2025 edition

Deep Learning for Music Analysis and Generation

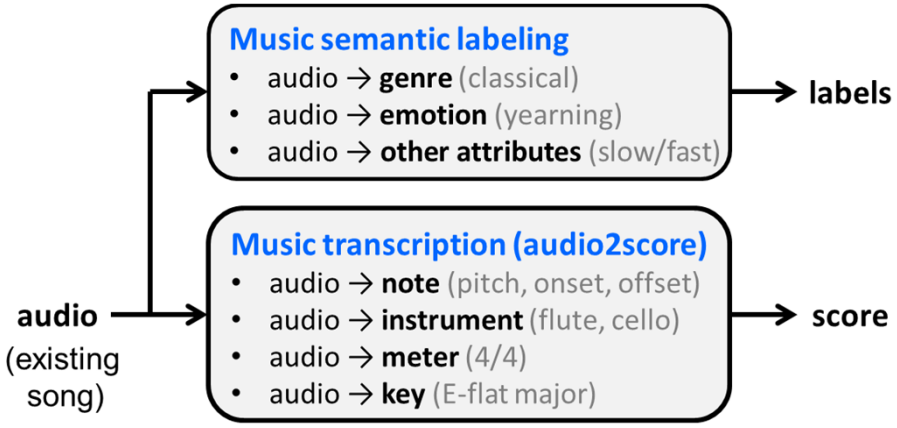
Music Classification

(audio → labels)

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

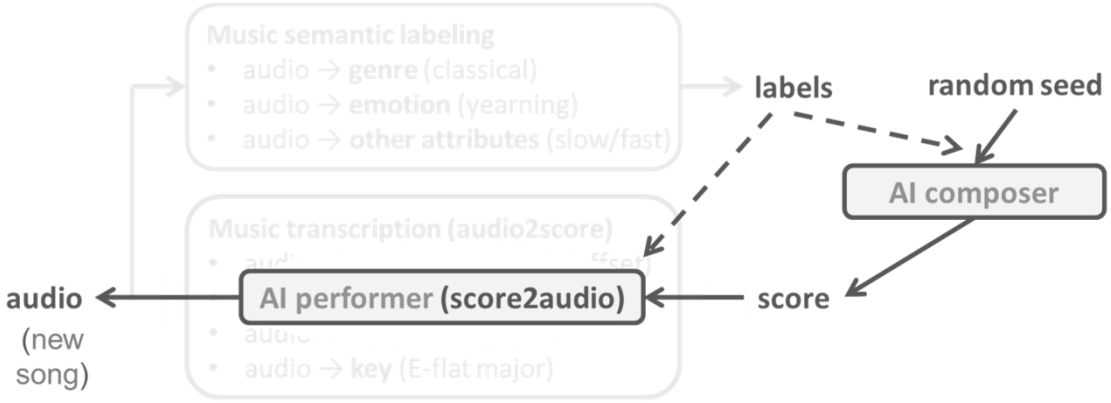
Music AI; or *Music Information Research* (MIR)

- **Music analysis**



  - music understanding
  - music search
  - music recommendation

- **Music generation**

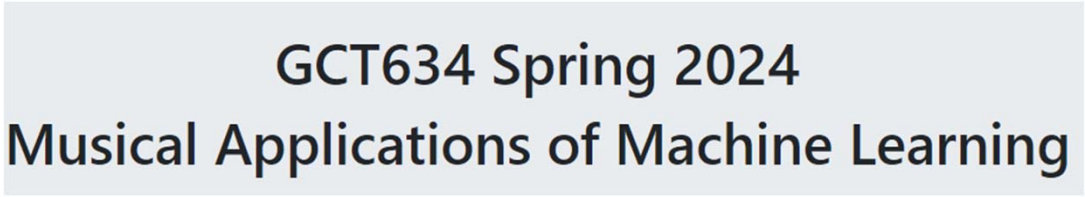


  - MIDI generation
  - audio generation
  - MIDI-to-audio generation

Reference 1: 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

Reference 2

- *Deep Learning An MIT Press book* (2016)
  by Ian Goodfellow and Yoshua Bengio and Aaron Courville
  https://www.deeplearningbook.org/

- *Deep Learning - Foundations and Concepts* (2024)
  by Christopher M. Bishop & Hugh Bishop
  https://link.springer.com/book/10.1007/978-3-031-45468-4

- *Deep Learning 101 for Audio-based MIR* (2024)
  by Geoffroy Peeters, Gabriel Meseguer-Brocal, Alain Riou & Stefan Lattner
  https://geoffroypeeters.github.io/deeplearning-101-audiomir_book/task_musicprocessing.html

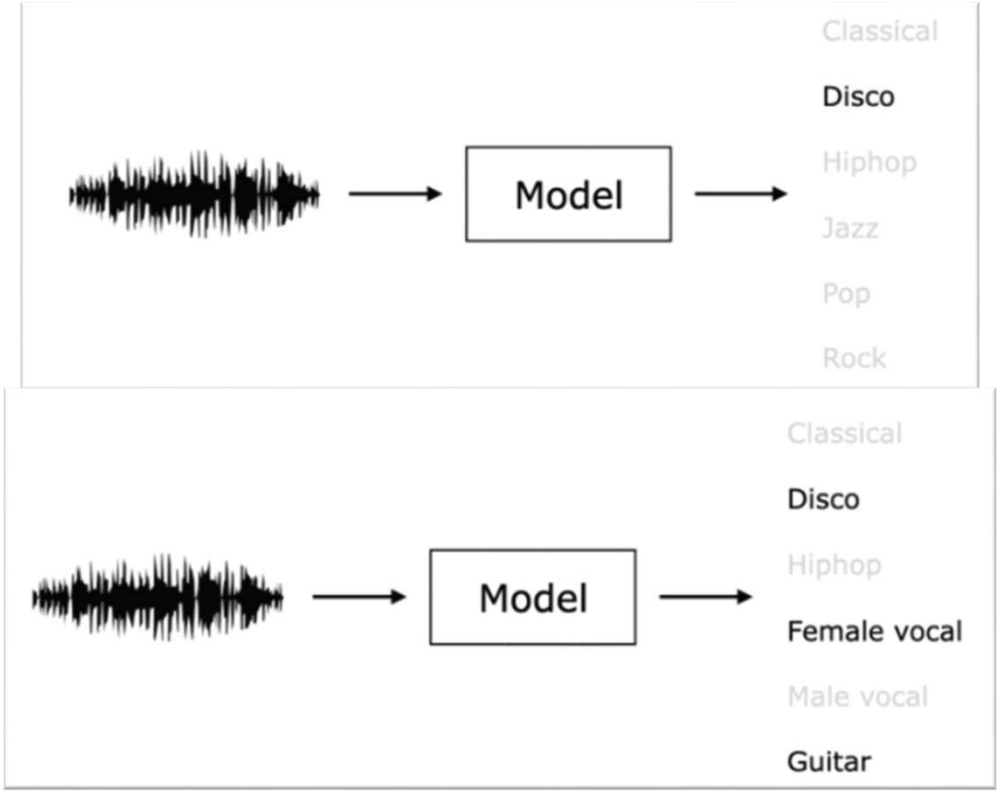
Outline

- **Music classification: Basics**
- ML-based music classification (and hand-crafted audio features)
- DL-based music classification
- Music foundation models

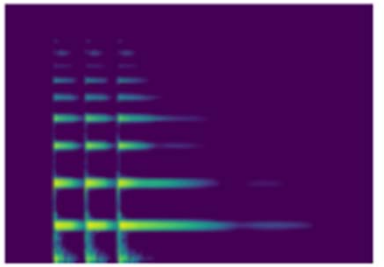
Different Classification Tasks

https://music-classification.github.io/tutorial/part1_intro/what-is-music-classification.html

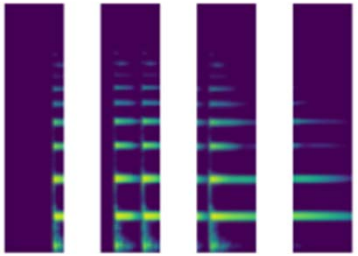
- Single-label vs multi-label



- Song-level vs instance-level
  - instance/chunk/clip/segment (various names)



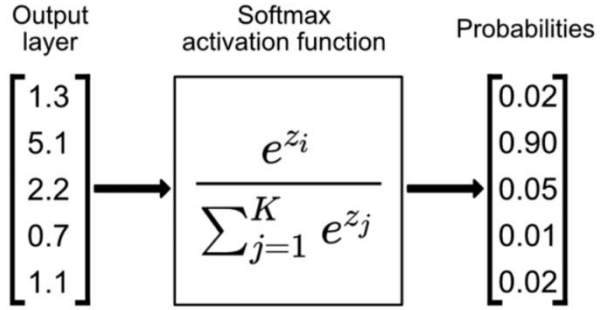
Song-level



Instance-level

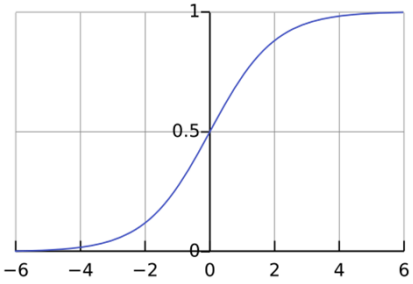
Single-label vs Multi-label Classification

- **Single-label** classification
  - One-hot; one out of many (mutually exclusive classes)
  - Can be *binary* or *multi-class* classification
  - Activation function (in DL): **softmax** (sum-to-one)
  - Output interpretation: argmax
  - Loss function (in DL): **categorical cross entropy** (CE)



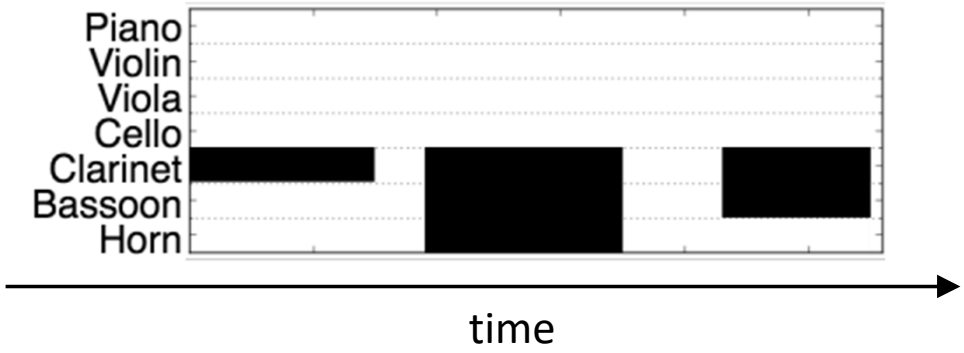
https://www.singlestore.com/blog/a-guide-to-softmax-activation-function/

- **Multi-label** classification
  - Multi-hot; some out of many (assigning an input to multiple classes simultaneously)
  - Activation function (in DL): **sigmoid** (each in [0,1], not sum-to-one)
  - Output interpretation: ≥ 0.5 (or other thresholds)
  - Loss function in DL: **binary cross entropy** (BCE)



Song-label vs Instance-label Classification

- **Song-level**: Make a prediction for the entire song (or a long audio clips), without specifying the temporal location/activation of each class
- **Instance-level**: Need to mark the temporal location/activation of each class



- It's related to the **length of the model input**
  - Make a prediction per STFT frame
  - Make a prediction every second
  - Make a prediction for a 30-second spectrogram
  - Make a prediction per musical note (variable-length)
  - Make a prediction per musical section (verse, chorus, etc) (variable-length)

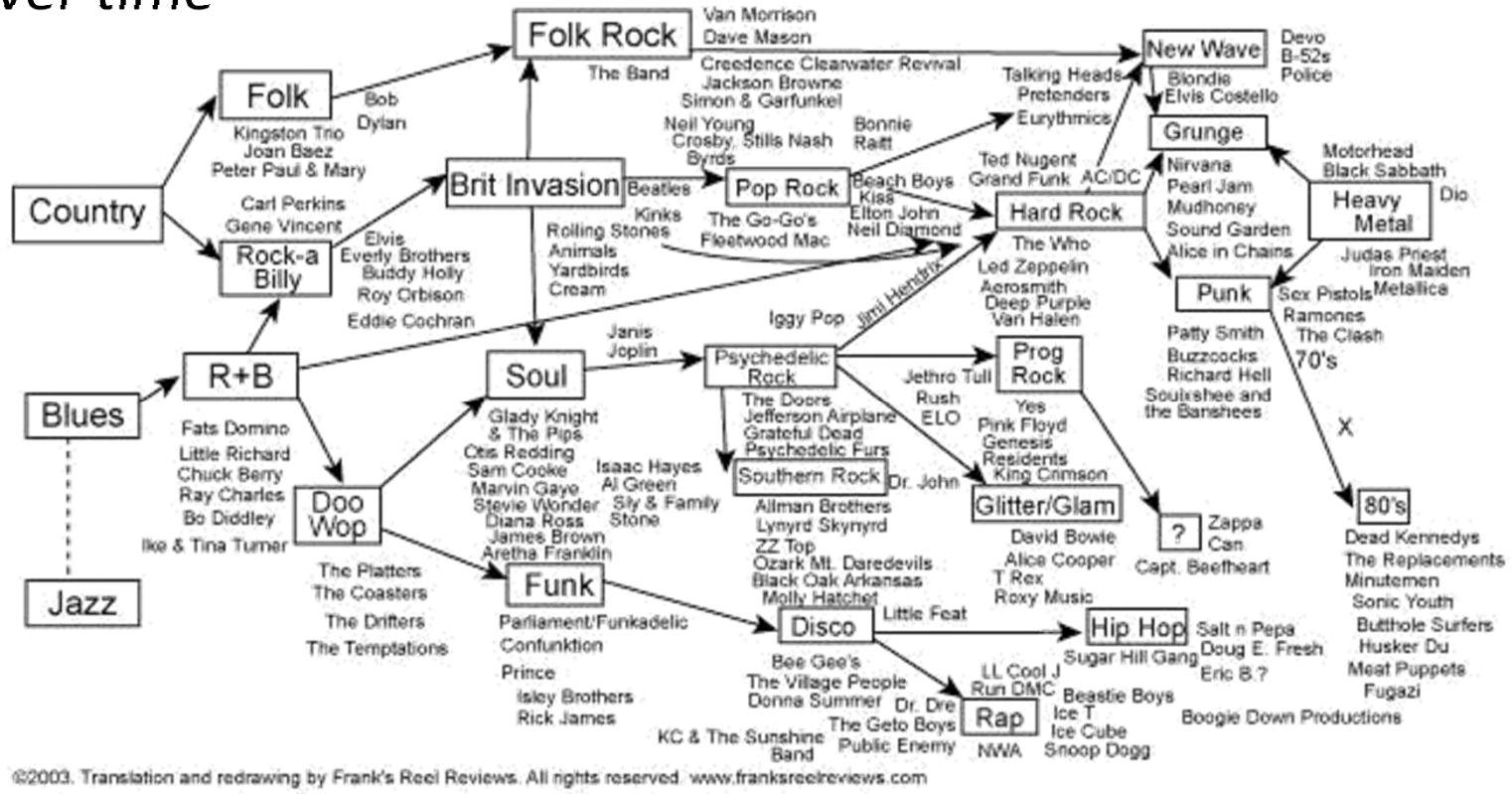
Different Classification Tasks

https://music-classification.github.io/tutorial/part1_intro/what-is-music-classification.html

- The most explored music classification tasks in MIR
  - Genre classification [TC02]
  - Mood classification [KSM+10]
  - Instrument identification [HBPD03]
  - Music tagging [Lam08]
- Many others
  - singer/composer classification
  - technique classification
  - audio event detection

Genre/Style Classification

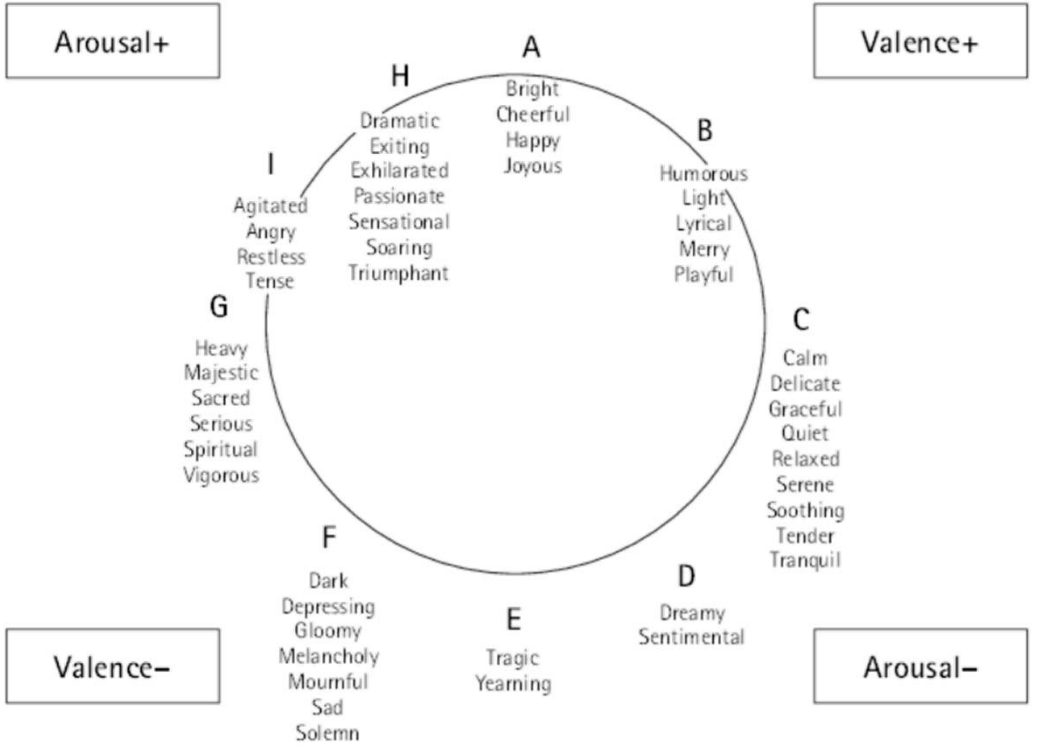
- A conventional category that identifies some pieces of music as belonging to a shared tradition or set of conventions
- Evolve over time



