}{L}\sum_{n=\langle L \rangle} x[n] e^{-jk\frac{2\pi}{L}n}$

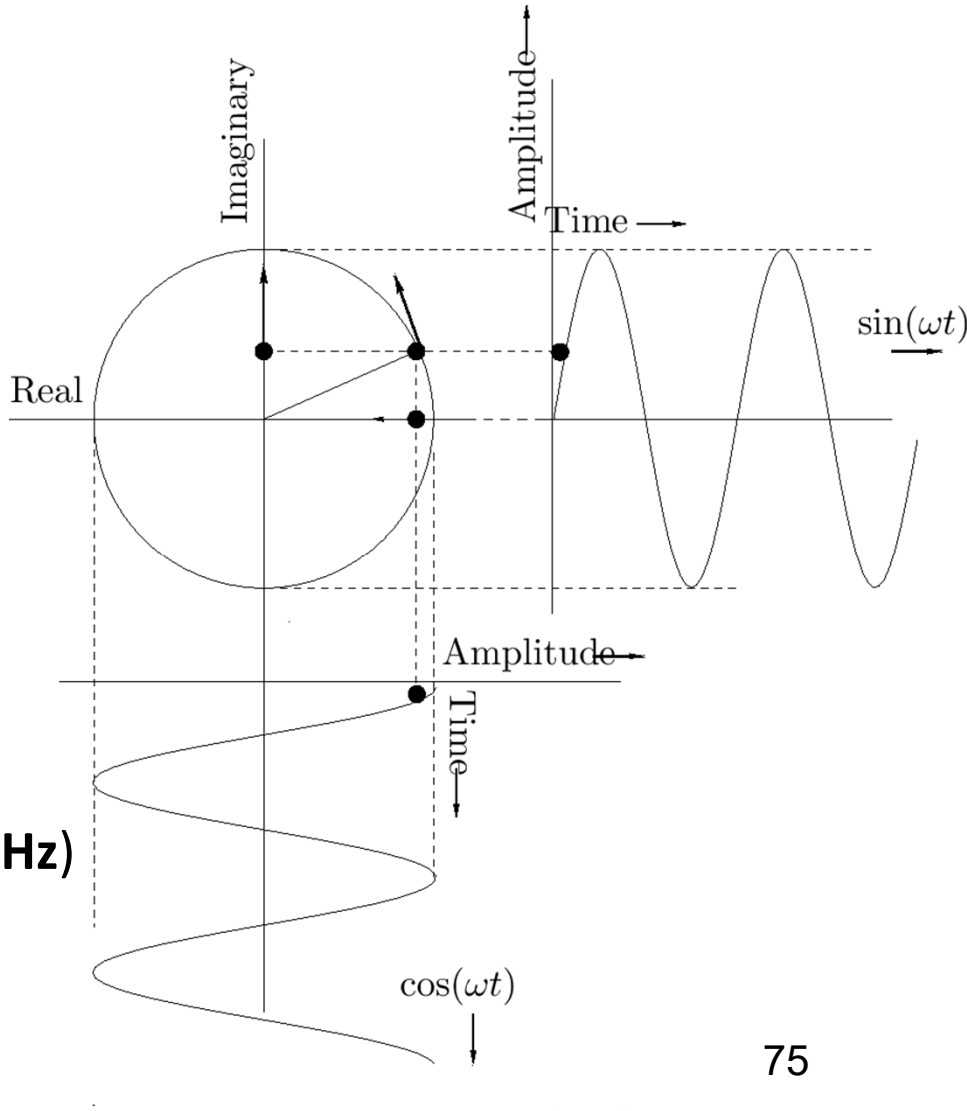
In discrete-time, $e^{jk\omega n} = e^{j(k+L)\omega n}$; so there are **only L unique basis vectors**

Sum over a set of L successive integers

The Basis Functions in DTFS

- $e^{j\omega n} = \cos(\omega n) + j\sin(\omega n)$





- angular or ordinary frequency
  - ω: **angular frequency** (radians per second)
  - f: **ordinary frequency** (circles per second; in **Hz**)
  - $e^{j2\pi fn} = \cos(2\pi fn) + j\sin(2\pi fn)$

Discrete-time Fourier Transform (DFT): Extending DTFS to Deal with Aperiodic, Finite-Duration DT Signals

- For **periodic DT** signal $x[n] \overset{\mathcal{FS}}{\leftrightarrow} a_k$

Synthesis: $x[n] = \sum_{k=\langle L \rangle} a_k e^{jk\omega n}$

Analysis: $a_k = \frac{1}{N} \sum_{n=\langle L \rangle} x[n] e^{-jk\omega n}$

- For **aperiodic, finite-duration DT** signal $x[n] \overset{\mathcal{DFT}}{\longleftrightarrow} X_k$

Synthesis: $x[n] = \sum_{k=0}^{L-1} X_k e^{jk\omega n}$

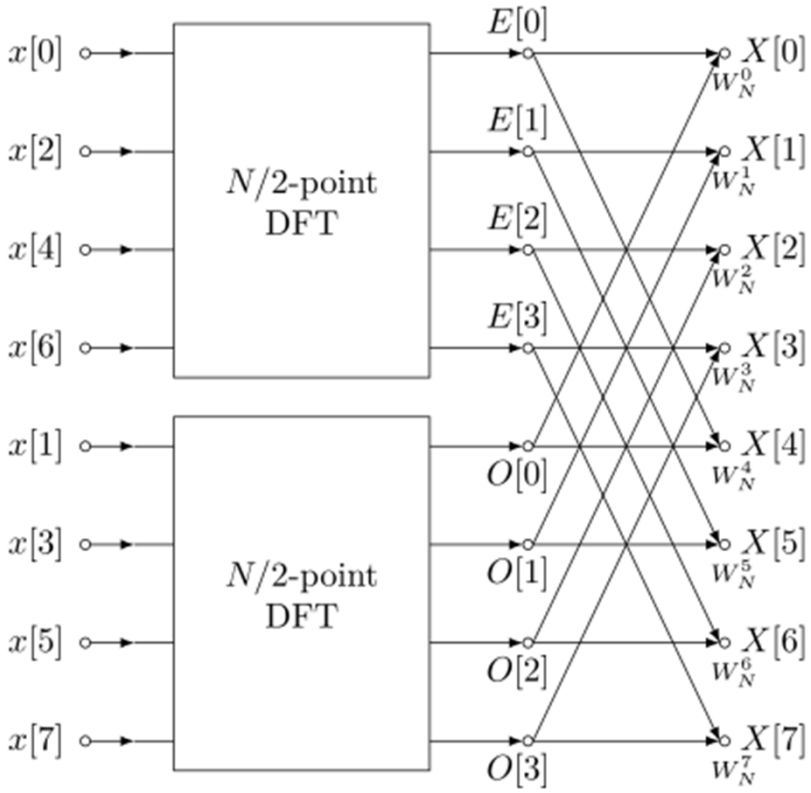
Analysis: $X_k = \frac{1}{L} \sum_{n=0}^{L-1} x[n] e^{-jk\omega n}$

NOTE: While the length of $x[n]$ is N, the length of X_k is also L

DFT and FFT (Fast Fourier Transform)

https://www.audiolabs-erlangen.de/resources/MIR/FMP/C2/C2_DFT-FFT.html

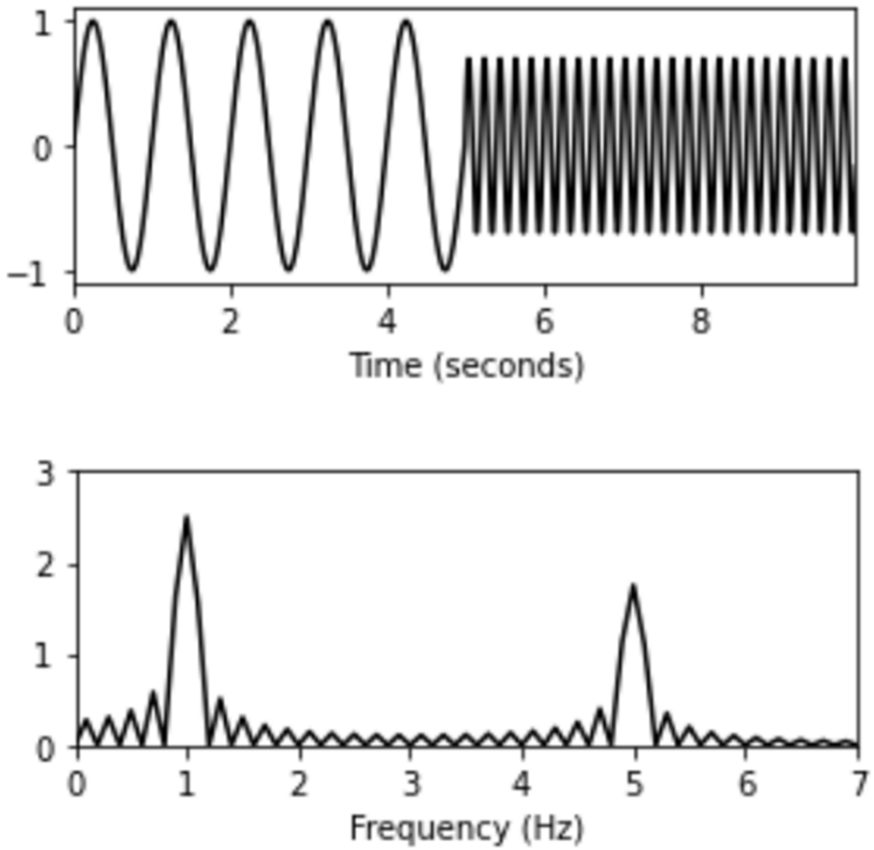
- FFT is a fast algorithm to compute the DFT