Emotion/Mood Classification/Regression

https://github.com/juansgomez87/datasets_emotion

- Perceived vs. felt emotion
- Song-level or instance-level
  - music emotion variation detection
- Classification vs. regression
  - **arousal**: energy or neuro-physiological stimulation level
  - **valence**: pleasantness or positive/negative affective states
  - popular taxonomy: 4Qs of the valence/arousal plane
- Inherently subjective

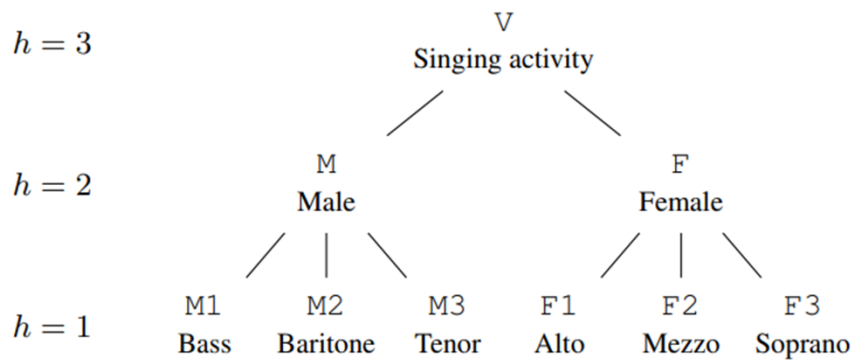


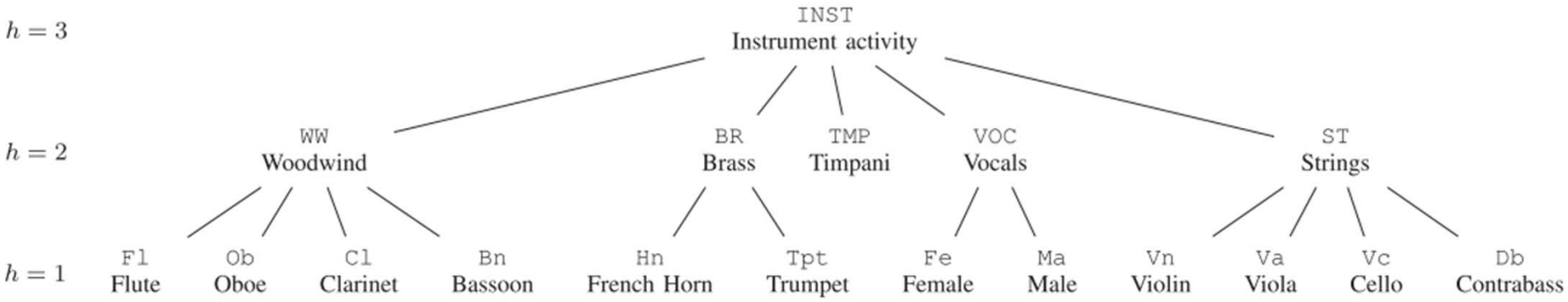
Geneva Emotional Music Scale (GEMS)

https://musemap.org/resources/gems

Instrument Classification/Detection

- Song-level or instance-level
  - singing activity detection
  - instrument activity detection
- Hierarchical taxonomy





Ref1: Krause et al., “Hierarchical classification of singing activity, gender, and type in complex music recordings,” ICASSP 2022
Ref2: Krause et al., “Hierarchical classification for instrument activity detection in orchestral music recordings,” TASLP 2023

Music Tagging

MagnaTagATune (https://mirg.city.ac.uk/codeapps/the-magnatagatune-dataset)

- Top 50 by categories (source)
  - **genre**: classical, techno, electronic, rock, indian, opera, pop, classic, new age, dance, country, metal
  - **instrument**: guitar, strings, drums, piano, violin, vocal, synth, female, male, singing, vocals, no vocals, harpsichord, flute, no vocal, sitar, man, choir, voice, male voice, female vocal, harp, cello, female voice, choral
  - **mood**: slow, fast, ambient, loud, quiet, soft, weird
  - **etc**: beat, solo, beats

Ref: Law et al., "Evaluation of algorithms using games: the case of music annotation," ISMIR 2009

Technique Classification

- Electric guitar
  - bend, vibrato, hammer-on, pull-off, slide
  - https://zenodo.org/record/1414806

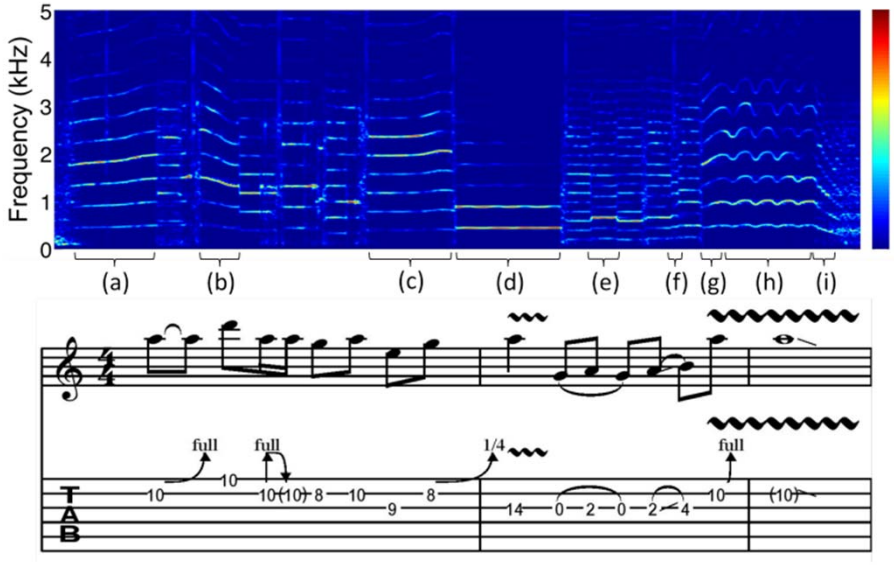
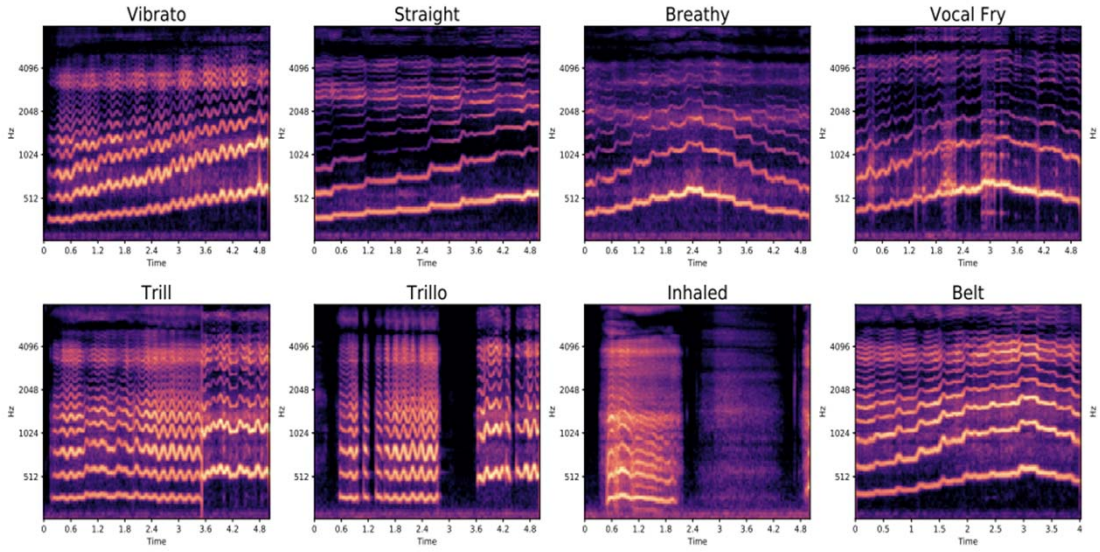


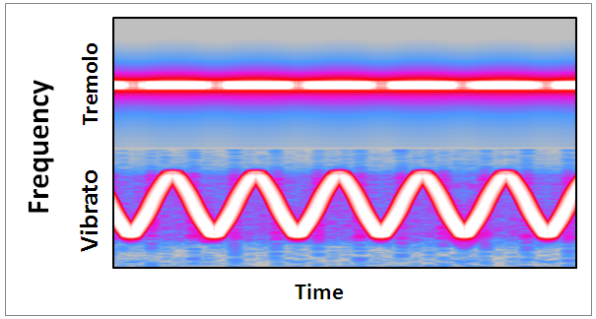
Figure 1. The spectrogram and tablature of a guitar phrase that contains the following techniques: bend (a, b, c, g), vibrato (d, h), hammer-on & pull-off (e) and slide (f, i).

- Singing voice
  - breathy, vibrato, vocal fry, etc
  - https://zenodo.org/record/1193957

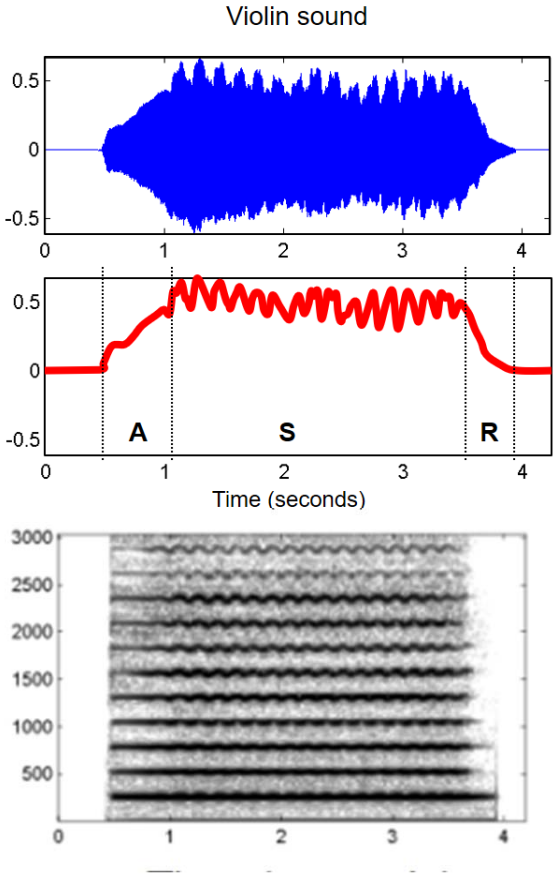


Technique Classification: Vibrato and Tremolo

- **Tremolo**: periodic variations in amplitude (amplitude modulations)
- **Vibrato**: periodic variations in frequency (frequency modulations)
  - Wind and bowed instruments generally use vibratos with an extent of less than half a *semitone* either side
  - Tremolo and vibrato do not necessarily evoke a perceived change in loudness or pitch of the tone



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

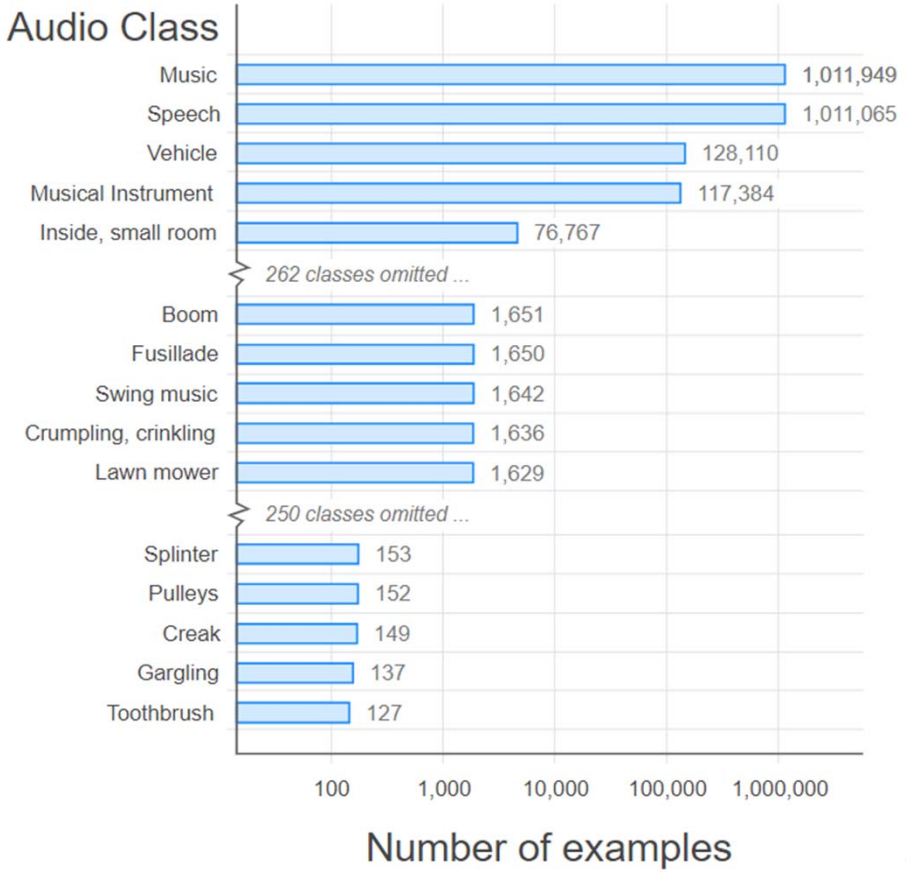


Ref: Sundberg, "Acoustic and psychoacoustic aspects of vocal vibrato," 1994

Audio Event Detection

Audio Set (http://research.google.com/audioset/)

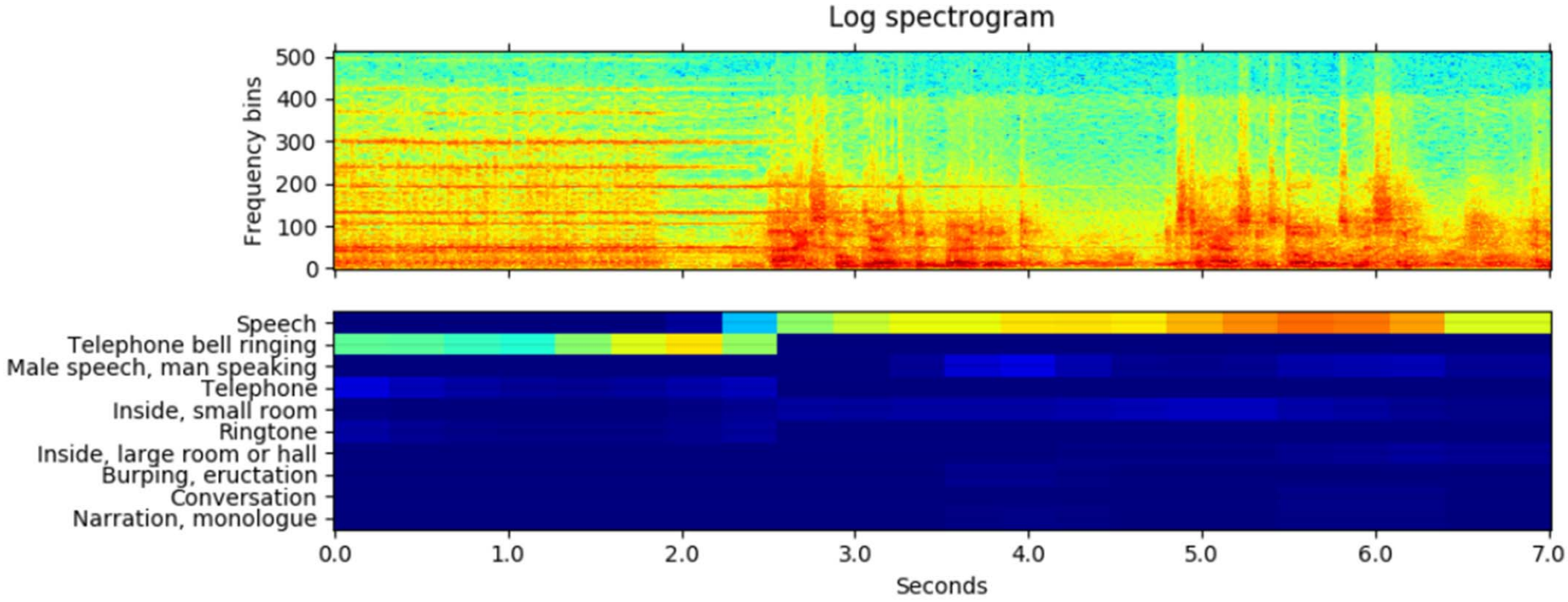
- **527** audio classes
  - Over 2M audio clips from YouTube
  - Each 10 second
- Widely used benchmark for **audio classification** and **audio captioning**
- Can be useful for sound design



Ref: Gemmeke et al., "Audio Set: An ontology and human-labeled dataset for audio events," ICASSP 2017

Audio Event Detection

https://github.com/qiuqiangkong/audioset_tagging_cnn



- Useful for
  - Music/singing/speech detection; instrument activity detection

Different Classification Tasks

Task	Single-label?	Multi-label?	Song-level?	Instance-level?
Genre analysis				
Emotion analysis				
Instrument analysis				
Technical analysis				
Audio event detection				

- It depends on your dataset and your problem formulation
- Single-label, song-level classification is the simplest setting (good for beginners)
- We can do instance level first, then aggregate the result into the song level

Outline

- Music classification: Basics
- **ML-based music classification (and hand-crafted audio features)**
- DL-based music classification
- Music foundation models

ISMIR 2024 Test-of-Time (ToT) Awardee

Musical Genre Classification of Audio Signals

https://www.cs.cmu.edu/~gtzan/work/pubs/tsap02gtzan.pdf

- "A musical genre is characterized by the common characteristics shared by its members. These characteristics typically are related to the instrumentation, rhythmic structure, and harmonic content of the music."
- "In this paper, […] **three feature sets** for representing timbral texture, rhythmic content and pitch content are proposed."
- "The performance and relative importance of the proposed features is investigated by training **statistical pattern recognition classifiers** using real-world audio collections. […] Using the proposed feature sets, classification of 61% for ten musical genres is achieved."