https://en.wikipedia.org/wiki/Fast_Fourier_transform

FFT cannot Localize Temporal Events

https://www.audiolabs-erlangen.de/resources/MIR/FMP/C2/C2_STFT-Basic.html



STFT: Short-time Fourier Transform

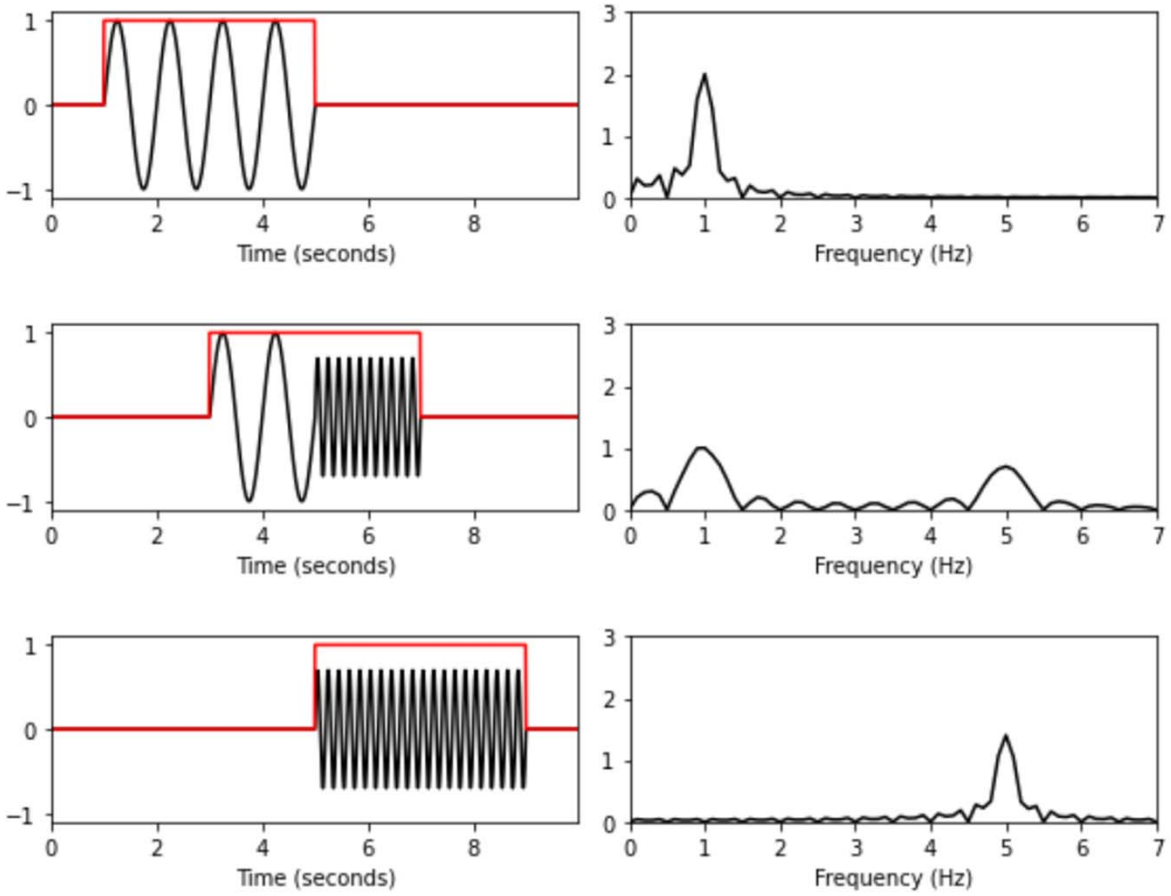
https://www.audiolabs-erlangen.de/resources/MIR/FMP/C2/C2_STFT-Basic.html

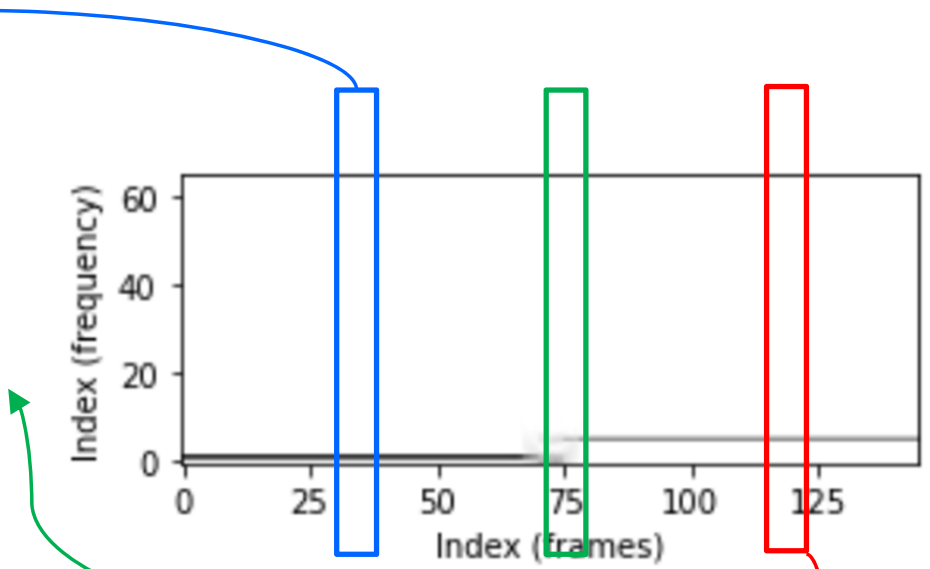
$$\mathcal{X}(m,k) := \sum_{n=0}^{N-1} x(n+mH)w(n)\exp(-2\pi ikn/N)$$

- $x[n]$: real-valued discrete-time signal of length *L* obtained by equidistant sampling with respect to a fixed sampling rate
- $w[n]$: window function of length $N < L$
- H: "hop size"

Windowed FFT

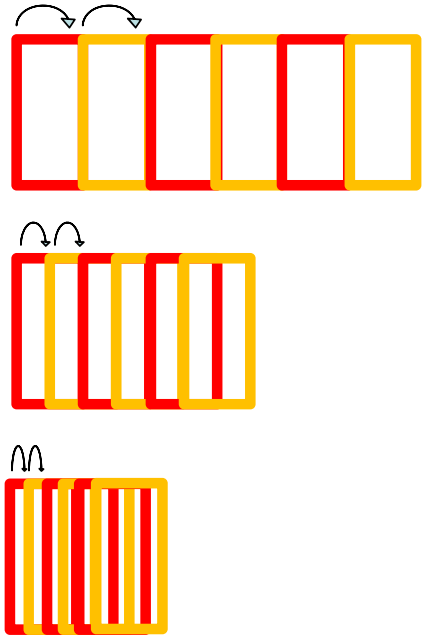
https://www.audiolabs-erlangen.de/resources/MIR/FMP/C2/C2_STFT-Basic.html





Hop Size in STFT

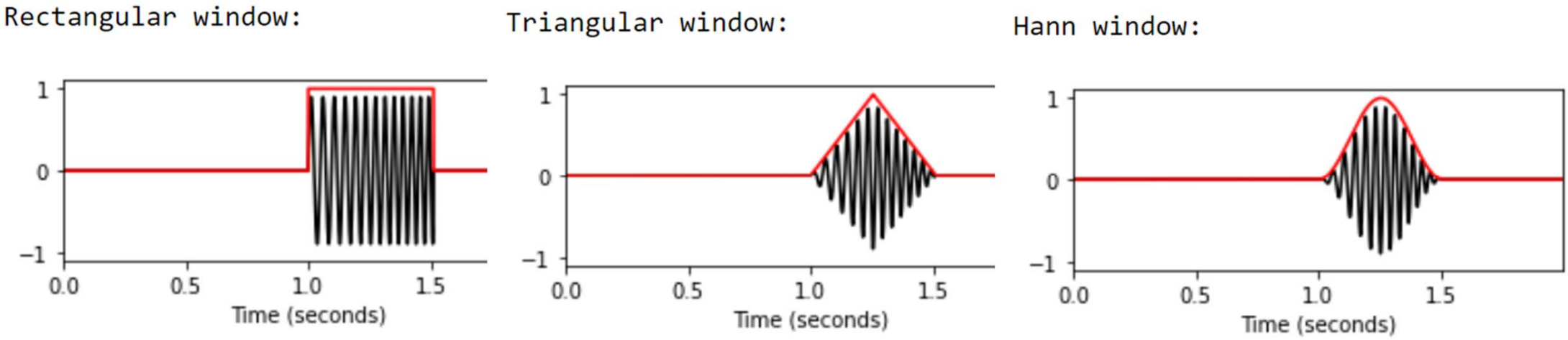
- Hop size
  - Can be proportional to the window size
    - hop_size = win_size
    - hop_size = 0.5*win_size
    - hop_size = 0.1*win_size
  - Can also be not proportional to the window size