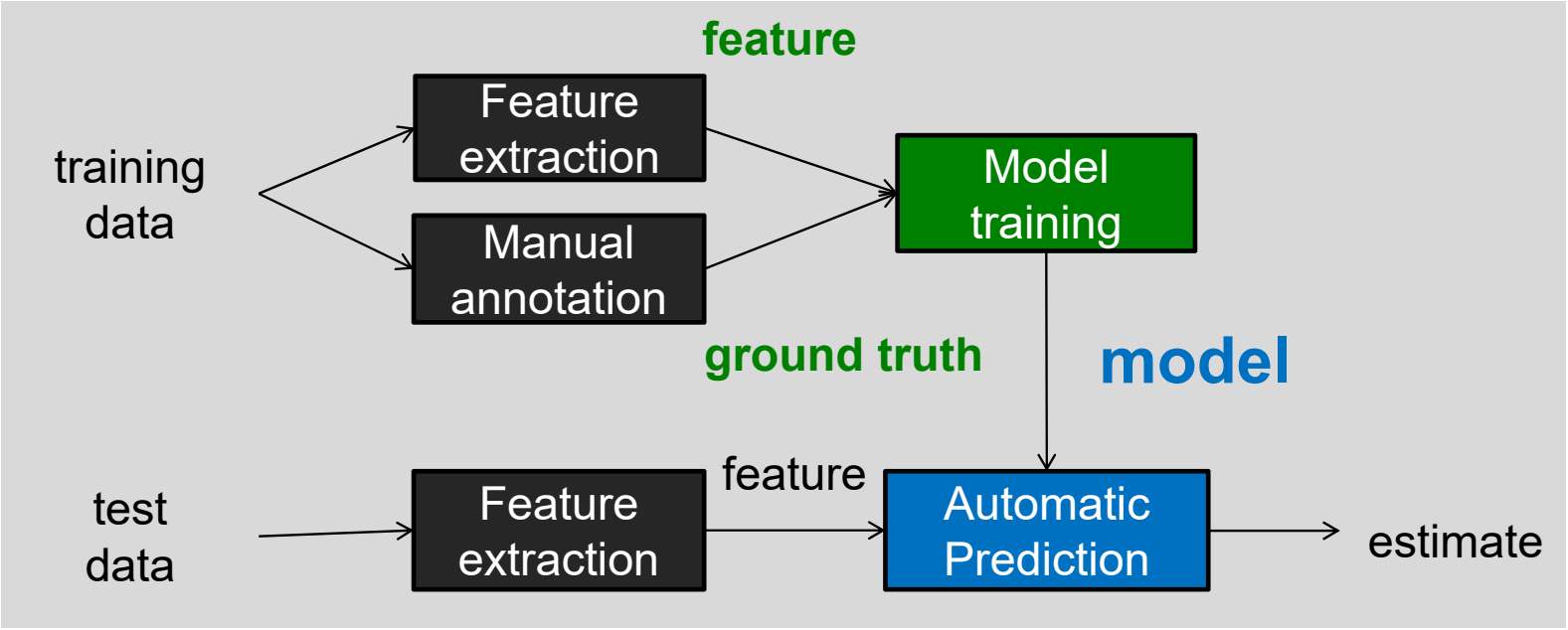
The GTZAN Dataset

https://www.tensorflow.org/datasets/catalog/gtzan

- "The dataset consists of 1000 audio tracks each 30 seconds long. It contains 10 genres, each represented by 100 tracks. The tracks are all 22,050Hz Mono 16-bit audio files in WAV format"
- The genres are:
  - blues
  - classical
  - country
  - disco
  - hiphop
  - jazz
  - metal
  - pop
  - reggae
  - rock

Representing the Audio as Features is Needed in ML/DL

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



The Three Feature Sets Used By GTZAN

- They are basically **statistics**

- **Timbral texture features:**
  Statistics from the waveform and magnitude spectrogram
  - Spectral centroid
  - Spectral rolloff
  - Spectral flux
  - Time domain zero crossings
  - MFCCs
  - Low-energy feature

- **Rhythmic content features:**
  Statistics from the result of a "beat estimator"

- **Pitch content features:**
  Statistics from the result of a "multi-pitch estimator"

The Rhythmic Content Features Used By GTZAN

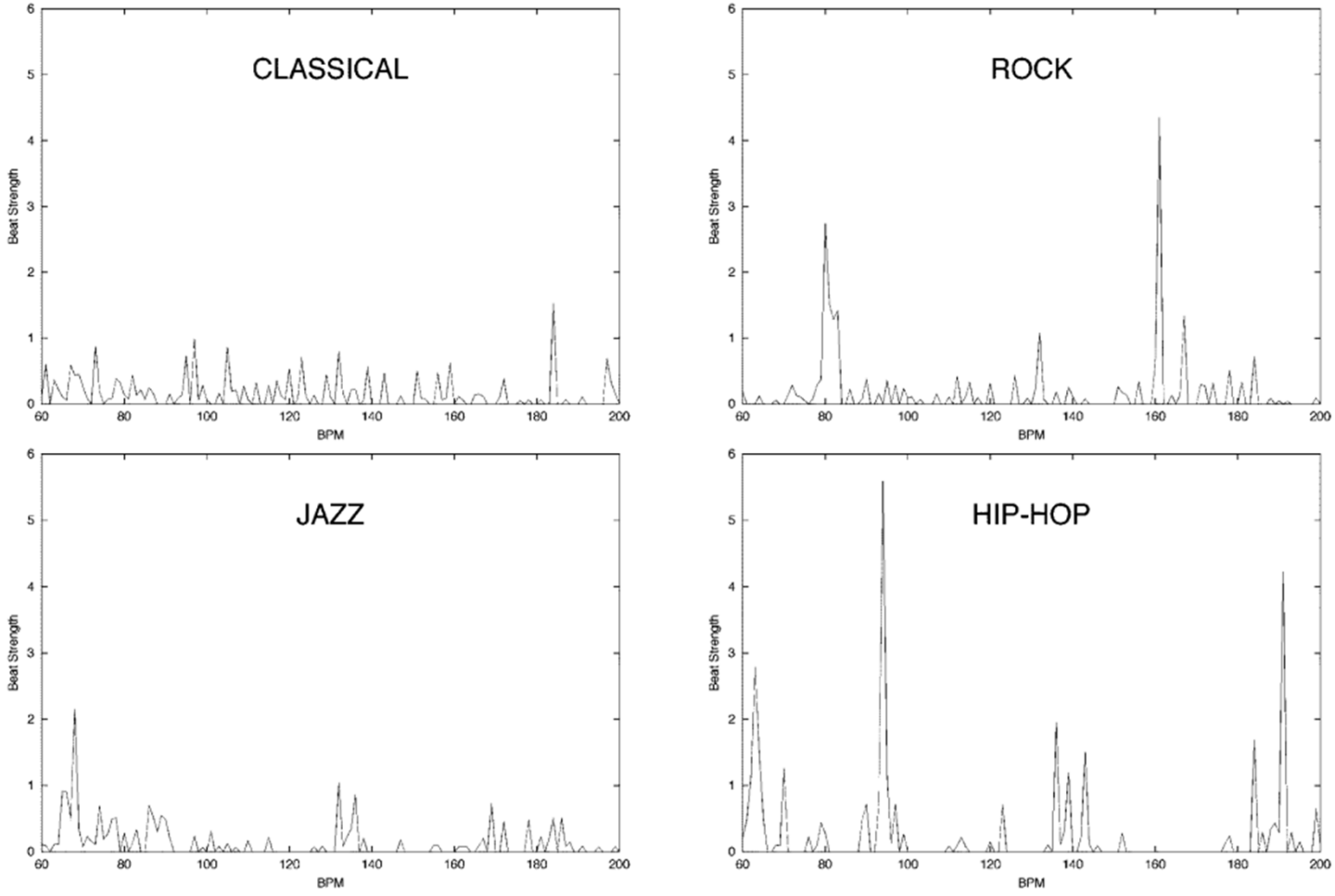
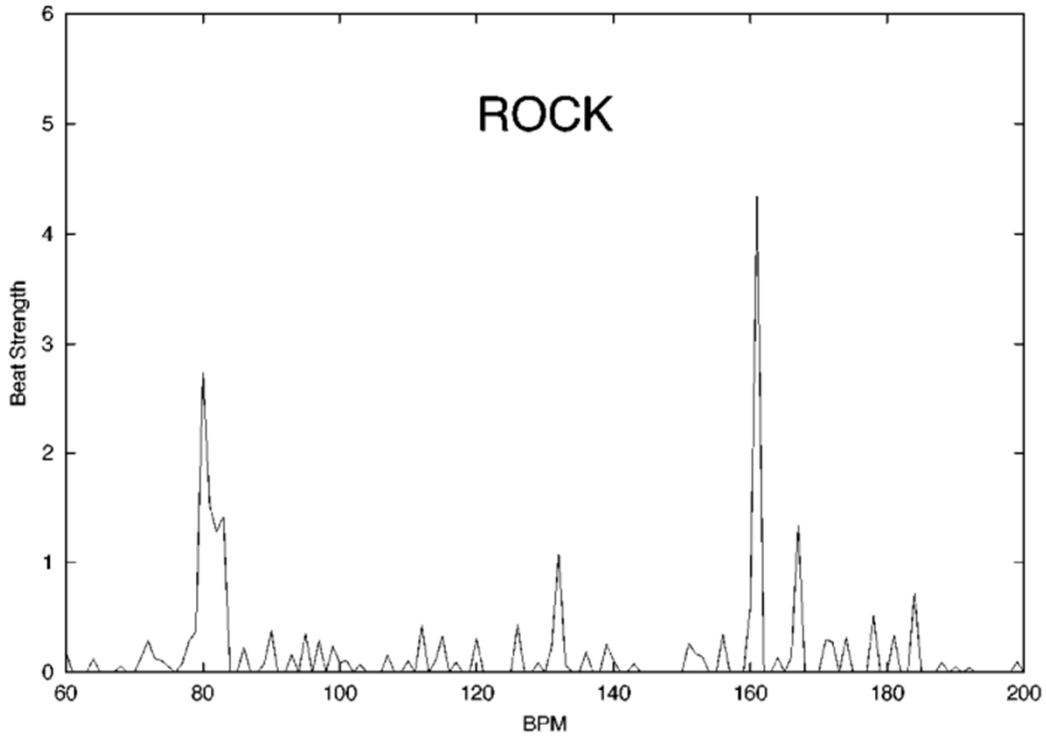


Fig. 2 shows a beat histogram for a 30-s excerpt of the song “Come Together” by the Beatles. The two main peaks of the BH correspond to the main beat at approximately 80 bpm and its first harmonic (twice the speed) at 160 bpm. Fig. 3 shows four beat histograms of pieces from different musical genres. The upper left corner, labeled classical, is the BH of an excerpt from “La Mer” by Claude Debussy. Because of the complexity of the multiple instruments of the orchestra there is no strong self-similarity and there is no clear dominant peak in the histogram. More strong peaks can be seen at the lower left corner, labeled jazz, which is an excerpt from a live performance by Dee Dee Bridgewater. The two peaks correspond to the beat of the song (70 and 140 bpm). The BH of Fig. 2 is shown on the upper right corner where the peaks are more pronounced because of the stronger beat of rock music

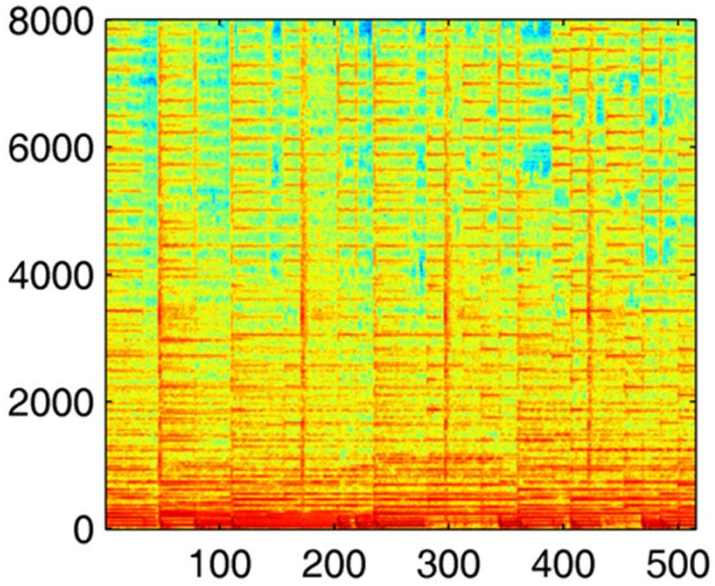
The Rhythmic Content Features Used By GTZAN

- **Rhythmic content features:**
  Statistics from a "beat histogram"
  - A0, A1: relative amplitude (divided by the sum of amplitudes) of the first, and second histogram peak
  - RA: ratio of the amplitude of the second peak divided by the amplitude of the first peak
  - P1, P2: period of the first, second peak in bpm
  - SUM: overall sum of the histogram (indication of beat strength)



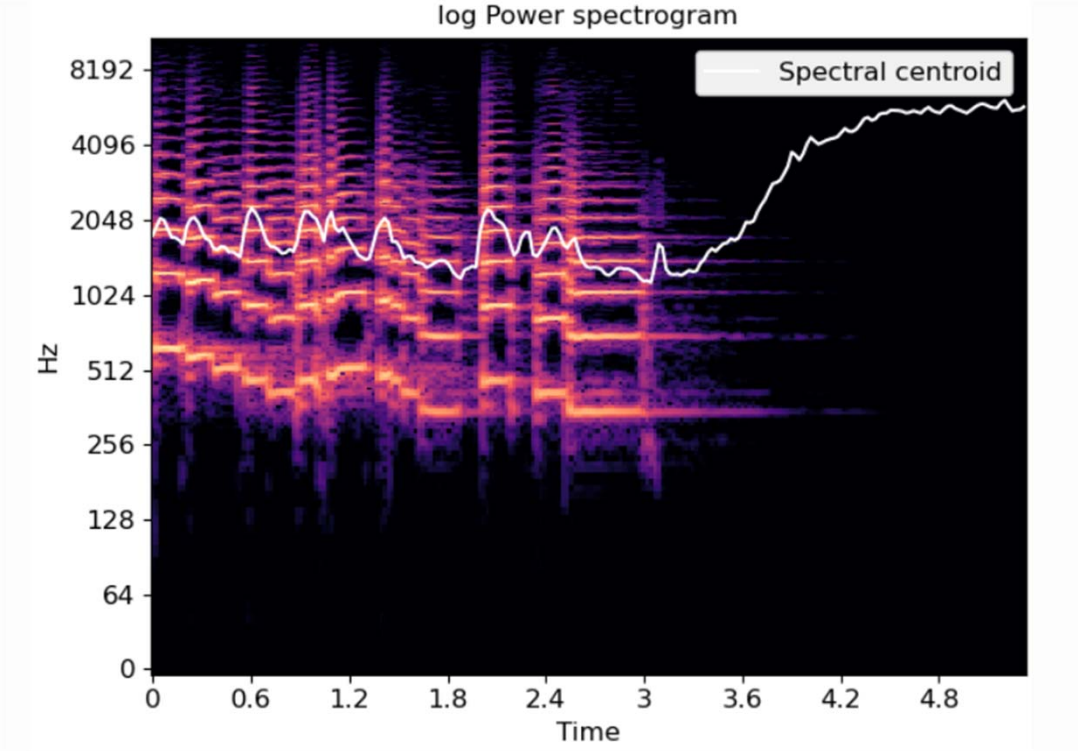
The Timbral Texture Features Used By GTZAN

- Question: how to compute features/statistics from the spectrogram (or waveform)?
  - The number of features cannot be too large
  - The features have to be somehow “meaningful”



Spectral Centroid

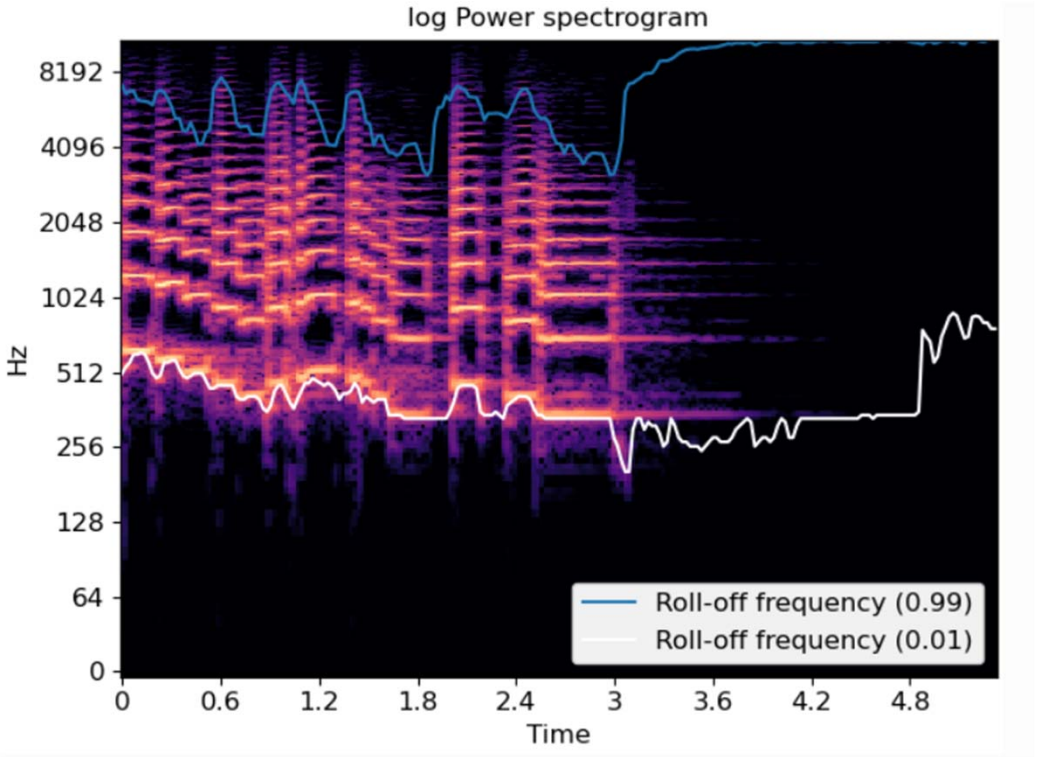
- Each frame of a magnitude spectrogram is normalized and treated as a distribution over frequency bins, from which the mean (centroid) is extracted per frame
- A measure of spectral shape
- Higher centroid values imply "**brighter**" textures with more high frequencies
- Other statistics can also be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



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

Spectral 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

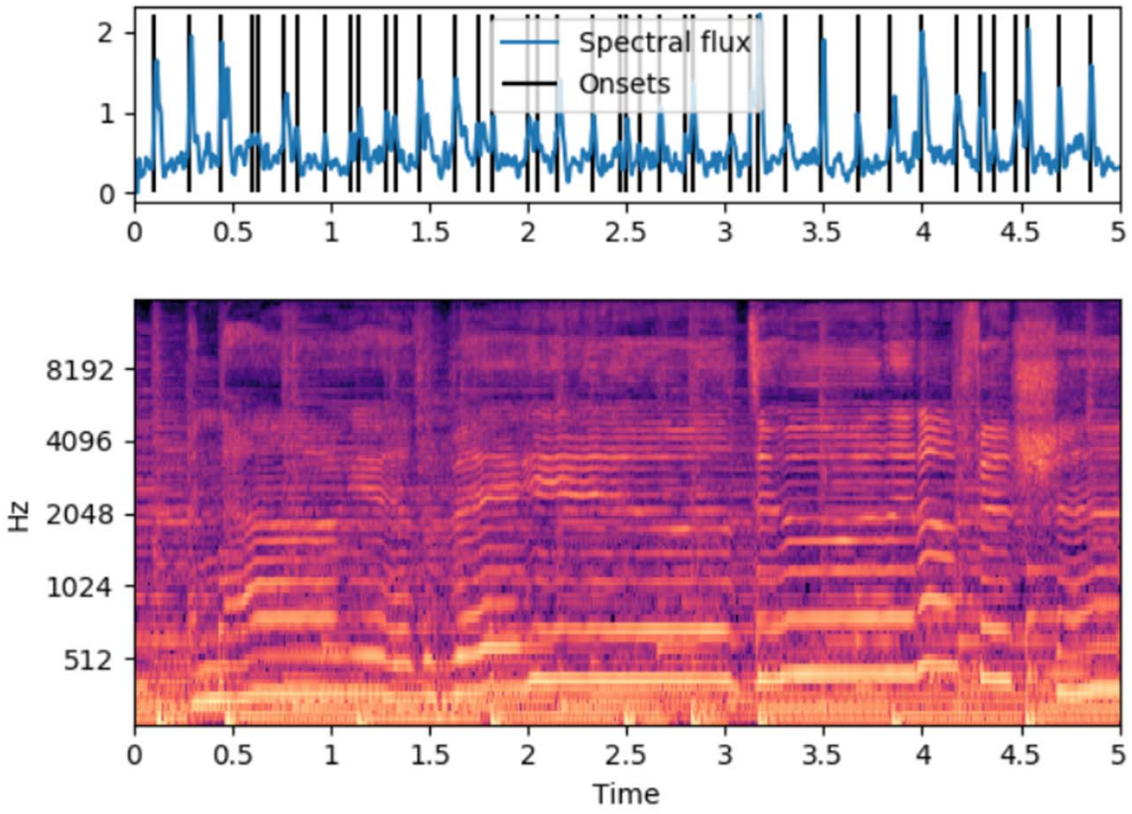
Spectral Contrast

- Each frame of a spectrogram is divided into multiple **sub-bands**
- Compute the mean energy for each sub-band
- Compare 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**
- Alternatively: entropy of the sub-band mean energy

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-difference** between two adjacent normalized spectra
- BTW, this feature is also often used for musical onset detection (more related to rhythm)

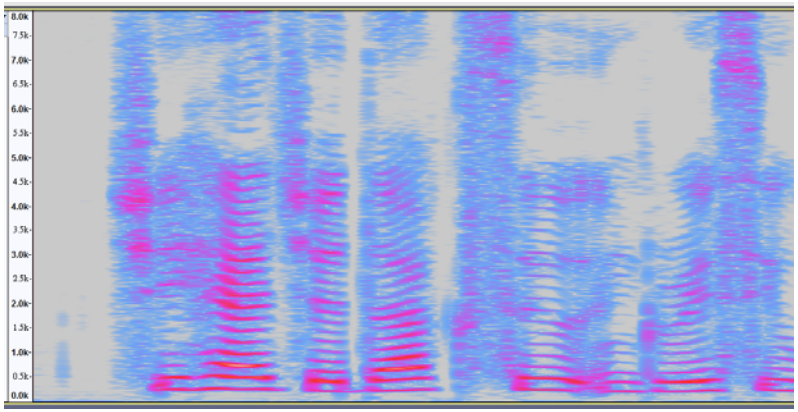


Ref1: https://en.wikipedia.org/wiki/Spectral_flux
Ref2: https://librosa.org/librosa_gallery/auto_examples/plot_superflux.html

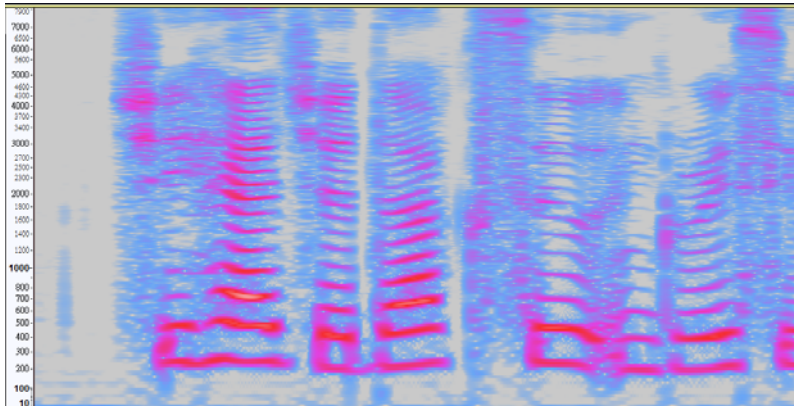
Mel-Spectrogram

- 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: *hundreds* of frequency bins

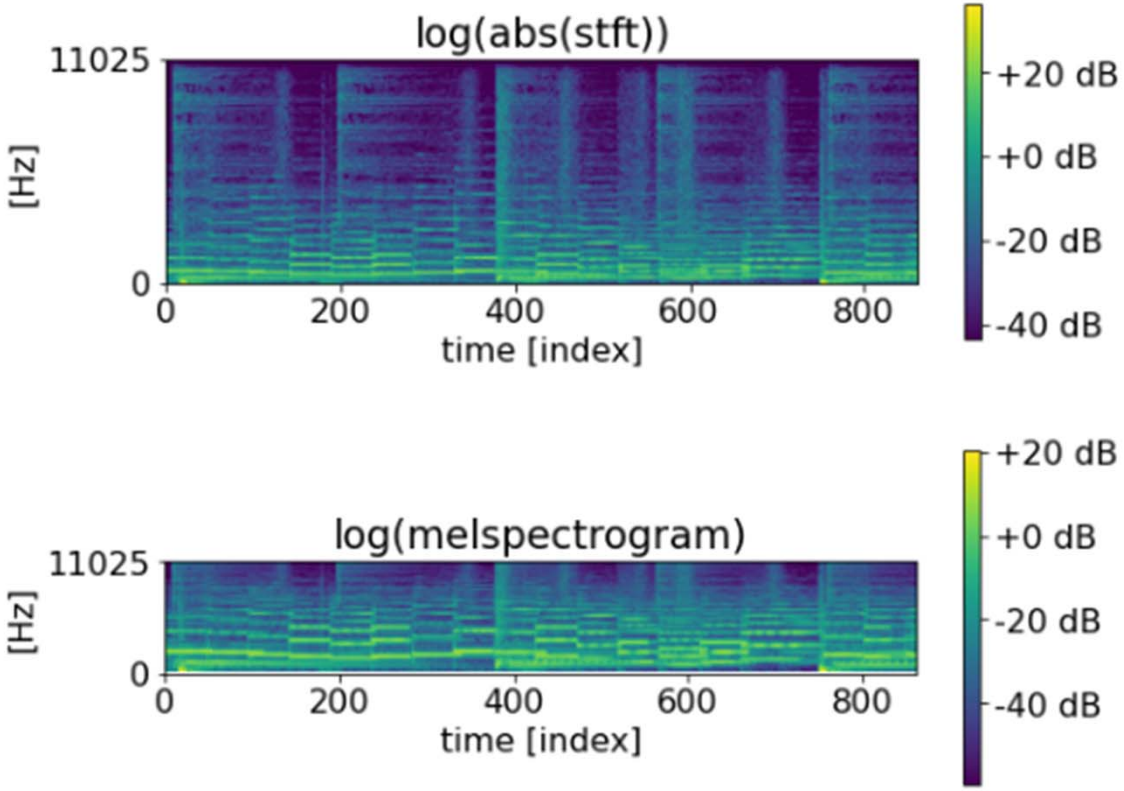


mel scale: *tens* of frequency bands



Mel-Spectrogram

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



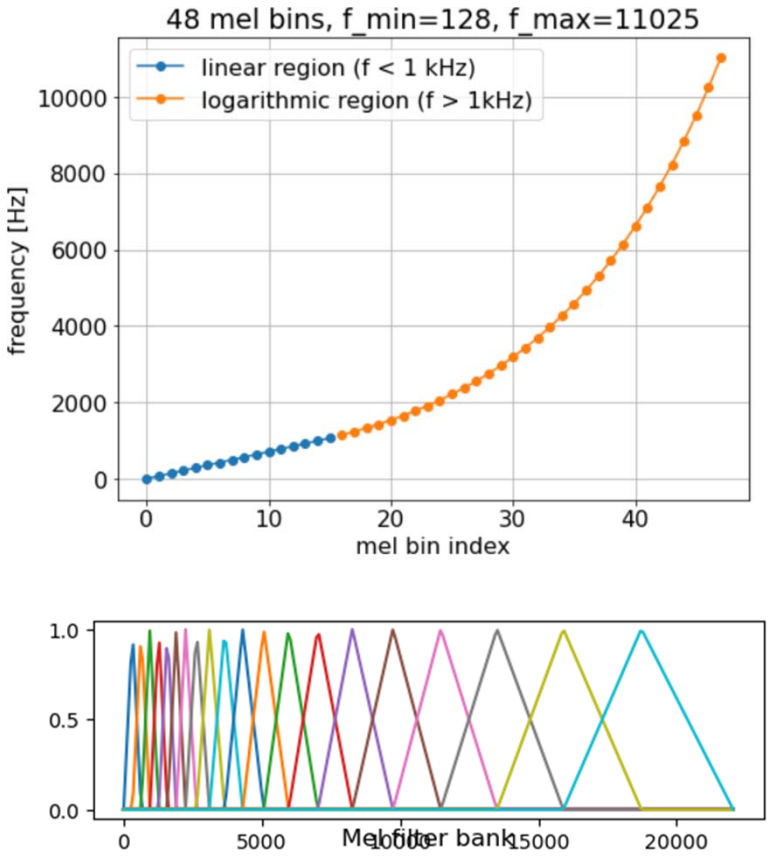
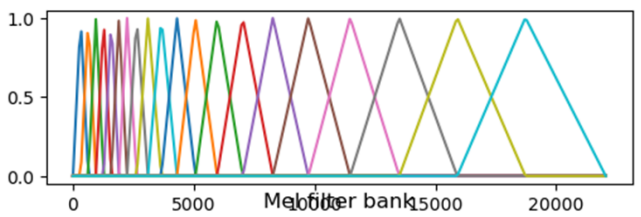
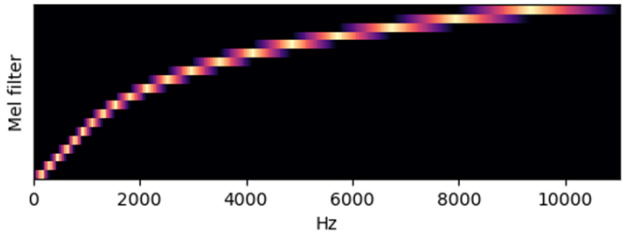


Figure from librosa

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

The GTZAN Classifier

- Each 30-sec audio is represented as **a single vector** (that is composed of the timbre, rhythmic and pitch features)
- Then train a classifier using Gaussian mixture model (GMM) or K-nearest neighbor (K-NN)
- Confusion matrix
  - Columns: GT / rows: predictions
  - "Classical music is misclassified as jazz music for pieces with strong rhythm from composers like Leonard Bernstein and George Gershwin."
  - "Rock music has the worst classification accuracy and is easily confused with other genres which is expected because of its broad nature."