Window Functions in STFT

https://www.audiolabs-erlangen.de/resources/MIR/FMP/C2/C2_STFT-Window.html

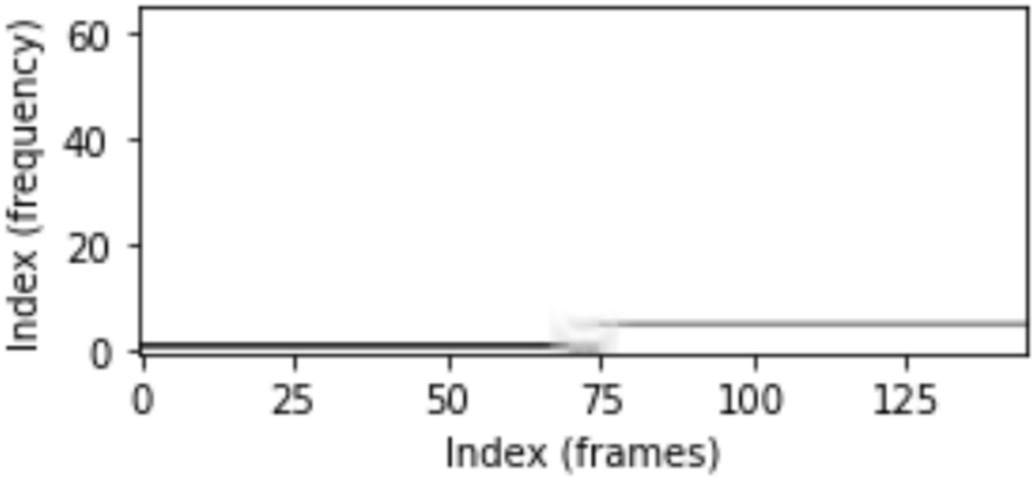
- Hann window is often used



Interpretation of Time and Frequency Indices in STFT

https://www.audiolabs-erlangen.de/resources/MIR/FMP/C2/C2_STFT-Basic.html

$$F_{\mathrm{coef}}(k) := \frac{k \cdot F_{\mathrm{s}}}{N}$$



- **Frequency resolution**: $\frac{F_s}{N}$
- **Temporal resolution**: $\frac{H}{F_s}$

$$T_{\mathrm{coef}}(m) := \frac{m \cdot H}{F_{\mathrm{s}}}$$

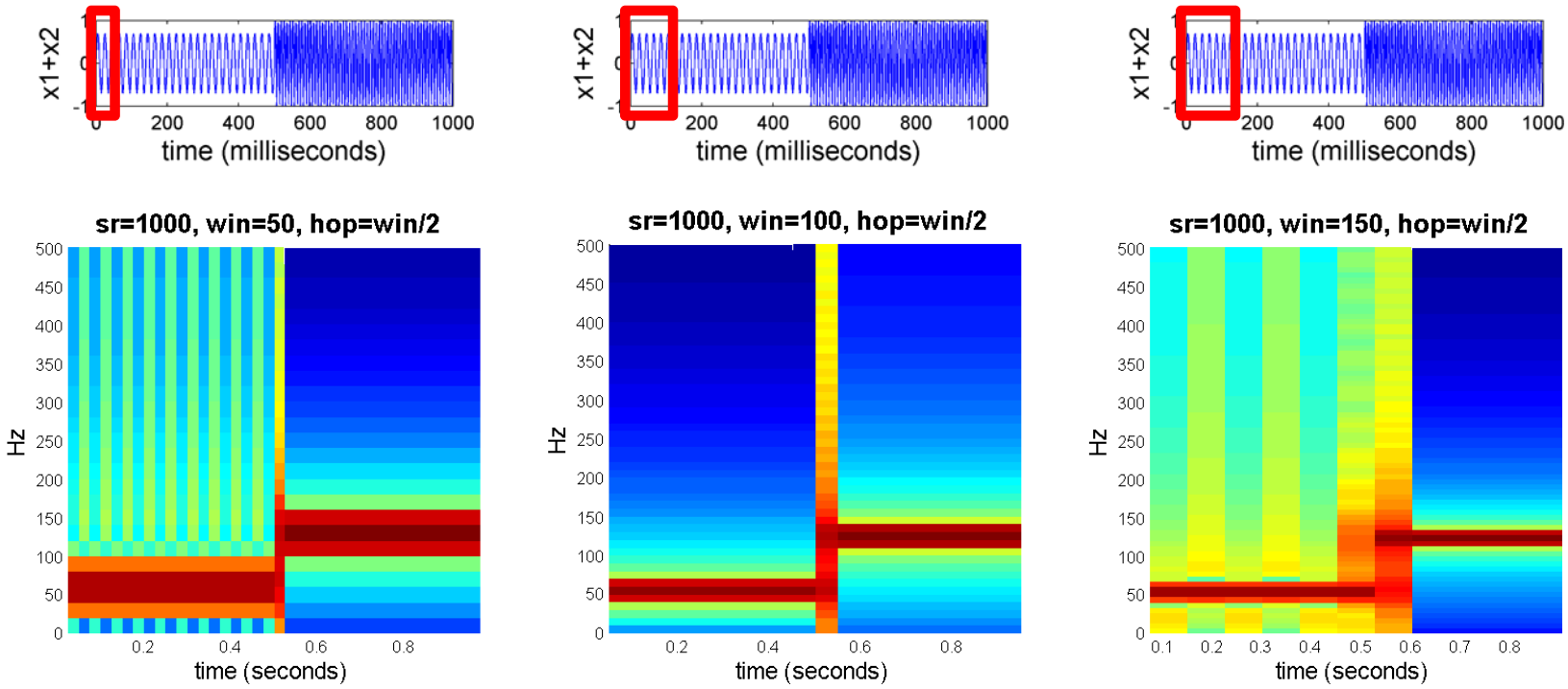
Trade-off between Frequency/Temporal Resolution in STFT

- **Frequency resolution**: $\frac{F_S}{N}$
  - Longer window size (N) → finer frequency resolution (but larger resulting STFT)
    → can localize events along the frequency axis
- **Temporal resolution**: $\frac{H}{F_S}$
  - Smaller hop size (H) → finer temporal resolution (but larger resulting STFT)
    → can localize events along the time axis

- If $H = N/R$ (e.g., $R = 4$)
  - Temporal resolution: $\frac{N}{RF_S}$ (no good freq/time resolution at the same time)

Trade-off between Frequency/Temporal Resolution in STFT

- Different window size (win_size = 50, 100, 150)
- hop_size = win_size/2



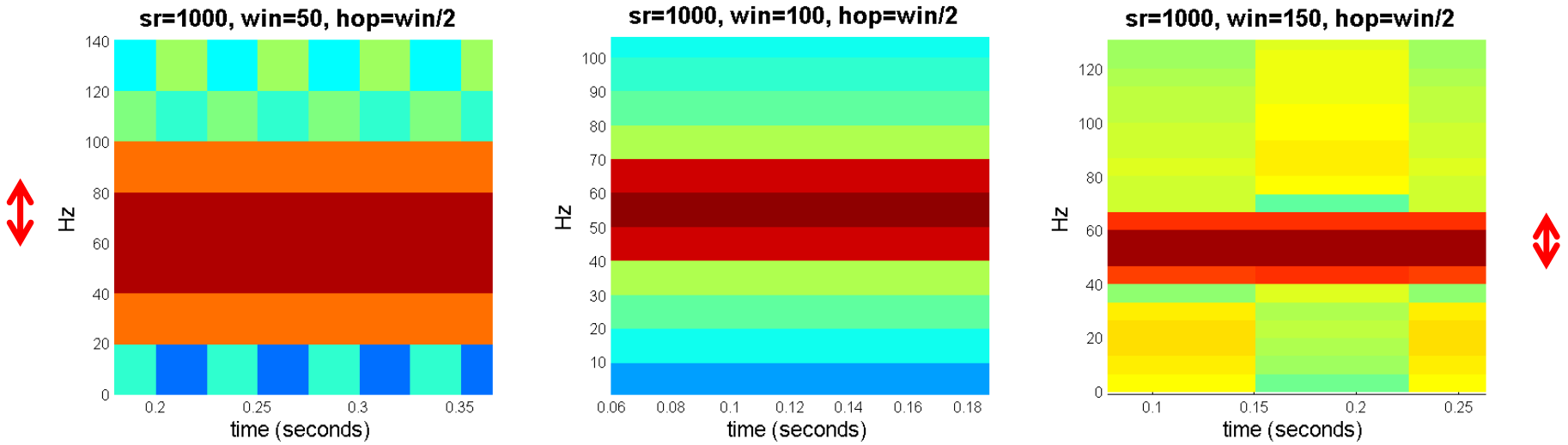
size: 26 x 39

size: 51 x 19

size: 76 x 12

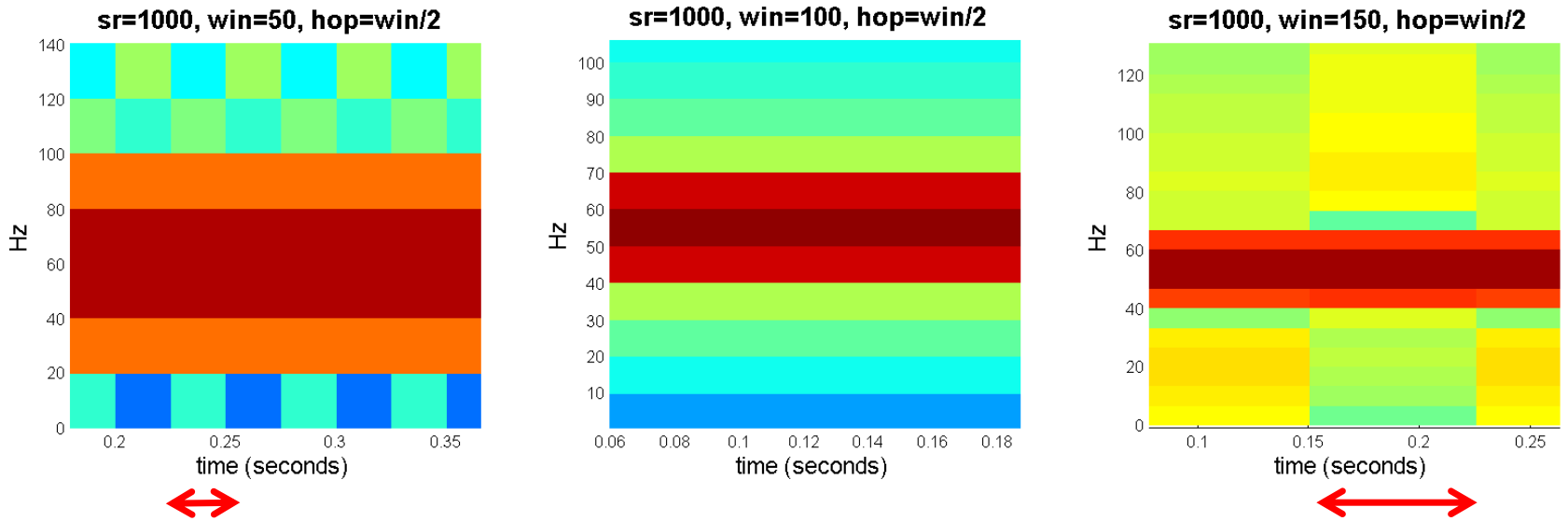
Frequency Resolution in STFT

- **freq_resolution = Fs/win_size**
  - longer window → better frequency resolution
  - freq_resolution = 20, 10, 6.6667 (Hz), respectively



Understanding STFT

- **temporal_resolution: hop_size/Fs**
  - longer hop → worse temporal resolution
  - temp_resolution = 25, 50, 75 (ms), respectively

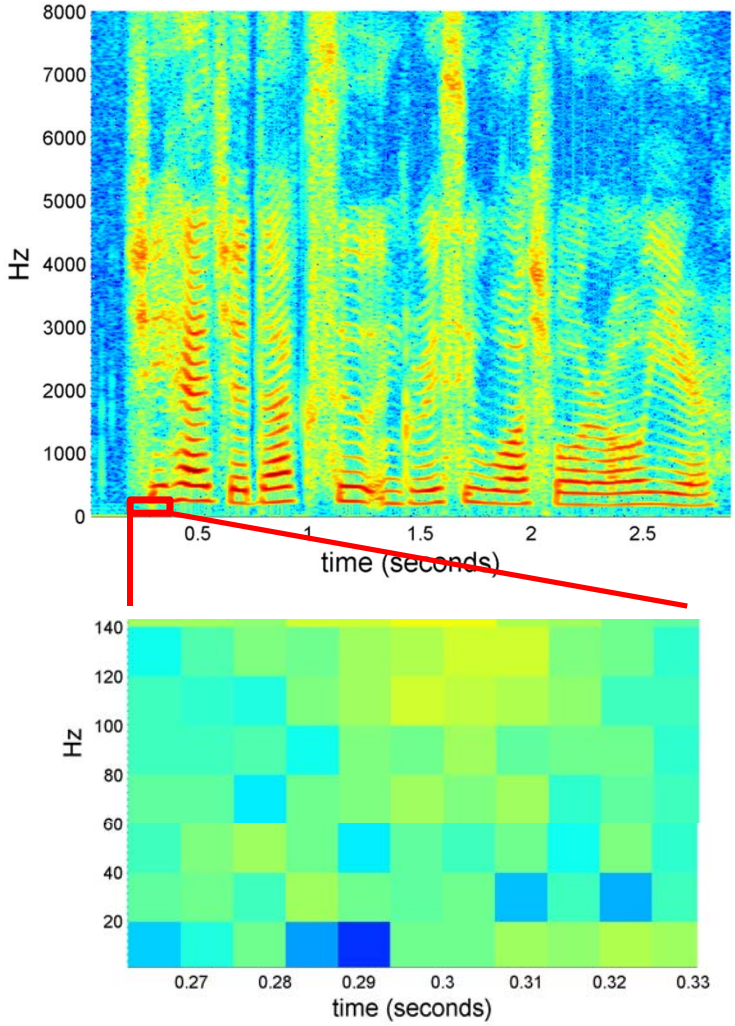


Quiz

1. The figure on the top-right is the spectrogram of a signal. What is the underlying sampling rate?

2. The figure on the bottom-right is a zoom-in of the above figure. We can see that the frequency resolution is 20 Hz. What is the window size (in samples)?

3. The temporal resolution is close to 6.6 ms. What's the hop size (in samples), approximately?



Quiz

4. Given an EEG headset that samples signals at 128 Hz, if we want to be able to discriminate frequency components that differ by 0.5 Hz in frequency, what is the minimal window size (in samples) we need to use? What is the length of such a window in seconds then?
5. Following the previous question, if we further want to discriminate events that differ in time by 0.5 second, what is the maximal hop size (in samples) we need to use?

Brainwaves, Frequencies and Functions

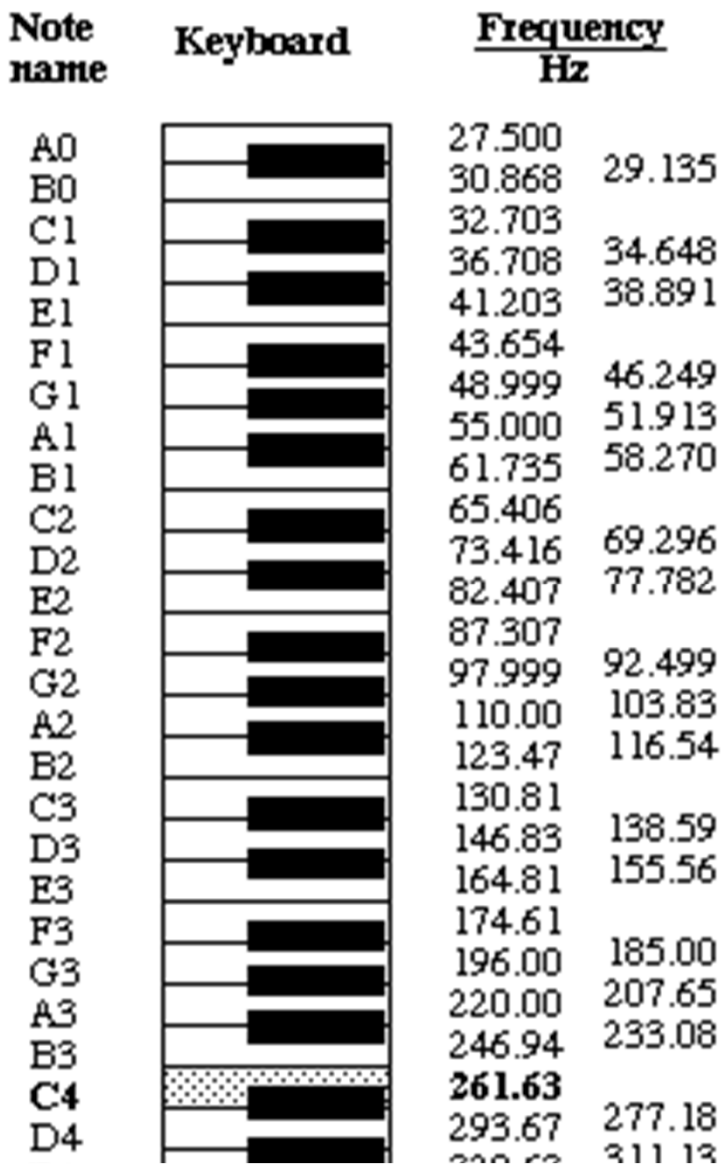
Unconscious		Conscious		
Delta	**Theta**	**Alpha**	**Beta**	**Gamma**
0,5 – 4 Hz	4 – 8 Hz	8 – 13 Hz	13 – 30 Hz	30-42 Hz
Instinct	**Emotion**	**Consciousness**	**Thought**	**Will**

Quiz

6. Given a music signal with sr = 44,100 Hz, when we use a window size of 1,024 samples, what would be the frequency resolution?

7. According to the figure on the right, we know that the fundamental frequency (f0) of A1 is 55 Hz, that of A♯1 is 58.27 Hz, etc. Following the previous question, which notes does the first frequency bin of the STFT would cover?

https://newt.phys.unsw.edu.au/jw/notes.html

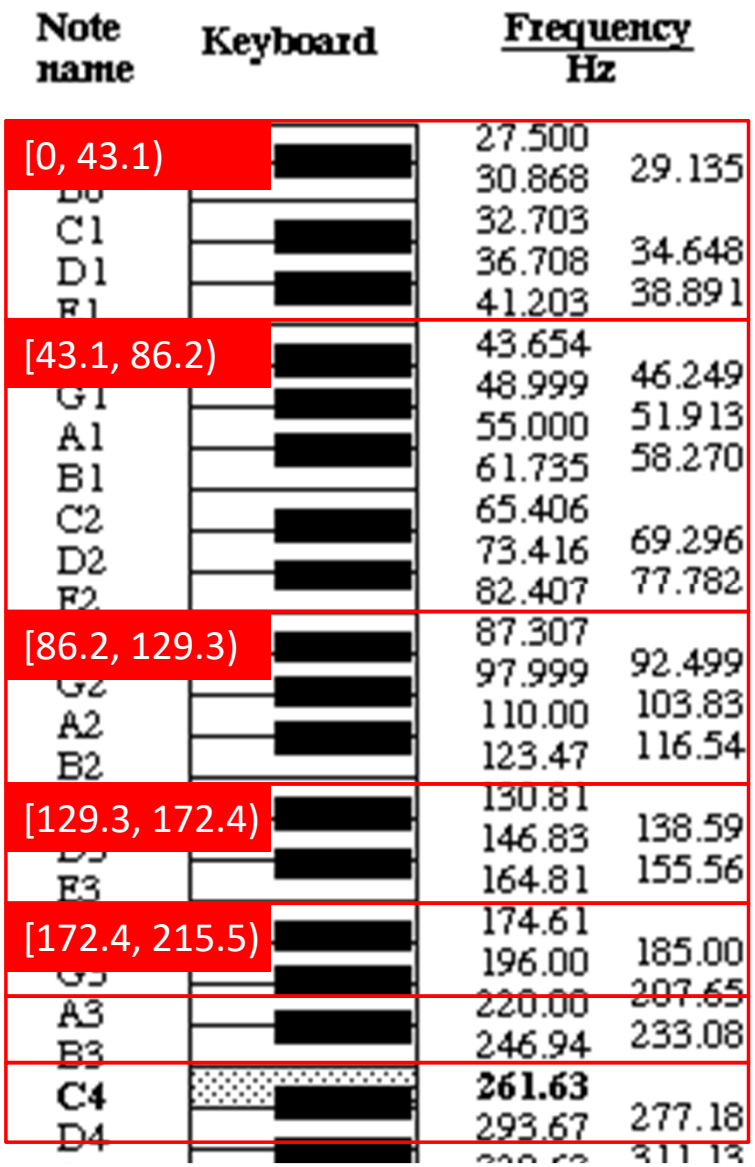


Quiz

6. Given a music signal with sr = 44,100 Hz, when we use a window size of 1,024 samples, what would be the frequency resolution?
   Sol: 43.1 Hz