TABLE II
GENRE CONFUSION MATRIX

	cl	co	di	hi	ja	ro	bl	re	po	me
cl	**69**	0	0	0	1	0	0	0	0	0
co	0	**53**	2	0	5	8	6	4	2	0
di	0	8	**52**	11	0	13	14	5	9	6
hi	0	3	18	**64**	1	6	3	26	7	6
ja	26	4	0	0	**75**	8	7	1	2	1
ro	5	13	4	1	9	**40**	14	1	7	33
bl	0	7	0	1	3	4	**43**	1	0	0
re	0	9	10	18	2	12	11	**59**	7	1
po	0	2	14	5	3	5	0	3	**66**	0
me	0	1	0	1	0	4	2	0	0	**53**

BTW, the Mel-Spectrogram is Actually Quite Important…

- 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"
- Used in all types of tasks, for both music analysis and generation

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 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"
```

More on 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])

More on 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)

More on 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

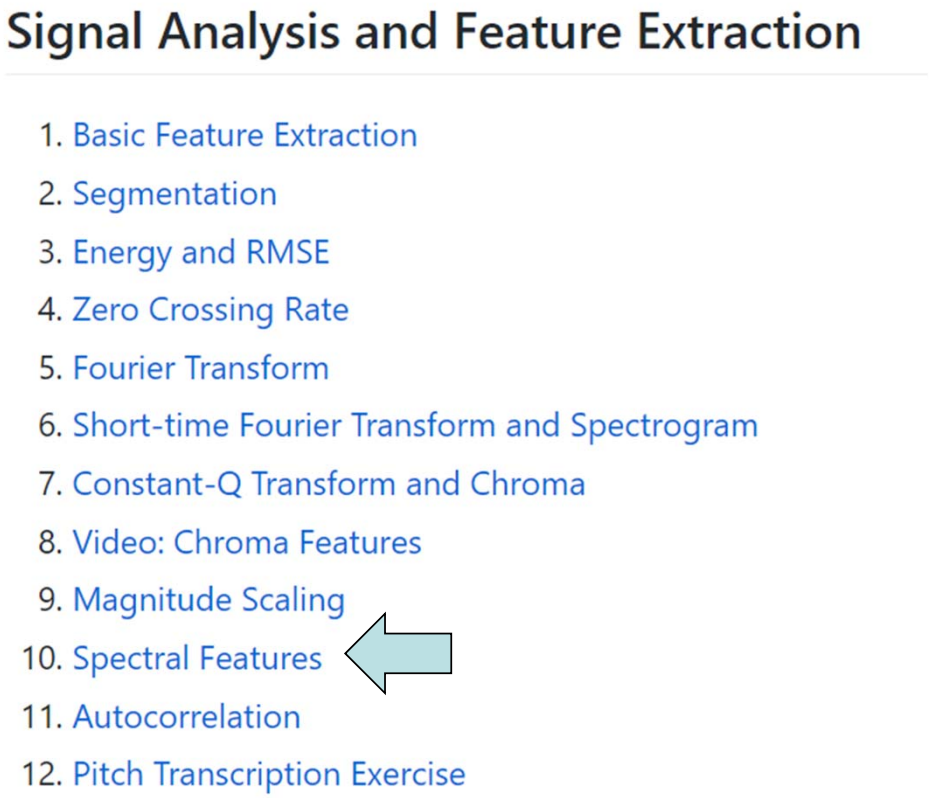
- `pyin` (y, *, fmin, fmax[, sr, frame_length, ...])
- `yin` (y, *, fmin, fmax[, sr, frame_length, ...])
- `estimate_tuning` (*[, y, sr, S, n_fft, ...])

Rhythm features

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

Demonstrations 1

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

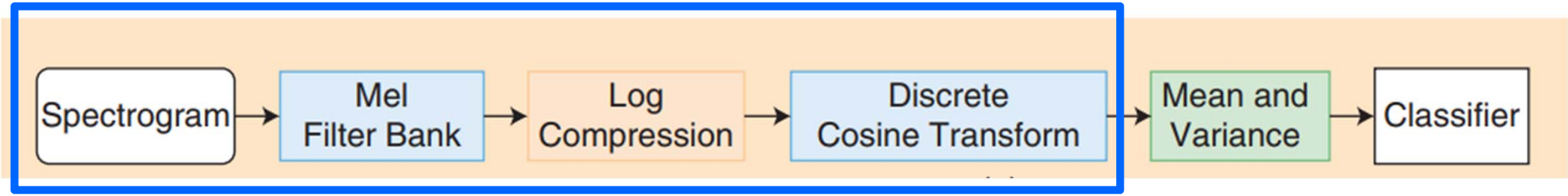


Demonstrations 2

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)
```

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

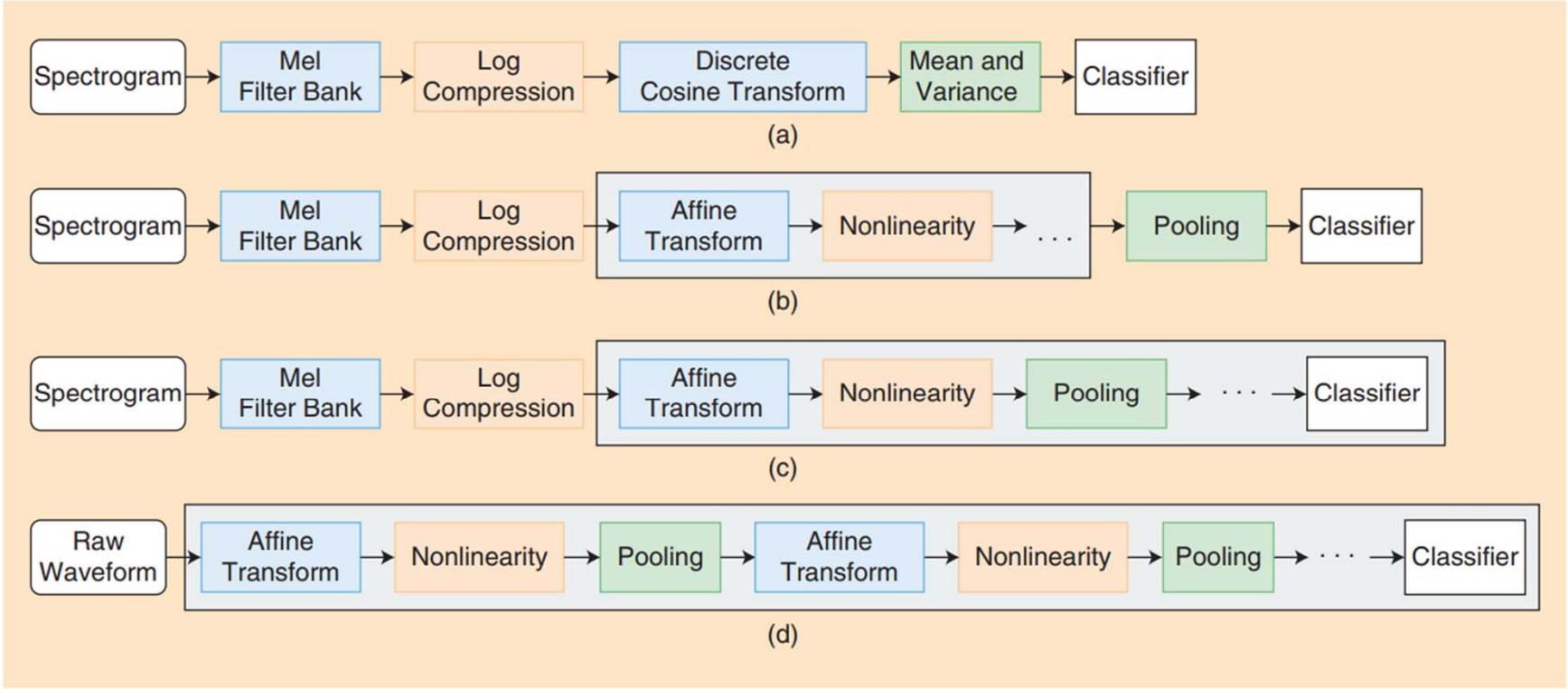
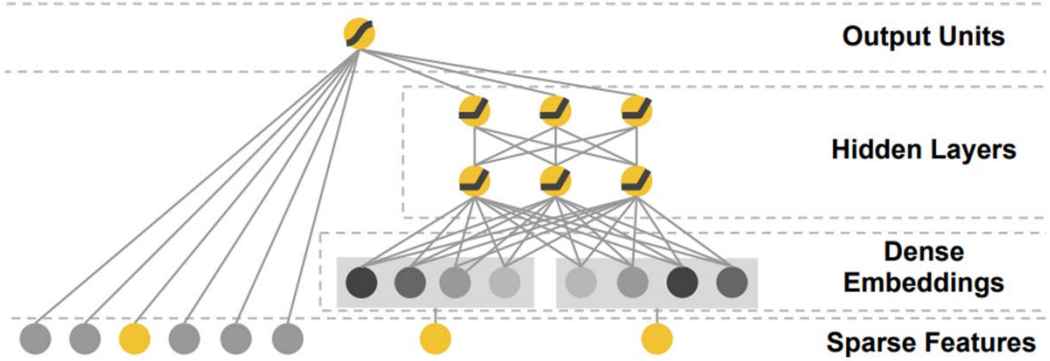


FIGURE 1. The transition of feature representation for music classification: (a) feature engineering [mel-frequency cepstral coefficients (MFCCs)], (b) low-level feature learning, (c) convolutional neural networks, and (d) end-to-end learning. The blocks inside the black lines indicate that they are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

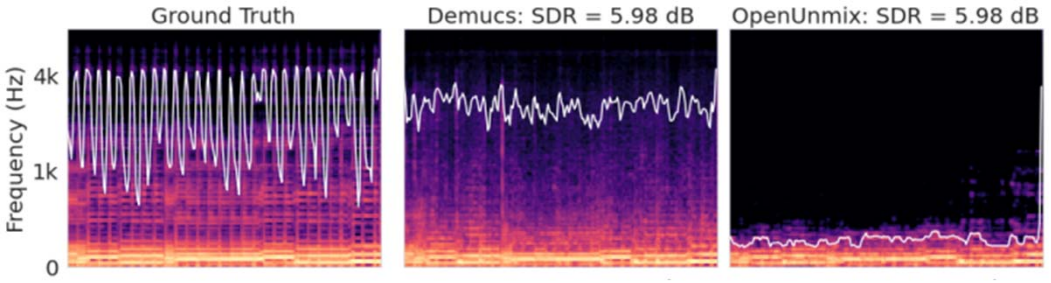
Ref: Nam et al., “Deep learning for audio-based music classification and tagging,” IEEE Signal Processing Magazine, 2019

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

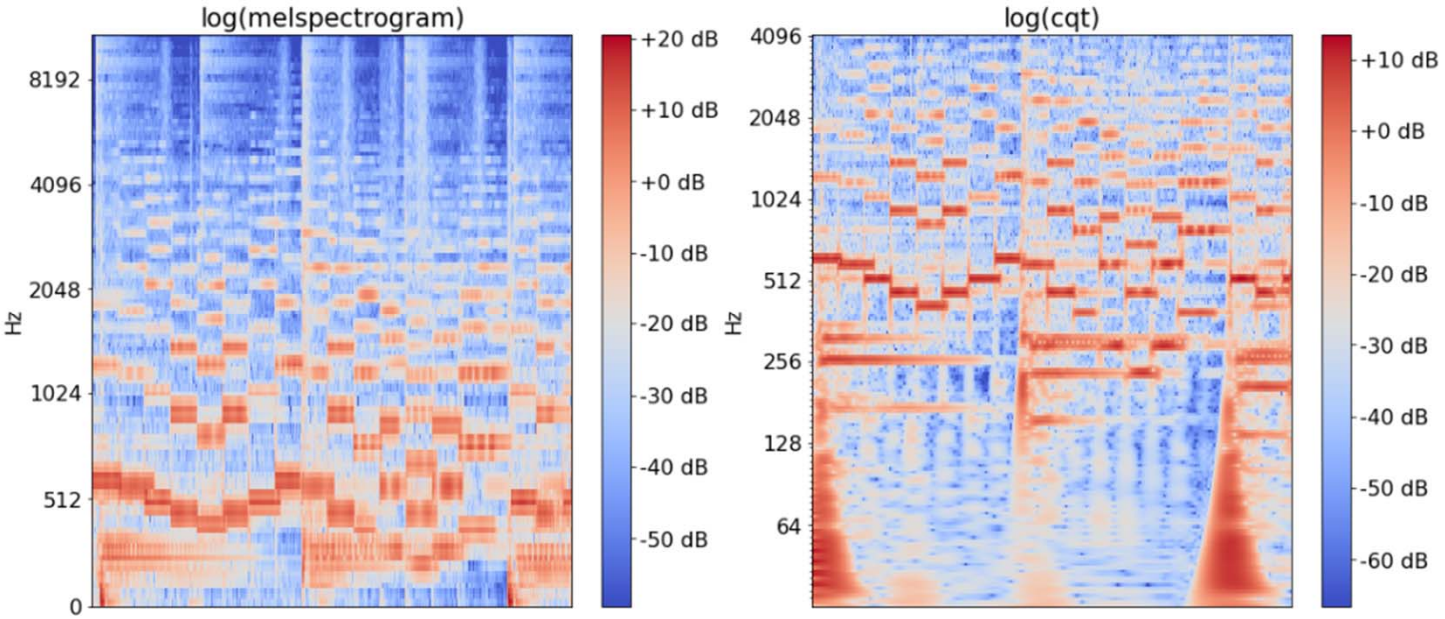


Ref: Schaffer et al., “Music separation enhancement with generative modeling,” ISMIR 2022

Moving Beyond STFT: 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)

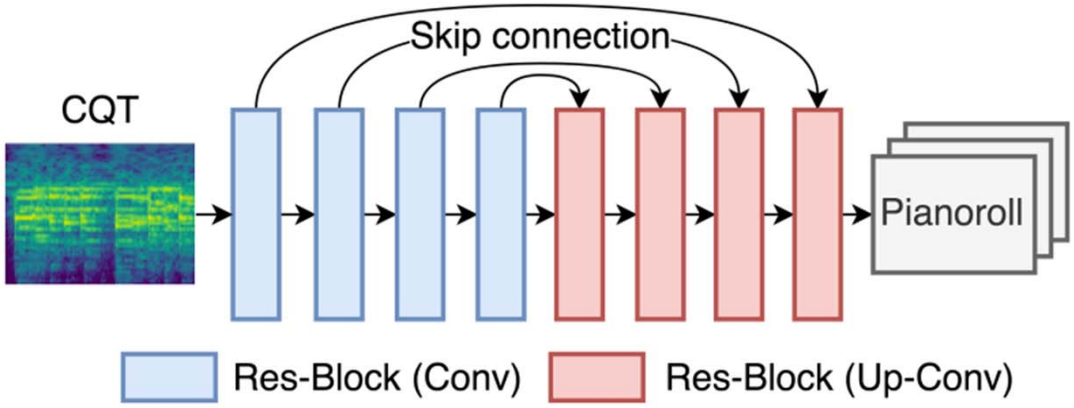


- **Logarithmic** frequency, **logarithmic** magnitude

Constant-Q Transform (CQT)