7. According to the figure on the right, we know that the fundamental frequency (f0) of A1 is 55 Hz, that of A♯1 is 58.27 Hz, etc. Following the previous question, which notes does the first frequency bin of the STFT would cover?

https://newt.phys.unsw.edu.au/jw/notes.html



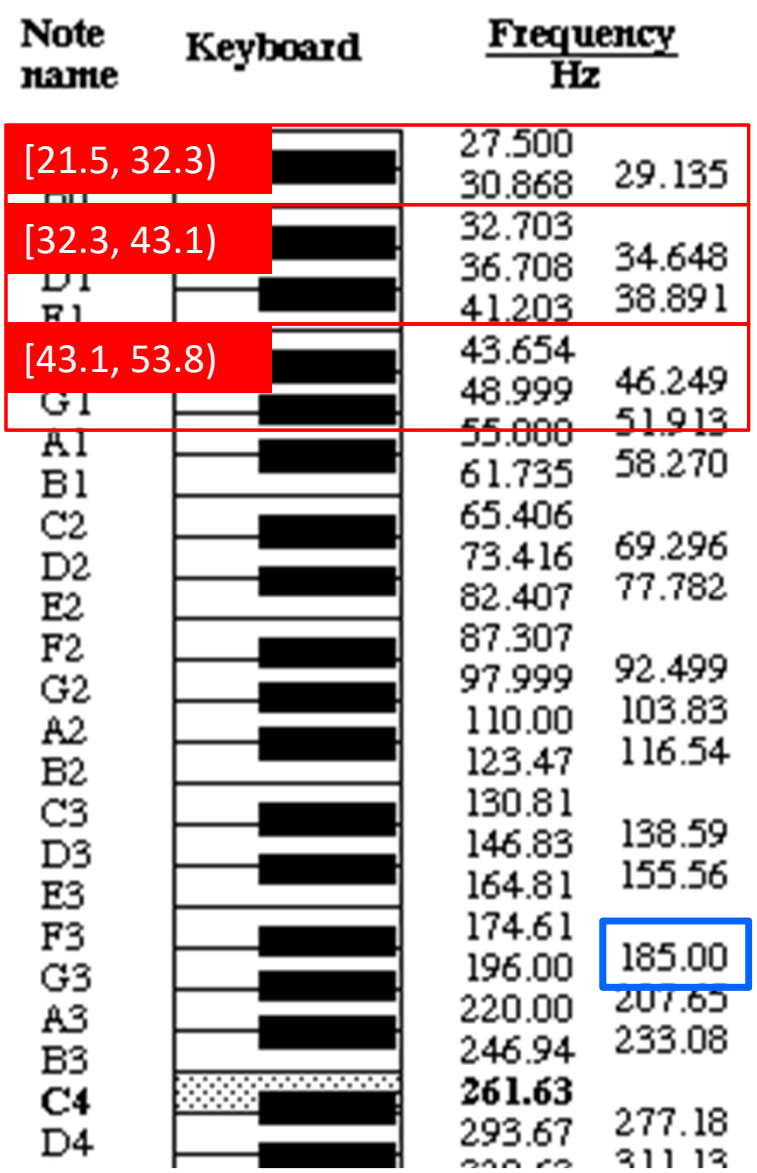
Quiz

8. Following the `6` and `7`;
   how if we use a window size of 4,096 samples?

[0, 10.8)
[10.8, 21.5)
[21.5, 32.3)
[32.3, 43.1)
[43.1, 53.8)
…

(Note: Musical notes after F♯3 would be covered by only one frequency bin now)

https://newt.phys.unsw.edu.au/jw/notes.html



Real Example 1: Piano Transcription

- Curtis Hawthorne et al., "**Onsets and Frames: Dual-objective piano transcription**," ISMIR 2018

  https://archives.ismir.net/ismir2018/paper/000019.pdf

Q: What is the frequency resolution?
And the temporal resolution?

Our onset and frame detectors are built upon the convolution layer acoustic model architecture presented in [13], with some modifications. We use *librosa* [15] to compute the same input data representation of mel-scaled spectrograms with log amplitude of the input raw audio with 229 logarithmically-spaced frequency bins, a hop length of 512, an FFT window of 2048, and a sample rate of 16kHz. We present the network with the entire input sequence, which allows us to feed the output of the convolutional frontend into a recurrent neural network (described below).

Real Example 2: Beat Tracking

- Sebastian Böck, Florian Krebs, and Gerhard Widmer, "**Joint beat and downbeat tracking with recurrent neural networks**," ISMIR 2016

  http://www.cp.jku.at/research/papers/Boeck_etal_ISMIR_2016.pdf

120 BPM = 120 beats per minute
= 2 beats per second

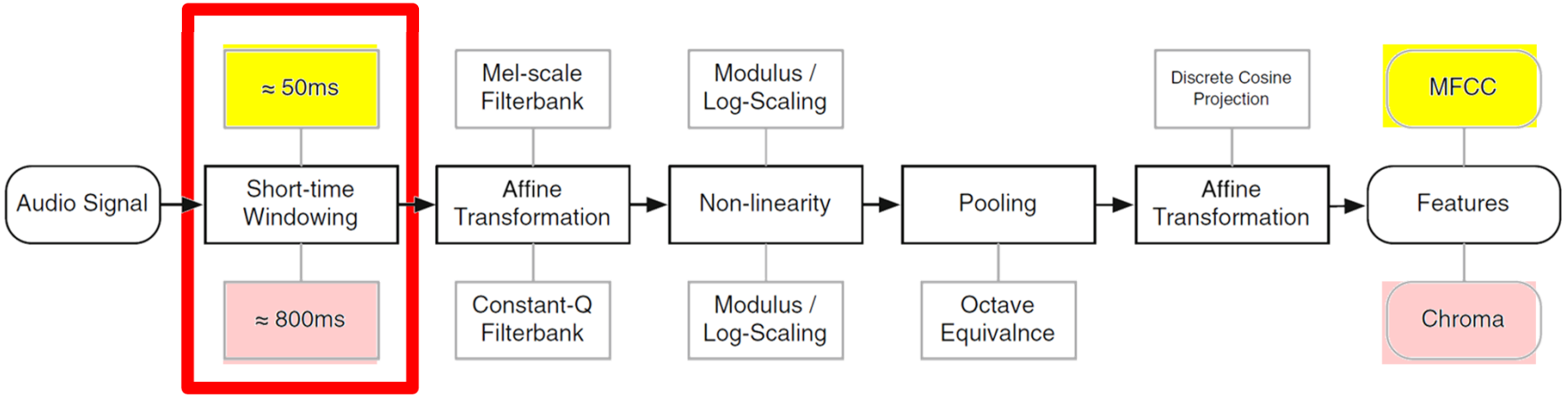
(16th note in 4/4 meter: 125 ms)

2.1 Signal Pre-Processing

The audio signal is split into overlapping frames and weighted with a Hann window of same length before being transferred to a time-frequency representation with the Short-time Fourier Transform (STFT). Two adjacent frames are located 10 ms apart, which corresponds to a rate of 100 fps (frames per second). We omit the phase portion of the complex spectrogram and use only the magnitudes for further processing. To enable the network to capture features which are precise both in time and frequency, we use three different magnitude spectrograms with STFT lengths of 1024, 2048, and 4096 samples (at a signal sample rate of 44.1 kHz). To reduce the dimensionality of the features, we limit the frequencies range to [30, 17000] Hz and process the spectrograms with logarithmically spaced

Different STFT Parameters are Needed for Different Tasks

- **Timbre/rhythm** related tasks tend to use **smaller** windows
- **Pitch/harmony** related tasks tend to use **longer** windows



- Make sure you use the right **Fs, window size** and **hop size!**
  - Especially when using models from open source projects
  - STFTs of the same matrix size may have different physical meanings

Other Time-Frequency Representations

- STFT of multiple window sizes

> we use three different magnitude spectrograms with STFT lengths of 1024, 2048, and 4096 samples (at a signal sample rate of 44.1 kHz). To reduce the dimensionality of the

- CQT
- Wavelets
- Scattering transform

- In DL, STFT is a common choice

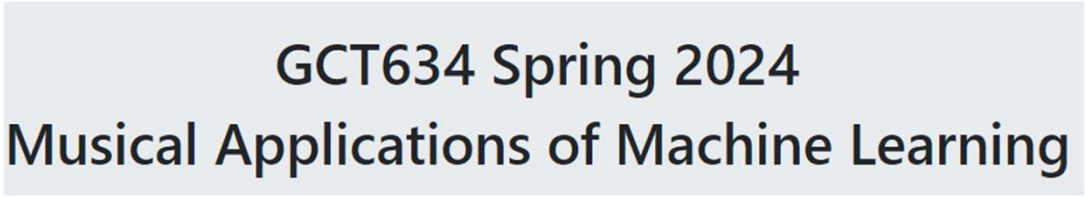
Outline

- Time and frequency representations for music
- Audio features for music
- Math in STFT
- **DL 101**

Reference: KAIST Course & ISMIR 2021 Tutorial

For fundamentals of deep learning and music classification

- https://mac.kaist.ac.kr/~juhan/gct634/Slides/05.%20music%20classification%20-%20deep%20learning.pdf



- https://music-classification.github.io/tutorial/landing-page.html

Feature Learning

(the blocks inside the black lines are learned)

(a) Feature engineering

(b) Low-level feature learning

(c) Convolution neural networks

(d) End-to-end learning

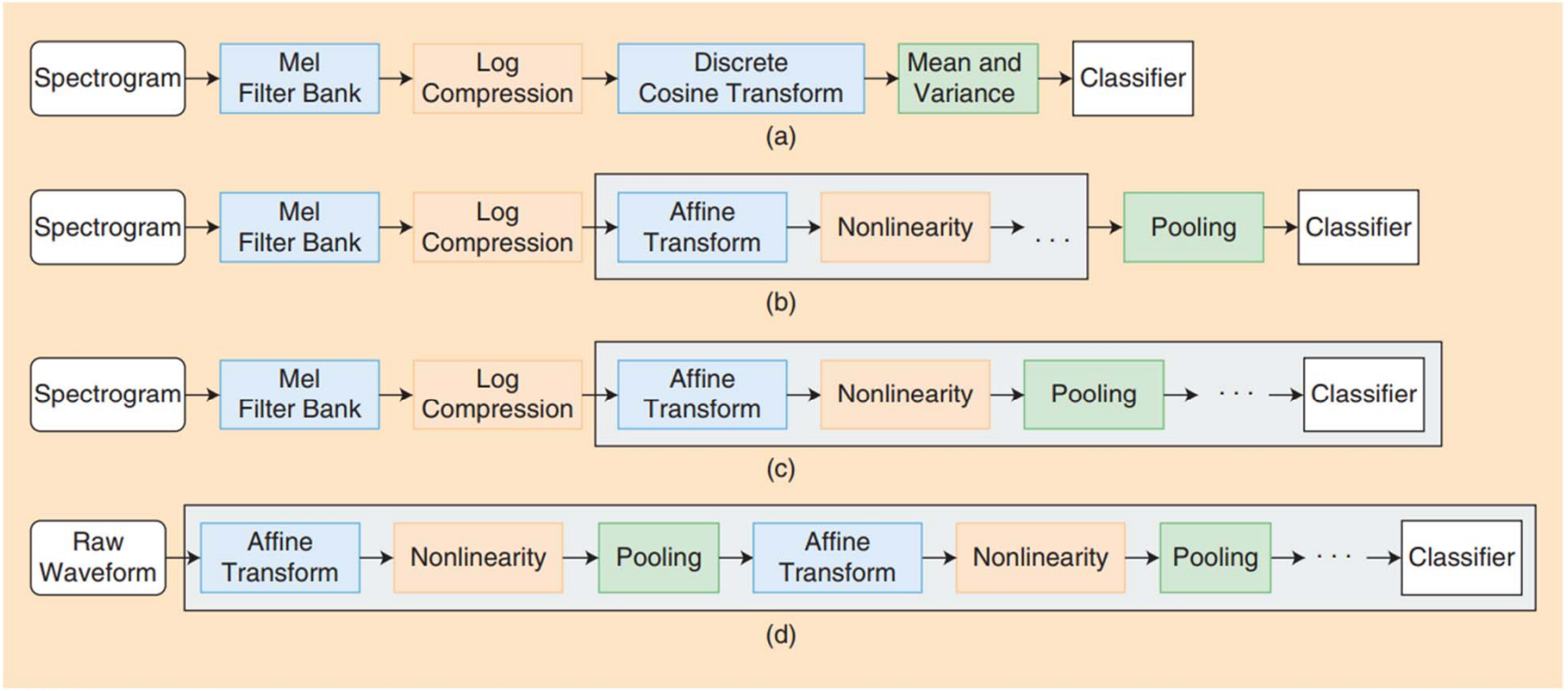


FIGURE 1. The transition of feature representation for music classification: (a) feature engineering [mel-frequency cepstral coefficients (MFCCs)], (b) low-level feature learning, (c) convolutional neural networks, and (d) end-to-end learning. The blocks inside the black lines indicate that they are learned by the algorithms.

Ref: Nam et al., "Deep learning for audio-based music classification and tagging," IEEE Signal Processing Magazine, 2019

Feature Learning by Convolutional Layers

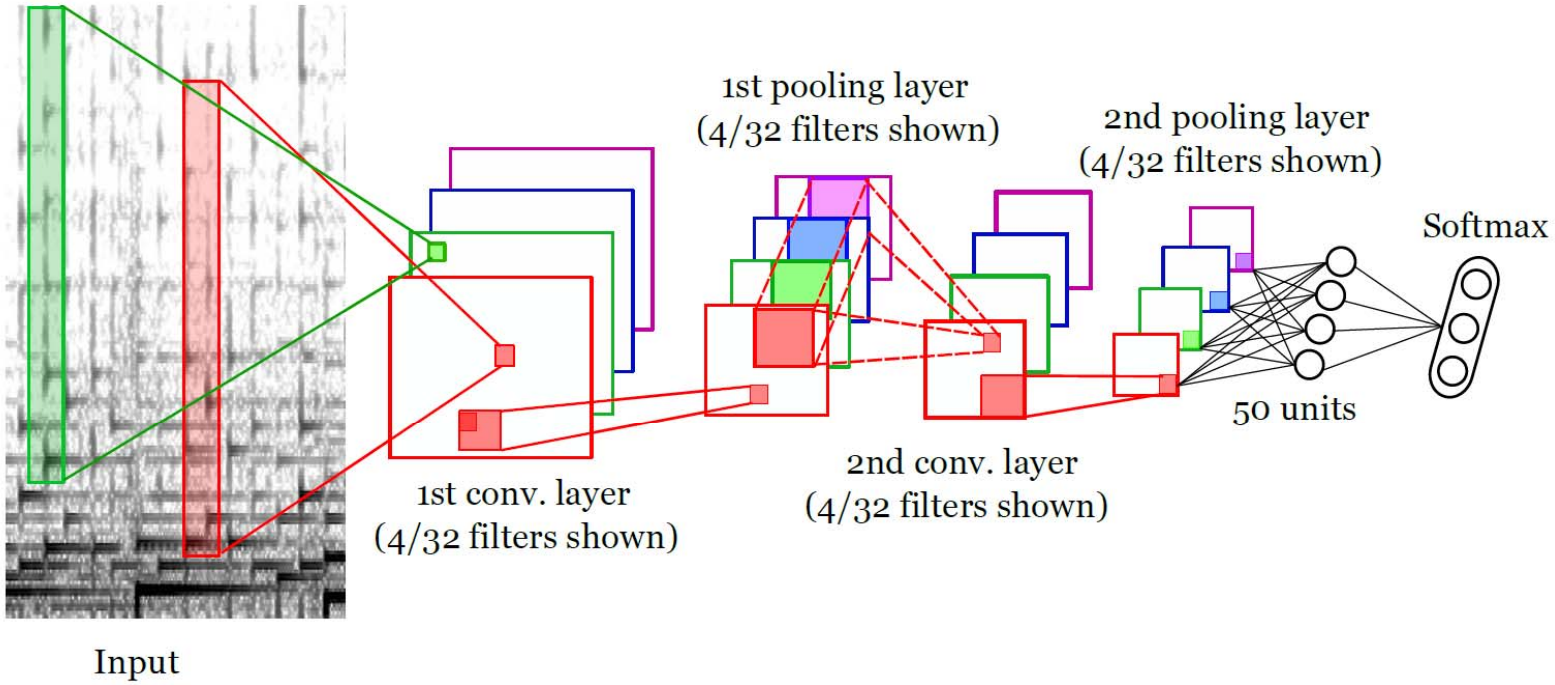


Fig. 1. Illustration of the CDNN architecture we use for our experiments. The CDNN first applies narrow vertical filters to the input sonogram (left) to capture harmonic structure. Then, it applies 32 different filters in the first convolutional layer (we show only 4). This is followed by the first max-pooling layer, and then a 2nd pair of convolutional and max-pooling layers. Finally, the output of the final max-pooling layer is fully connected to a final hidden layer of 50 units, followed by a softmax output unit. The input spectrogram contains 100 time slices, which means that the final layer of the CDNN summarises information over a total duration of 2.35 seconds.

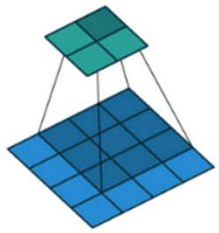
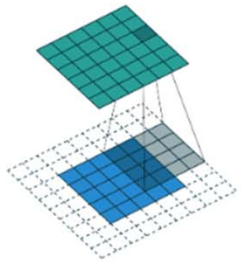
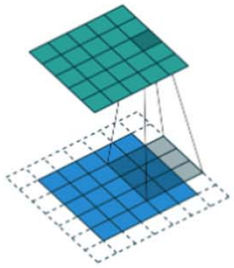
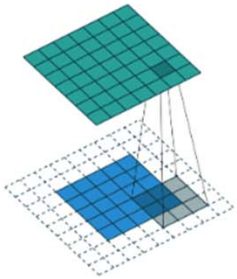
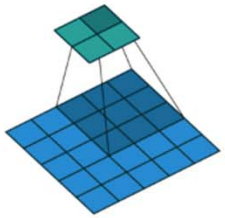
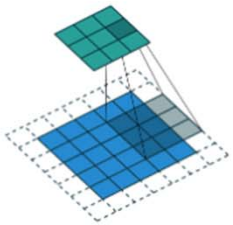
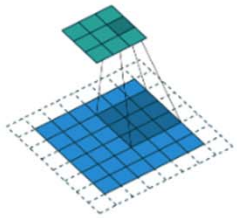
Ref: Kereliuk et al, "Deep learning and music adversaries," IEEE Trans. Multimedia, 2015

Downsampling Layers: Convolution

https://github.com/vdumoulin/conv_arithmetic

Convolution animations

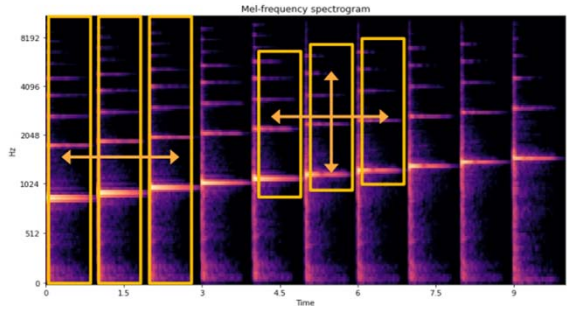
N.B.: Blue maps are inputs, and cyan maps are outputs.