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

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
)
```

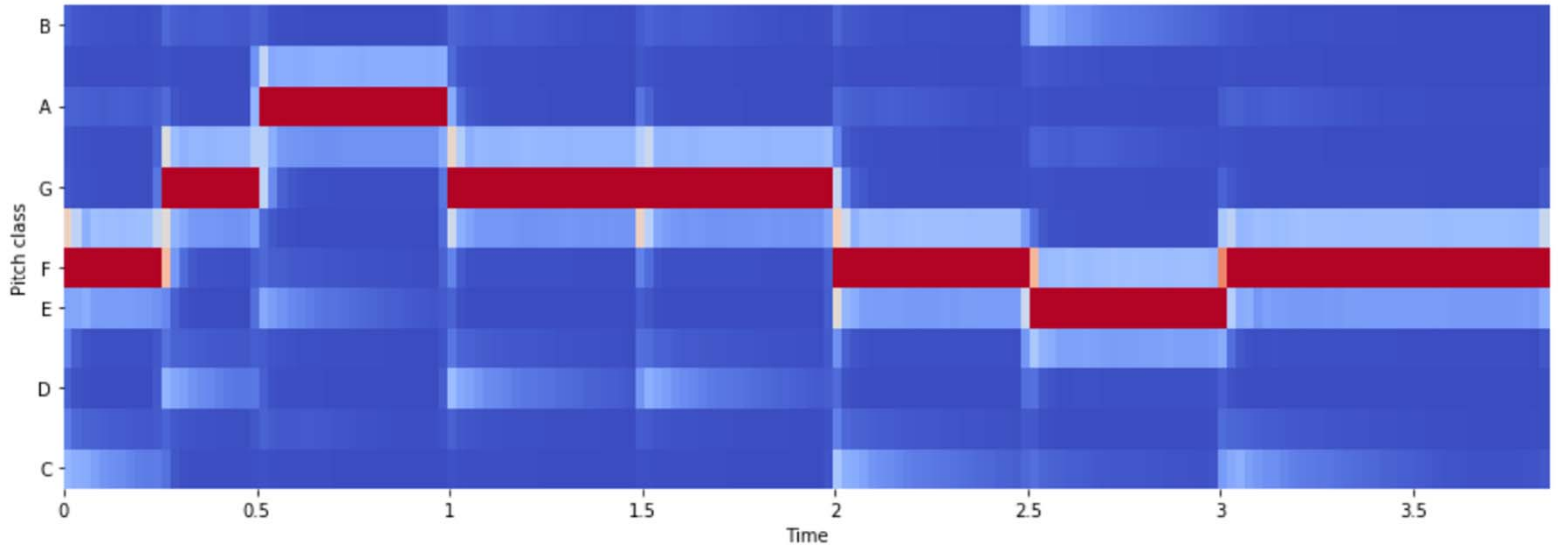
https://github.com/archinetai/cqt-pytorch

https://github.com/eloimoliner/CQT_pytorch

Pitch Class Profile / Chromagram

https://musicinformationretrieval.com/chroma.html

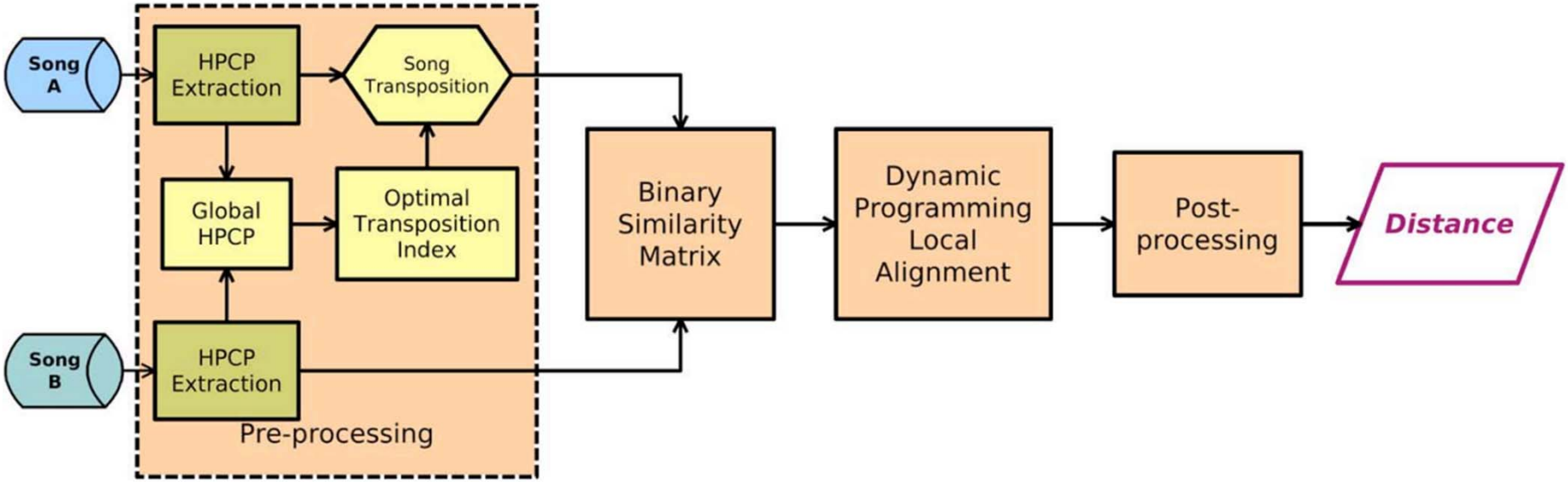
- "A **chroma vector** (Wikipedia) is a typically a 12-element feature vector indicating how much energy of each pitch class, {C, C#, D, D#, E, ..., B}, is present in the signal"
  - i.e., ignore octaves



Pitch Class Profile / Chromagram

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

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



Ref1: Muller et al, “Audio matching via chroma-based statistical features,” ISMIR 2005
Ref2: Serra et al, “Chroma binary similarity and local alignment applied to cover song identification,” TASLP 2008

Pitch Class Profile / Chromagram

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

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

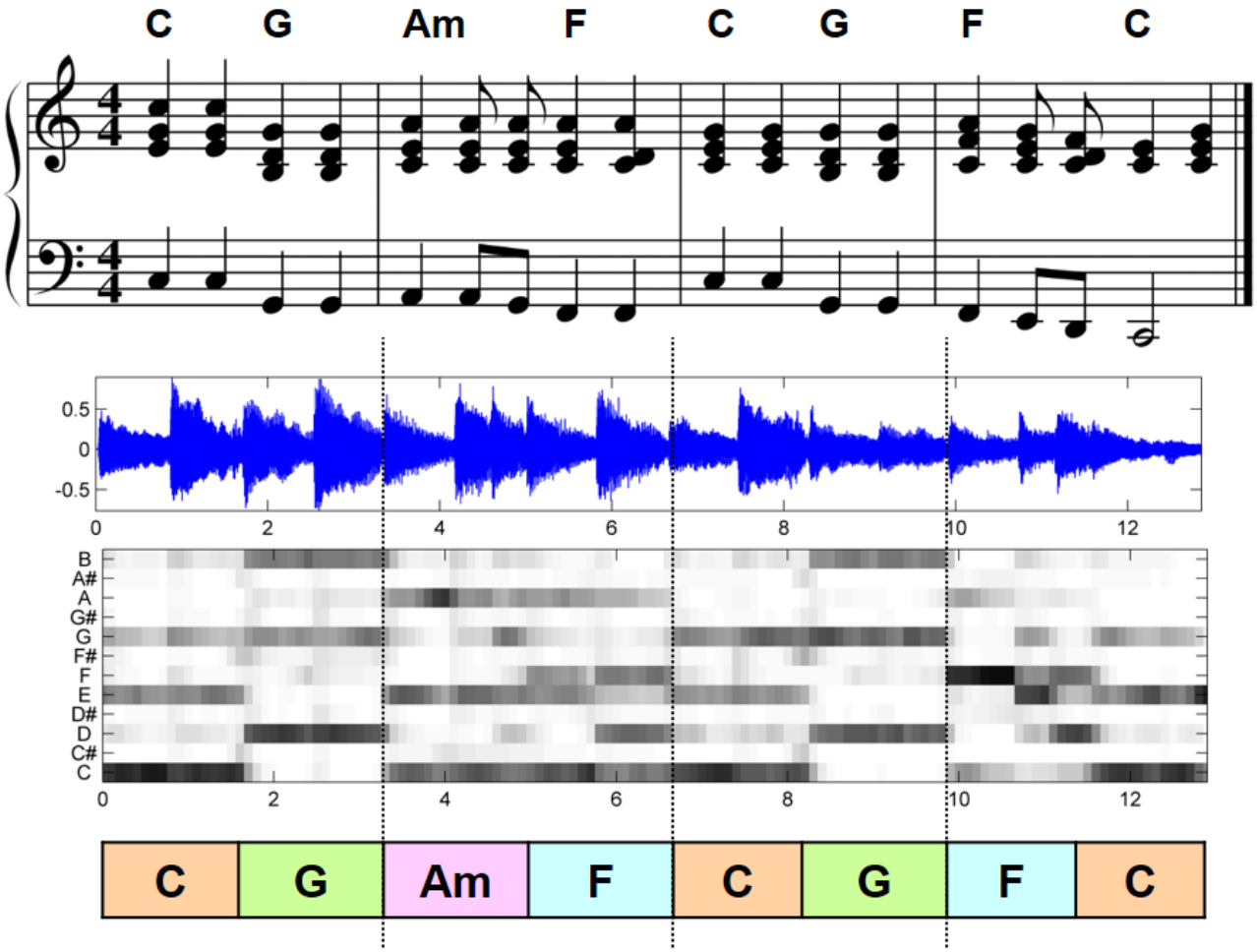


Figure 5.1 from [Müller, FMP, Springer 2015]

ISMIR 2025 Test-of-Time (ToT) Awardee

Different Features are Needed for Different Tasks

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

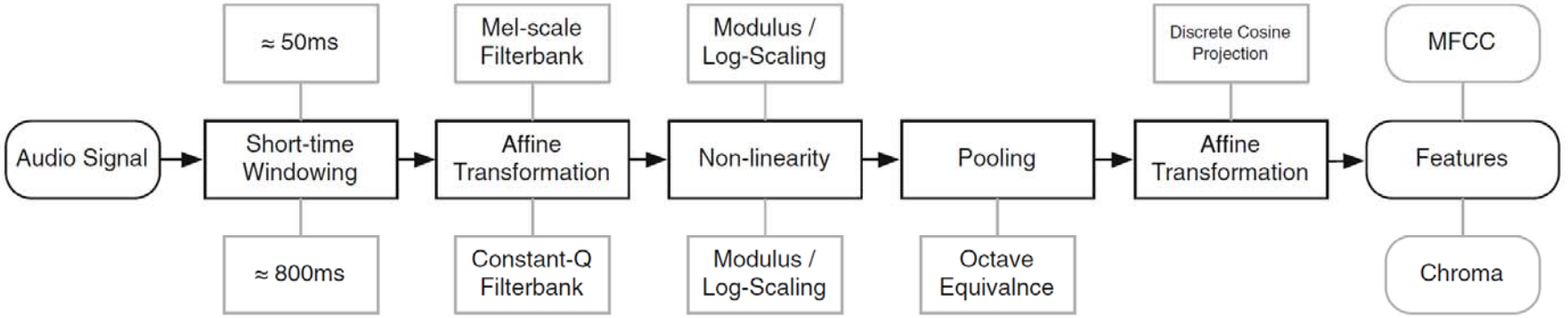
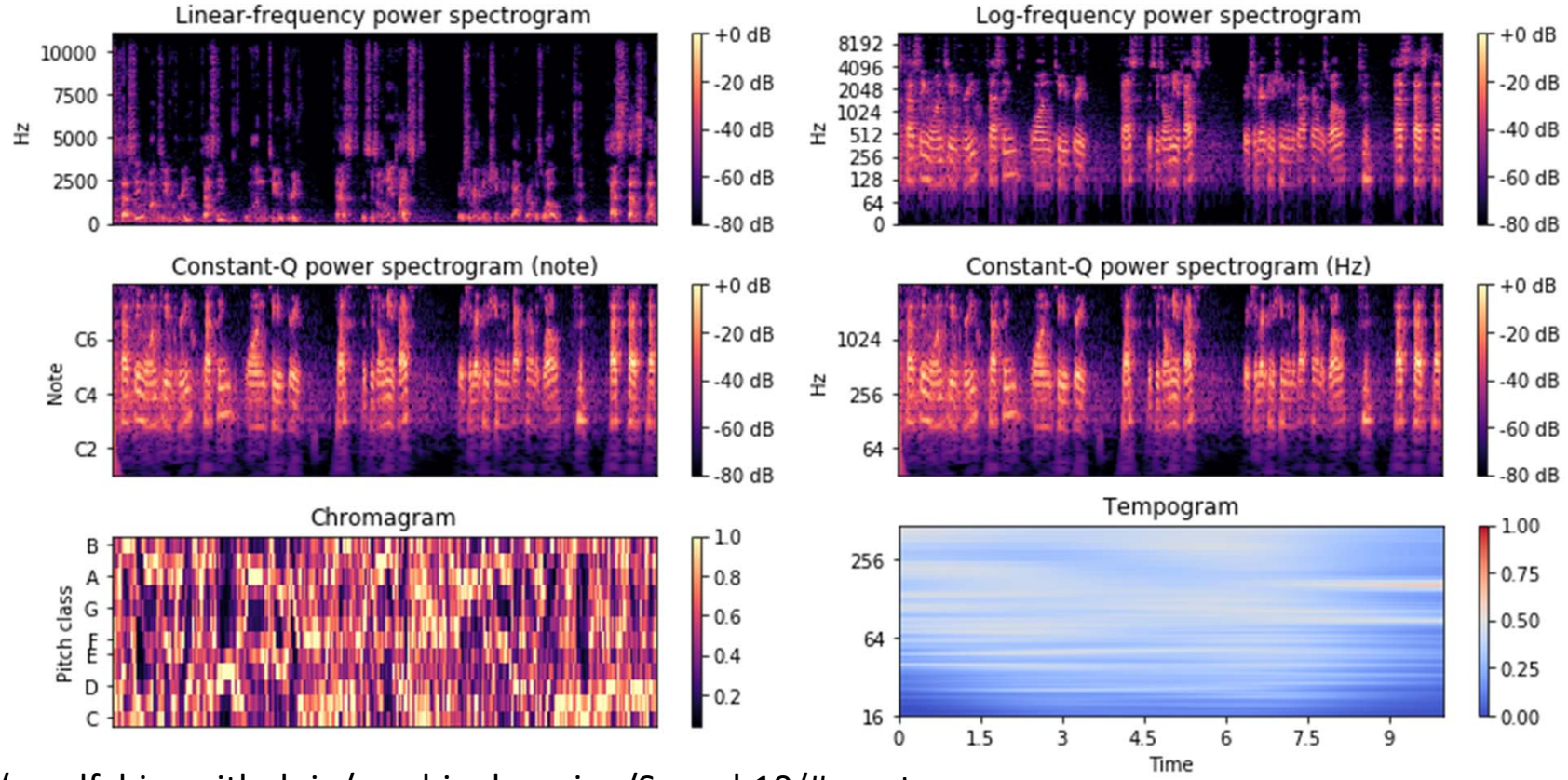


Fig. 3 *State of the art*: standard approaches to feature extraction proceed as the cascaded combination of a few simpler operations; on closer inspection, the main difference between chroma and MFCCs is the parameters used

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

Different Features are Needed for Different Tasks

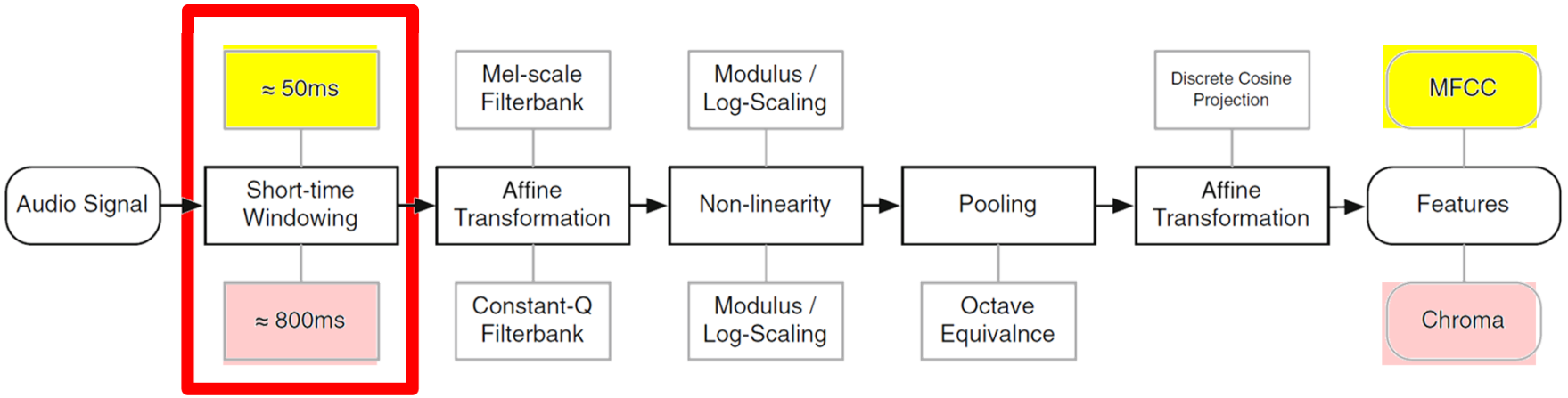
- We can **combine** different features to train a classifier for tasks such as music *genre* classification or music *emotion* classification



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

Recap: Math in STFT

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

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

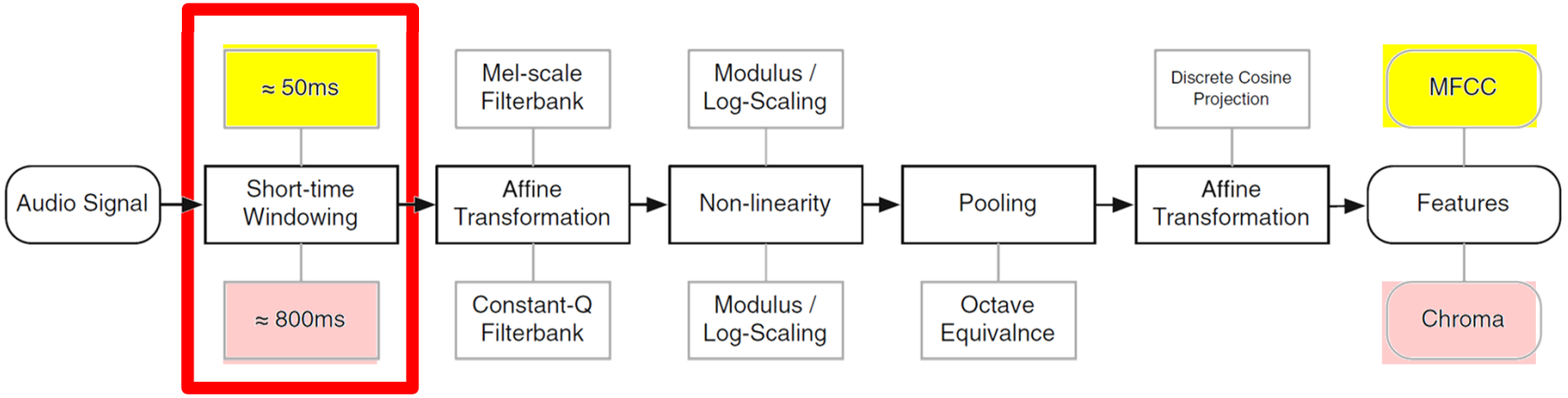
(16th note in 4/4 meter: 125 ms)

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 (often seen in DL papers)

> 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

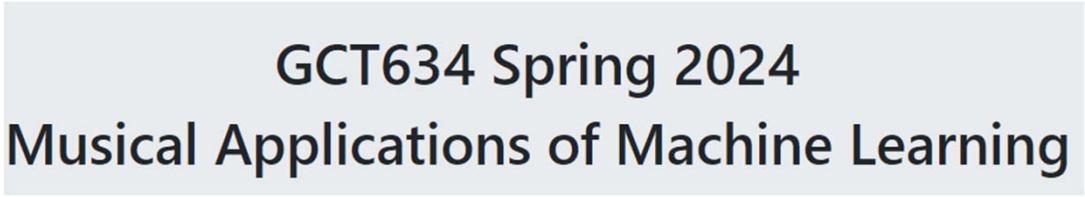
Outline

- Music classification: Basics
- ML-based music classification (and hand-crafted audio features)
- **DL-based music classification**
- Music foundation models

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

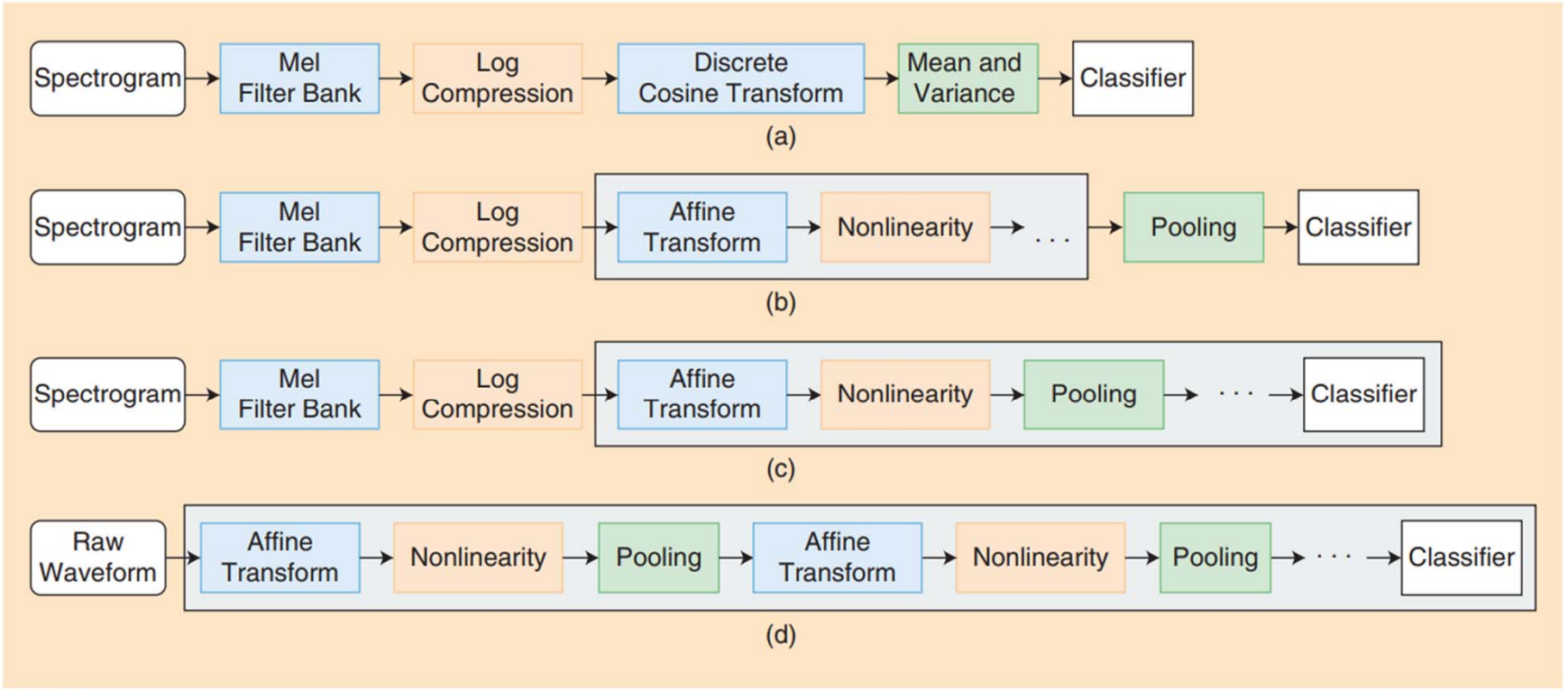


FIGURE 1. The transition of feature representation for music classification: (a) feature engineering [mel-frequency cepstral coefficients (MFCCs)], (b) low-level feature learning, (c) convolutional neural networks, and (d) end-to-end learning. The blocks inside the black lines indicate that they are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

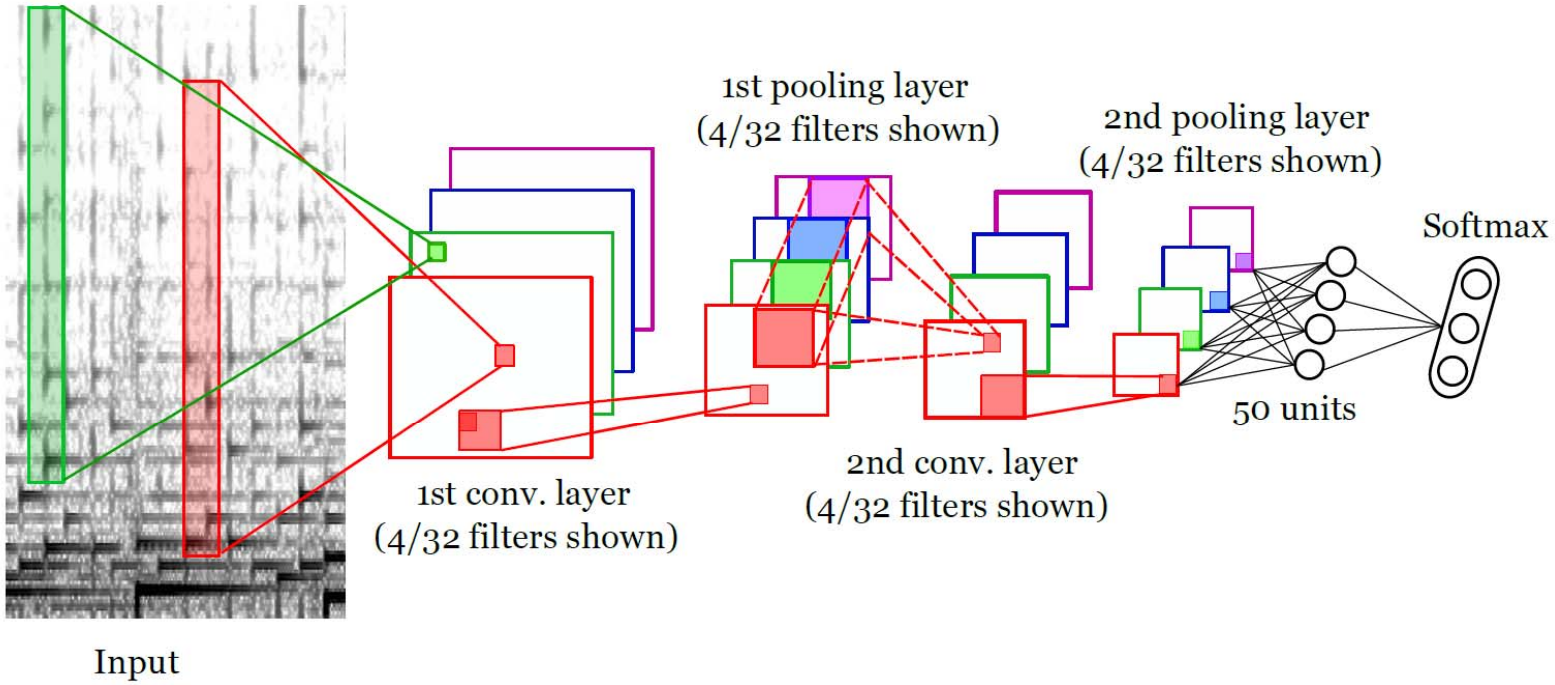