			
No padding, no strides	Arbitrary padding, no strides	Half padding, no strides	Full padding, no strides
			
No padding, strides	Padding, strides	Padding, strides (odd)	

Convolution: Locality and Translation Invariance

- Handle high-dimensional input based on locality and translation invariance of objects
  - Locality: the objects of interest tend to have a local spatial support
    Important parts of the object structures are locally correlated
  - Translation invariance: object appearance is independent of location

- The locality and translation invariance works in the audio domain
  - Both 1D and 2D translation are possible
  - 2D translation is only on the log-frequency scale which makes harmonic patterns approximately shifted-invariant



- "Convolution + pooling" may lead to translation invariance
- See Dr. Juhan Nam's slides (GCT634)

Different Convolution Approaches

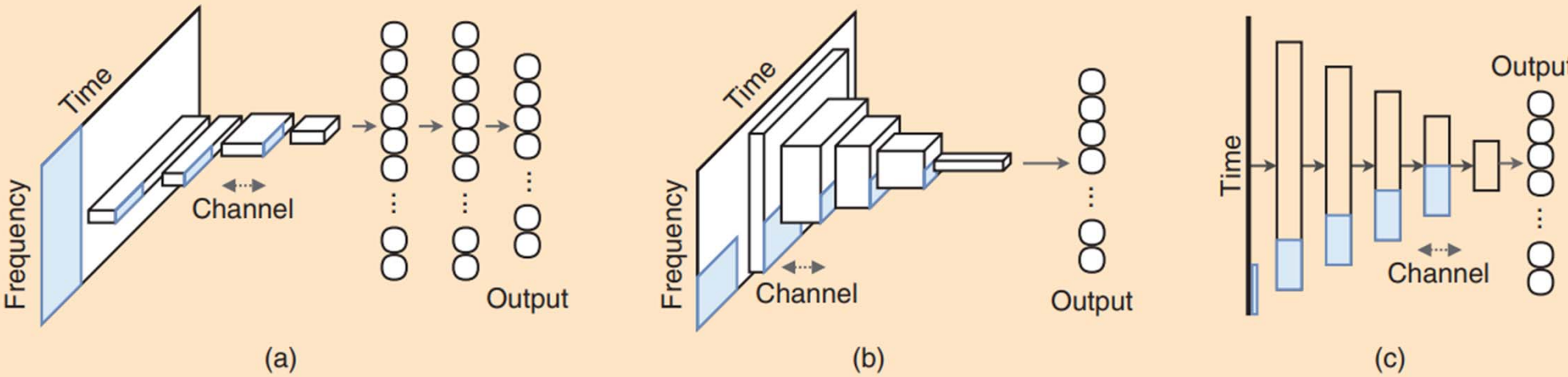
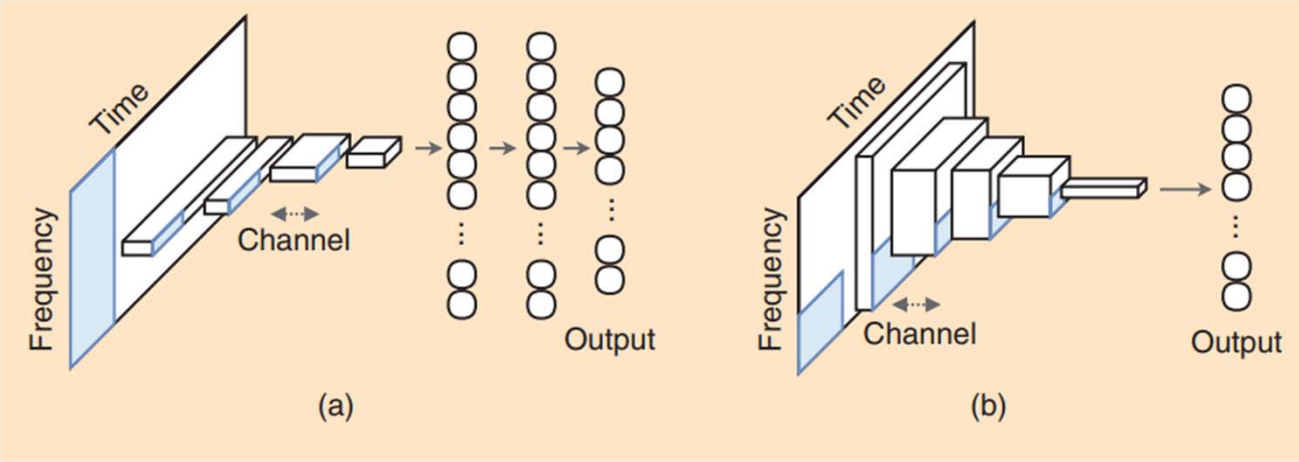


FIGURE 2. Block diagrams of (a) 1-D, (b) 2-D, and (c) sample-level CNNs. (a) and (b) are based on 2-D time–frequency representation inputs (e.g., mel-spectrograms or short-time Fourier transforms), and (c) is based on a time-series input.

- 1D CNNs
- 2D CNNs
- Sample-level CNNs

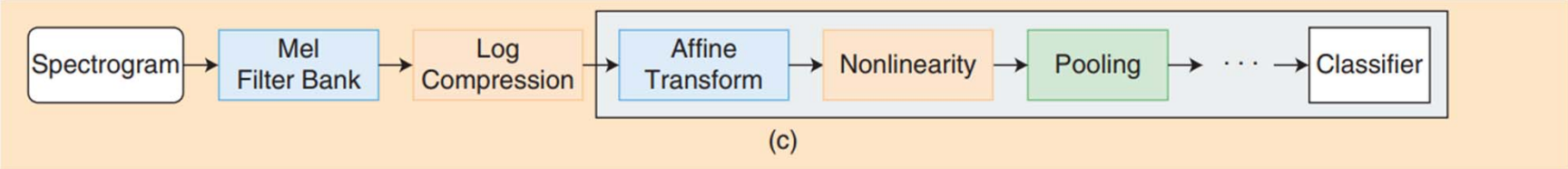
1D CNNs vs. 2D CNNs



- **1D CNNs**
  - The filter size of the first conv layer covers the *entire* frequency range (can cover multiple *frames*)
  - Fast to train
  - **Time**-invariant but not **pitch**-invariant
- **2D CNNs**
  - Significantly increases the number of parameters and thus need more computational resources
  - More flexible and powerful

Input Audio Representation

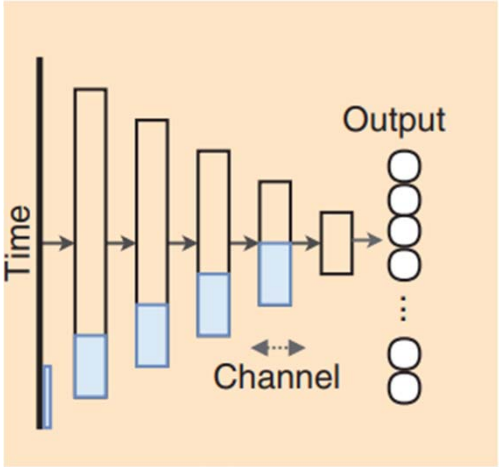
https://music-classification.github.io/tutorial/part2_basics/input-representations.html



- **Log-magnitude STFT**
  - The most "raw" kind of spectrograms
  - Discard phase: the human auditory system is insensitive to phase information
  - Log compresssion: human perception of loudness is closer to a logarithmic scale
- **Melspectrograms**
  - Based on a Mel-scale, which is nonlinear and approximates human perception
  - Reduces the number of frequency band greatly (1,024 → 128)

Sample-level CNN

- Work on audio samples (e.g., two or three samples) rather than a typical window size (e.g., 512 samples)
- Longer training time; need larger compute



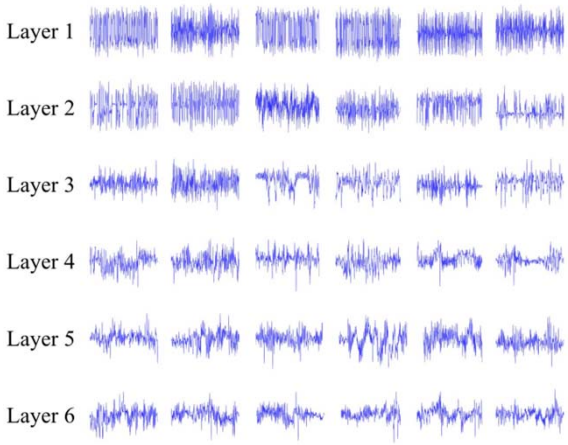
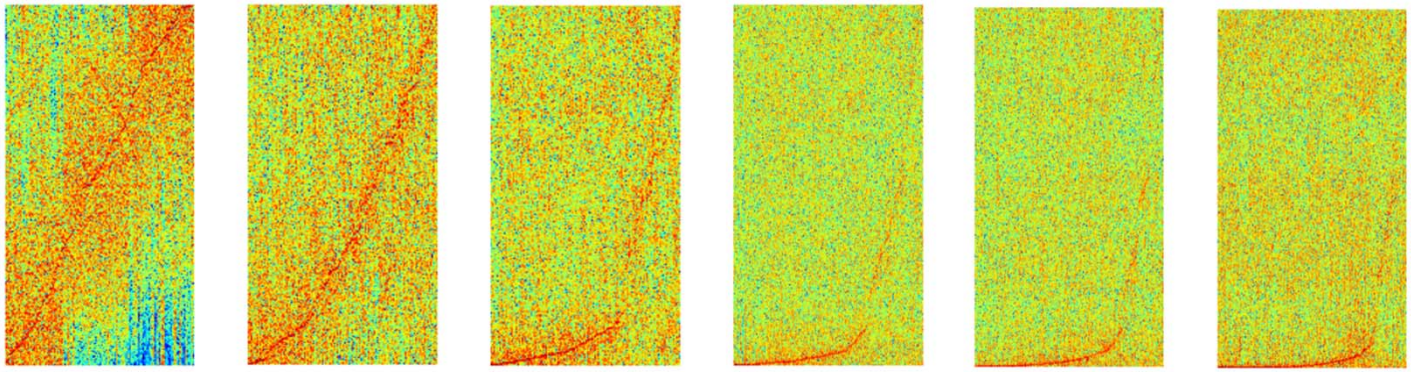


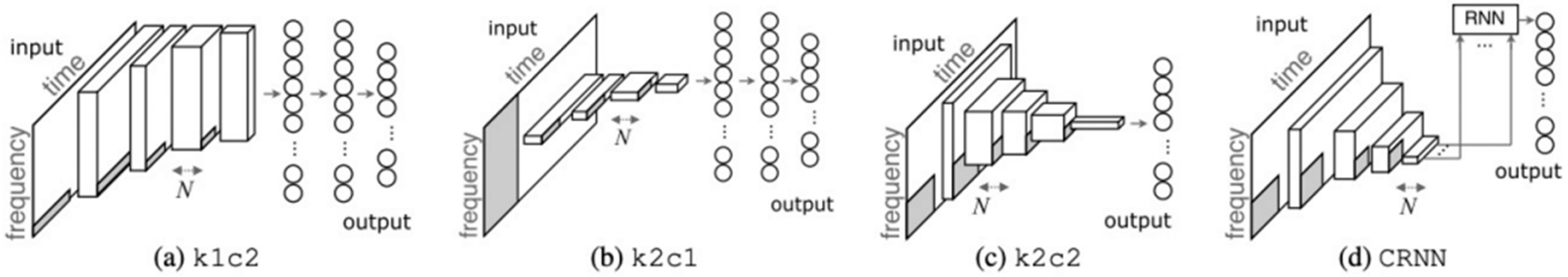
Figure 3. Examples of learned filters at each layer.