Fig. 1. Illustration of the CDNN architecture we use for our experiments. The CDNN first applies narrow vertical filters to the input sonogram (left) to capture harmonic structure. Then, it applies 32 different filters in the first convolutional layer (we show only 4). This is followed by the first max-pooling 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.

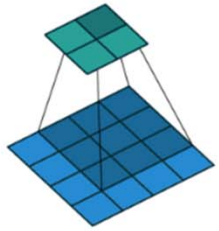
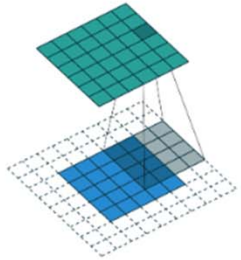
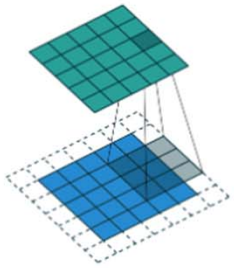
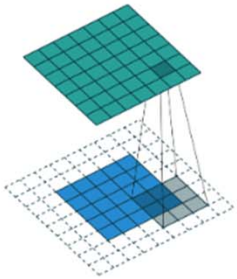
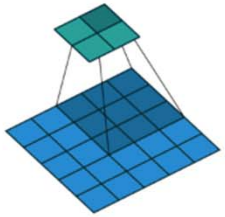
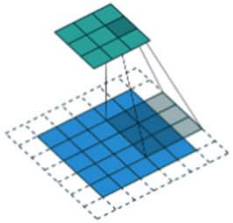
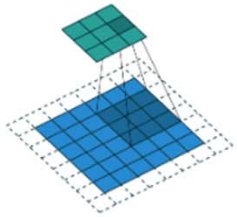
Ref: Kereliuk et al, "Deep learning and music adversaries," IEEE Trans. Multimedia, 2015

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)	

Padding & Stride

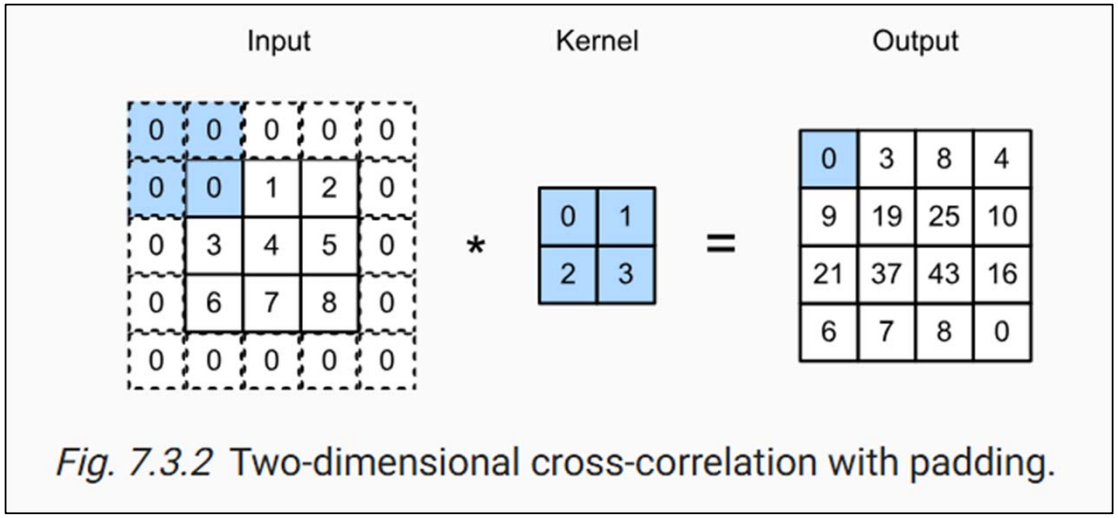


Fig. 7.3.2 Two-dimensional cross-correlation with padding.

From:
https://d2l.ai/chapter_convolutional-neural-networks/padding-and-strides.html

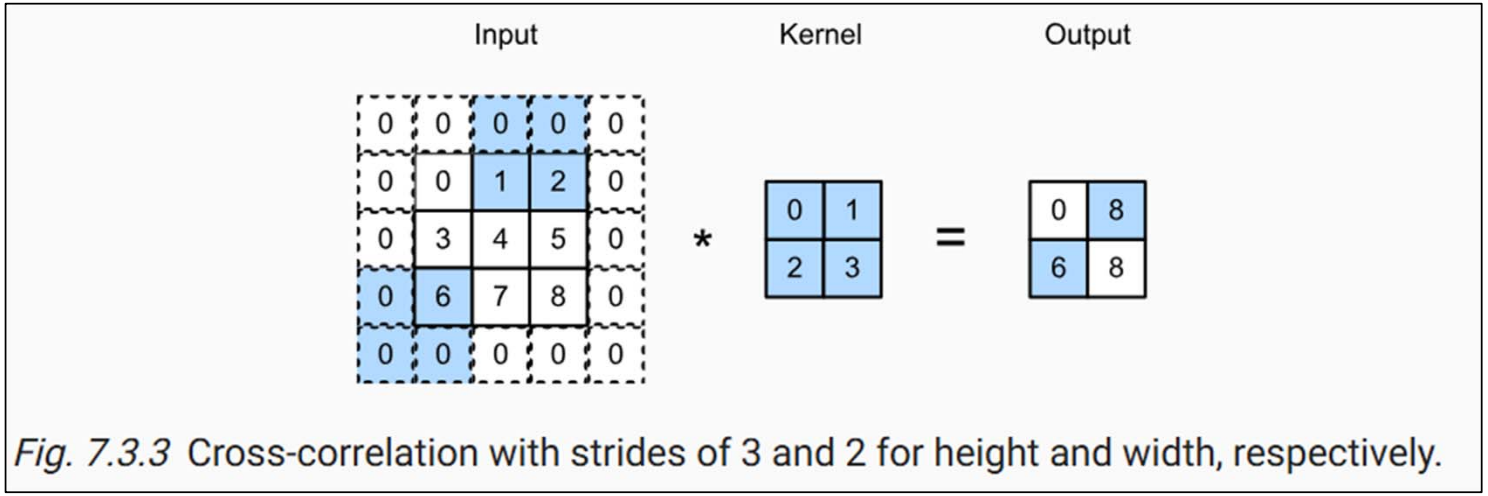


Fig. 7.3.3 Cross-correlation with strides of 3 and 2 for height and width, respectively.

Pooling

From:
https://d2l.ai/chapter_convolutional-neural-networks/pooling.html#pooling

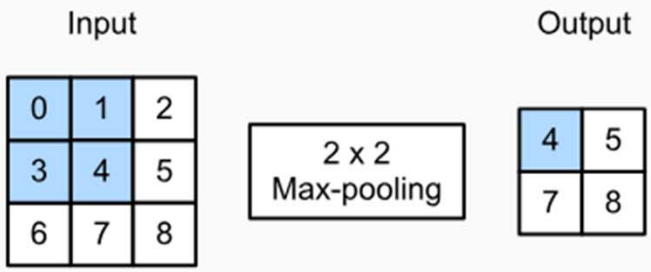
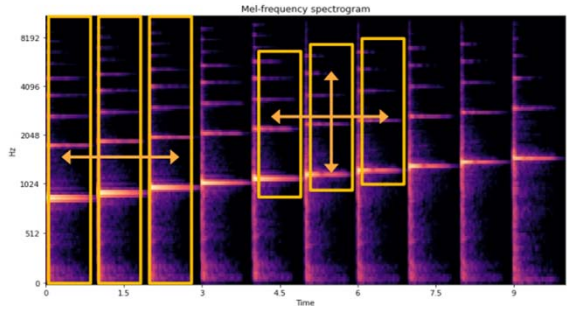


Fig. 7.5.1 Max-pooling with a pooling window shape of 2×2. The shaded portions are the first output element as well as the input tensor elements used for the output computation: $\max(0, 1, 3, 4) = 4$.

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

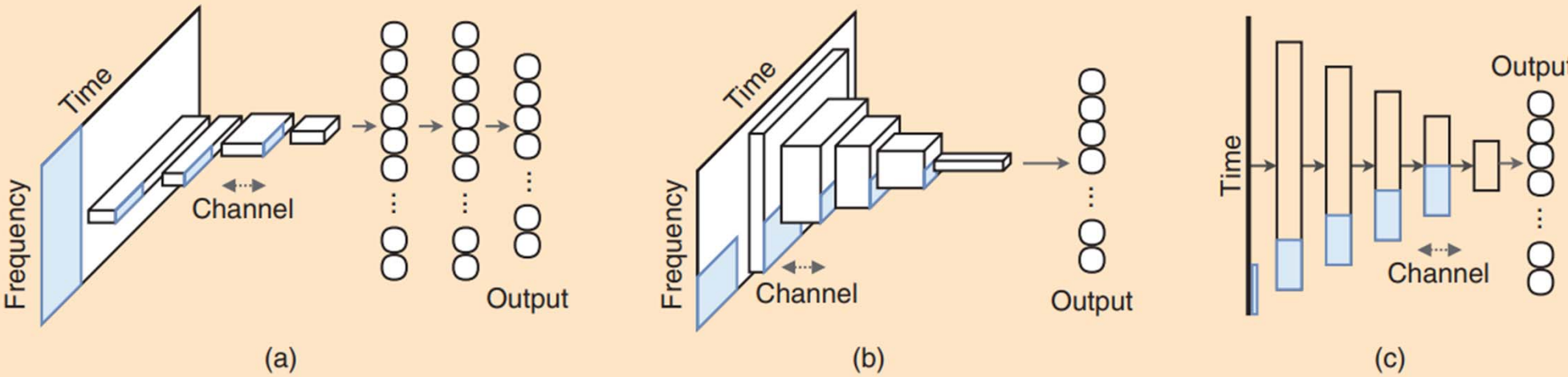
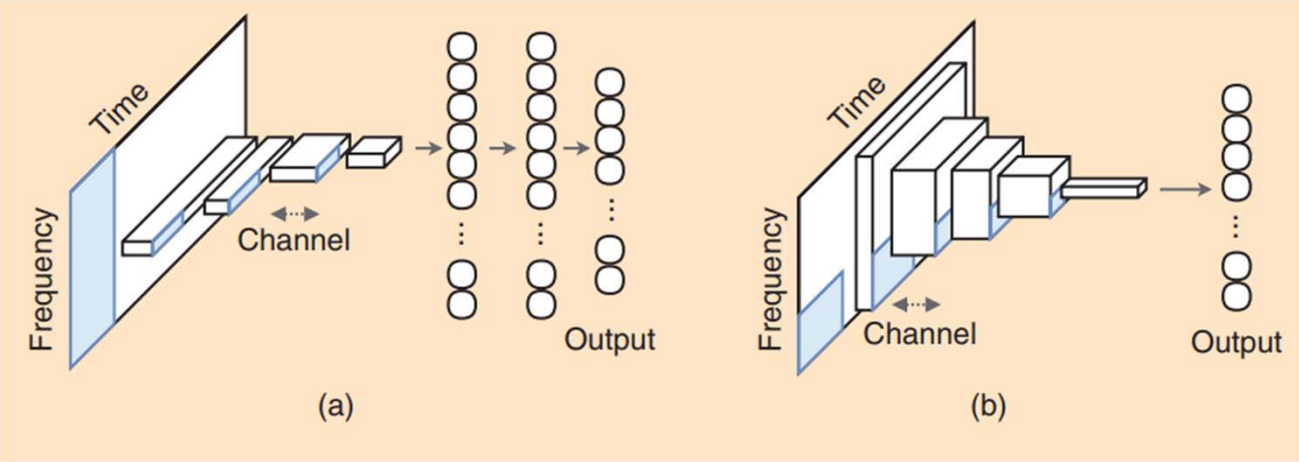


FIGURE 2. Block diagrams of (a) 1-D, (b) 2-D, and (c) sample-level CNNs. (a) and (b) are based on 2-D time–frequency representation inputs (e.g., mel-spectrograms or short-time Fourier transforms), and (c) is based on a time-series input.

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

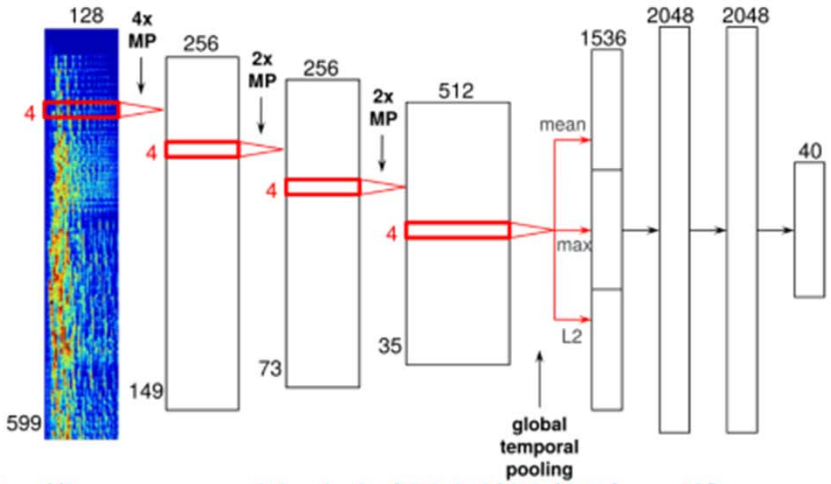
1D CNNs vs. 2D CNNs



- **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
  - Might be pitch-invariant

1D CNN

- Front-end: 1D CNN
  - Mel-spectrogram input filter with 128 (mel bin) x 4 (frames)
  - Feature maps (256 → 256 → 256 → 512), max-pooling in time (4 → 2 → 2)
- Back-end: global pooling (temporal summary) with mean, max, and L2



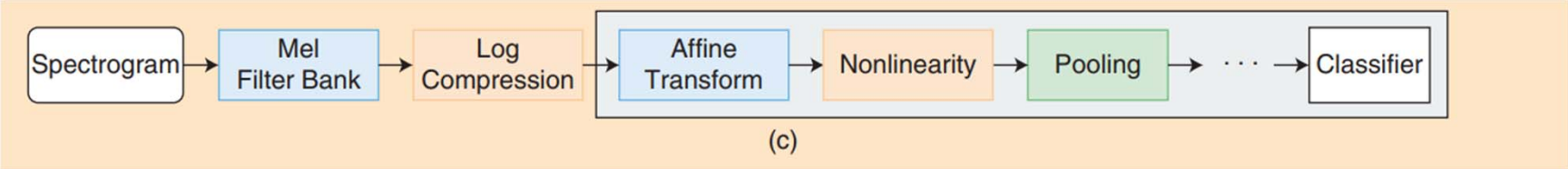
http://benanne.github.io/2014/08/05/spotify-cnns.html

(From Prof. Nam' slides)

Deep content-based music recommendation, Aaron van den Oord, Sander Dieleman, Benjamin Schrauwen, NIPS, 2013

Input Audio Representation

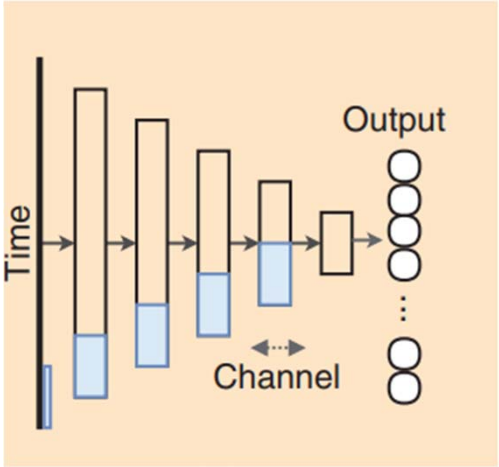
https://music-classification.github.io/tutorial/part2_basics/input-representations.html



- **Log-magnitude spectrogram**
  - Be viewed as a raw audio representation
  - 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



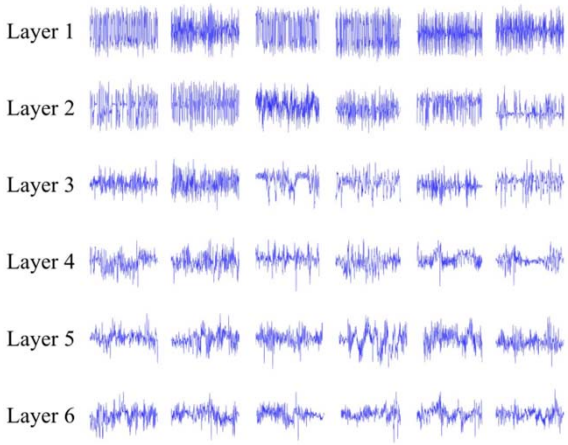
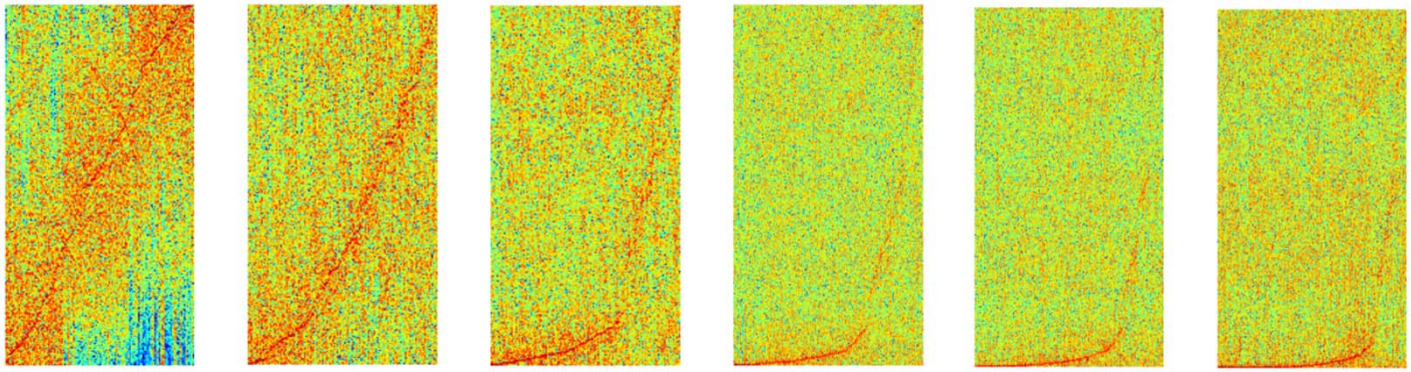


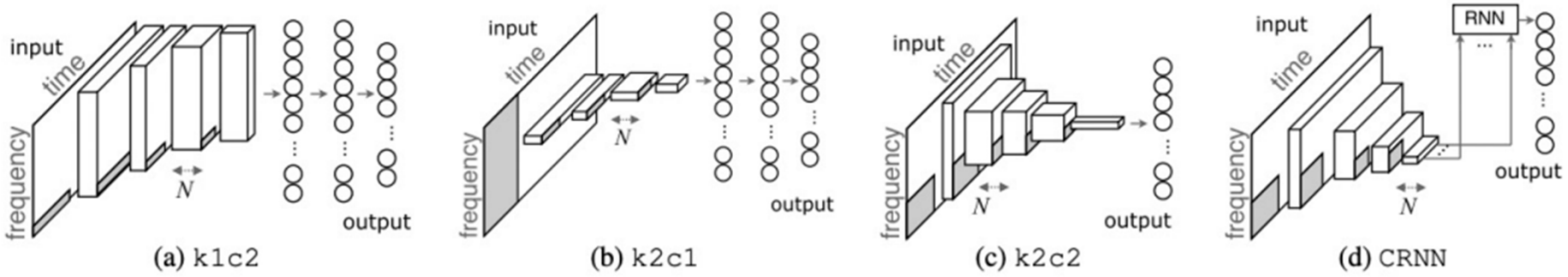
Figure 3. Examples of learned filters at each layer.



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/blob/master/training/model.py