Ref: Lee et al., “Sample-level deep convolutional neural networks for music auto-tagging using raw waveforms,” SMC 2017

Convolutional Recurrent Neural Networks

https://music-classification.github.io/tutorial/part3_supervised/architectures.html



- See Dr. Juhan Nam's slides (GCT634)
- Use RNN for temporal summary
- The CRNN model slightly outperforms the CNN models but it is slower

Ref: Choi et al., "Convolutional recurrent neural networks for music classification," ICASSP 2017

Short-Chunk CNN

https://github.com/minzwon/sota-music-tagging-models

```
self.to_db = torchaudio.transforms.AmplitudeToDB()
self.spec_bn = nn.BatchNorm2d(1)

# CNN
self.layer1 = Conv_2d(1, n_channels, pooling=2)
self.layer2 = Conv_2d(n_channels, n_channels, pooling=2)
self.layer3 = Conv_2d(n_channels, n_channels*2, pooling=2)
self.layer4 = Conv_2d(n_channels*2, n_channels*2, pooling=2)
self.layer5 = Conv_2d(n_channels*2, n_channels*2, pooling=2)
self.layer6 = Conv_2d(n_channels*2, n_channels*2, pooling=2)
self.layer7 = Conv_2d(n_channels*2, n_channels*4, pooling=2)

# Dense
self.dense1 = nn.Linear(n_channels*4, n_channels*4)
self.bn = nn.BatchNorm1d(n_channels*4)
self.dense2 = nn.Linear(n_channels*4, n_class)
self.dropout = nn.Dropout(0.5)
self.relu = nn.ReLU()
```

```
def forward(self, x):
    # Spectrogram
    x = self.spec(x)
    x = self.to_db(x)
    x = x.unsqueeze(1)
    x = self.spec_bn(x)

    # CNN
    x = self.layer1(x)
    x = self.layer2(x)
    x = self.layer3(x)
    x = self.layer4(x)
    x = self.layer5(x)
    x = self.layer6(x)
    x = self.layer7(x)
    x = x.squeeze(2)

    # Dense
    x = self.dense1(x)
    x = self.bn(x)
    x = self.relu(x)
    x = self.dropout(x)
    x = self.dense2(x)
    x = nn.Sigmoid()(x)
```

Ref: Won et al., “Evaluation of CNN-based automatic music tagging models,” SMC 2020

Short-Chunk CNN

https://github.com/minzwon/sota-music-tagging-models

Available Models

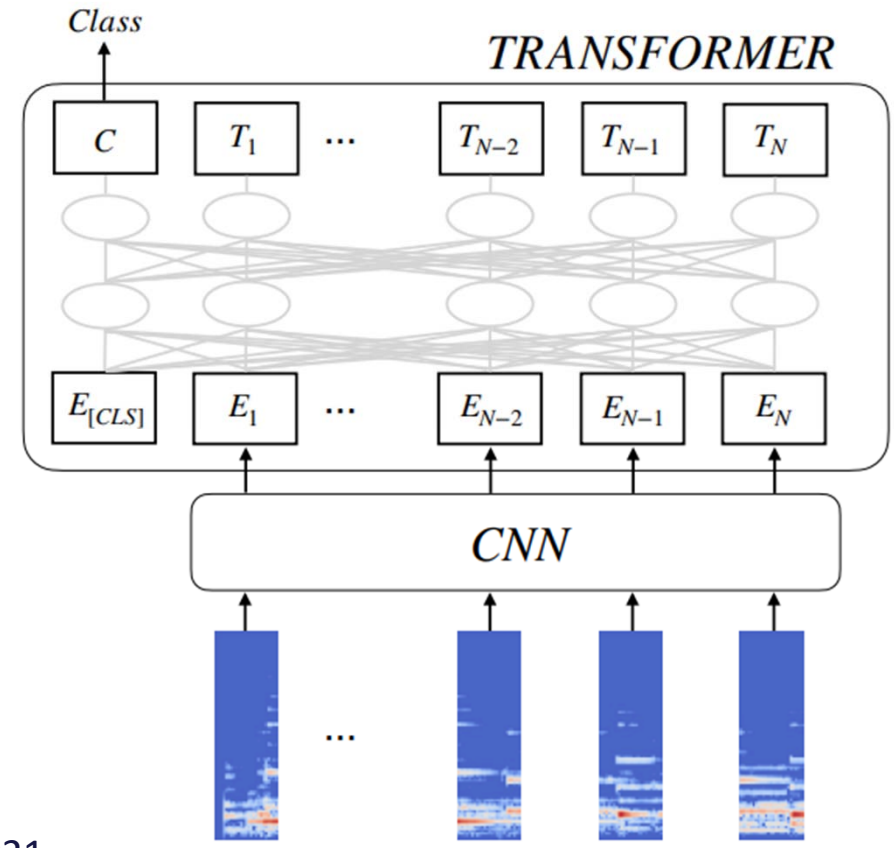
- **FCN** : Automatic Tagging using Deep Convolutional Neural Networks, Choi et al., 2016 [arxiv]
- **Musicnn** : End-to-end Learning for Music Audio Tagging at Scale, Pons et al., 2018 [arxiv]
- **Sample-level CNN** : Sample-level Deep Convolutional Neural Networks for Music Auto-tagging Using Raw Waveforms, Lee et al., 2017 [arxiv]
- **Sample-level CNN + Squeeze-and-excitation** : Sample-level CNN Architectures for Music Auto-tagging Using Raw Waveforms, Kim et al., 2018 [arxiv]
- **CRNN** : Convolutional Recurrent Neural Networks for Music Classification, Choi et al., 2016 [arxiv]
- **Self-attention** : Toward Interpretable Music Tagging with Self-Attention, Won et al., 2019 [arxiv]
- **Harmonic CNN** : Data-Driven Harmonic Filters for Audio Representation Learning, Won et al., 2020 [pdf]
- **Short-chunk CNN** : Prevalent 3x3 CNN. So-called *vgg*-ish model with a small receptieve field.
- **Short-chunk CNN + Residual** : Short-chunk CNN with residual connections.

CNN+Transformer for Music Classification

https://github.com/minzwon/semi-supervised-music-tagging-transformer

- Use Transformer for temporal summary

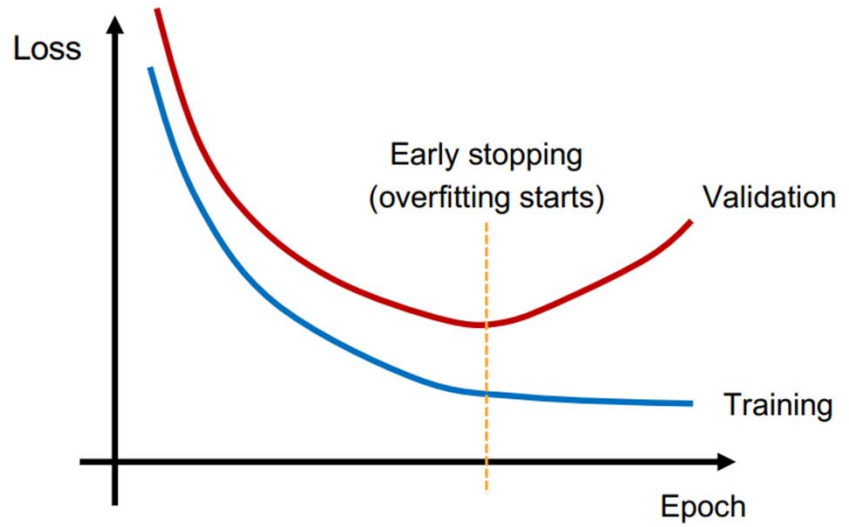
Models	ROC-AUC	PR-AUC
FCN [2]	0.8742	0.2963
Musicnn [3]	0.8788	0.3036
Sample-level [4]	0.8789	0.2959
Sample-level+SE [38]	0.8838	0.3109
CRNN [19]	0.8460	0.2330
CNNSA [20]	0.8810	0.3103
Harmonic CNN [5]	0.8898	0.3298
Short-chunk CNN [6]	0.8883	0.3251
Short-chunk ResNet [6]	0.8898	0.3280
Transformer (proposed)§	**0.8916**	**0.3358**
Transformer (proposed) + DA†	**0.8972**	**0.3479**



Ref: Won et al., “Semi-supervised music tagging Transformer,” ISMIR 2021

Tricks when Training DL Models

- Avoid overfitting
  - Dropout
  - Weight decay (L1, L2 norm of weights)
  - Early stopping
  - Reduce model size
  - Data augmentation



- Overfitting is better than underfitting
  - Scale up the model till it overfits

- Try different learning rates and optimizers