```
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/blob/master/training/modules.py

```python
class Conv_2d(nn.Module):
    def __init__(self, input_channels, output_channels, shape=3, stride=1, pooling=2):
        super(Conv_2d, self).__init__()
        self.conv = nn.Conv2d(input_channels, output_channels, shape, stride=stride, padding=shape//2)
        self.bn = nn.BatchNorm2d(output_channels)
        self.relu = nn.ReLU()
        self.mp = nn.MaxPool2d(pooling)
    def forward(self, x):
        out = self.mp(self.relu(self.bn(self.conv(x))))
        return out
```

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

Short-Chunk CNN and Others

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.

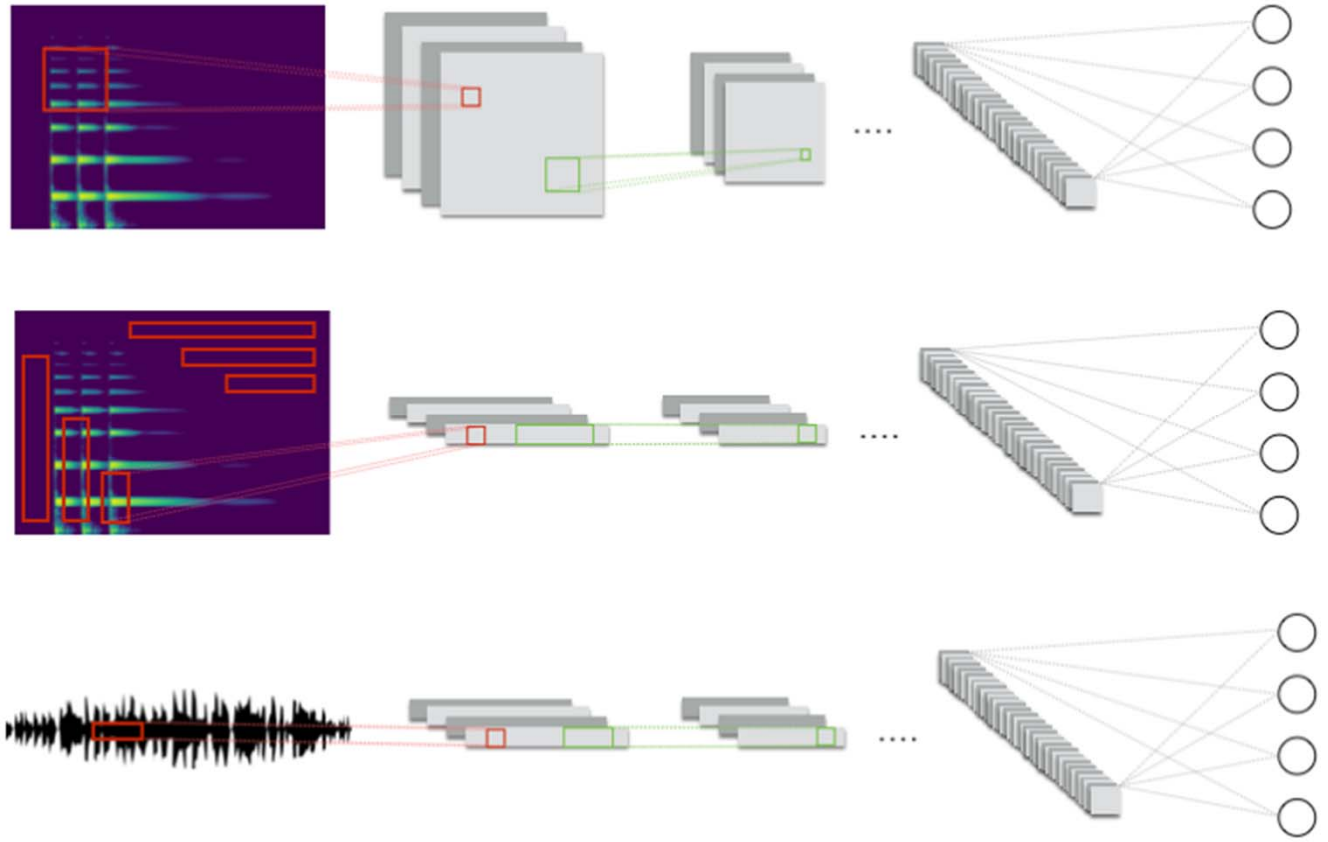
Exemplar Models

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

Model	Preprocessing	Input length	Front end	Back end	Training	Aggregation
FCN	STFT	29.1s	2D CNN	.	song-level	.
VGG-ish / Short-chunk CNN	STFT	3.96s	2D CNN	Global max pooling	instance-level	Average
Harmonic CNN	STFT	5s	2D CNN	Global max pooling	instance-level	Average
MusiCNN	STFT	3s	2D CNN	1D CNN	instance-level	Average
Sample-level CNN	.	3s	1D CNN	1D CNN	instance-level	Average
CRNN	STFT	29.1s	2D CNN	RNN	song-level	.
Music tagging transformer	STFT	5s-30s	2D CNN	Transformer	instance-level	Average

Exemplar Models

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



Audio Data Augmentations

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

Name	Author	Framework	Language	License	Link
Muda	B. McFee et al. (2015)	General Purpose	Python	ISC License	source code
Audio Degradation Toolbox	M. Mauch et al. (2013)	General Purpose	MATLAB	GNU General Public License 2.0	source code
rubberband	-	General Purpose	C++	GNU General Public License (non-commercial)	website, pyrubberband
torchaudio	pytorch.org	PyTorch	Python	BSD 2-Clause "Simplified" License	source code

Audio Degradation Toolbox

https://github.com/sevagh/audio-degradation-toolbox

- Exemplar use case: simulate the case of smartphone recording

```
{ "name": "noise", ["snr": 20, "color": "pink"] }
{ "name": "mp3", ["bitrate": 320] }
{ "name": "gain", ["volume": 10.0] }
{ "name": "normalize" }
{ "name": "low_pass", ["cutoff": 1000.0] }
{ "name": "high_pass", ["cutoff": 1000.0] }
{ "name": "trim_millis", ["amount": 100, "offset": 0] }
{ "name": "mix", "path": STRING, ["snr": 20.0] }
{ "name": "speedup", "speed": FLOAT }
{ "name": "resample", "rate": INT }
{ "name": "pitch_shift", "octaves": FLOAT }
{ "name": "dynamic_range_compression", ["threshold": -20.0, "ratio": 4.0, "attack": 5.0, "release": 50.0]
{ "name": "impulse_response", "path": STRING }
{ "name": "equalizer", "frequency": FLOAT, ["bandwidth": 1.0, "gain": -3.0] }
{ "name": "time_stretch", "factor": FLOAT }
{ "name": "delay", "n_samples": INT }
{ "name": "clipping", ["n_samples": 0, "percent_samples": 0.0] }
{ "name": "wow_flutter", ["intensity": 1.5, "frequency": 0.5, "upsampling_factor": 5.0 ] }
{ "name": "aliasing", ["dest_frequency": 8000.0] }
```

torchaudio_augmentations

https://pytorch.org/audio/stable/tutorials/audio_data_augmentation_tutorial.html

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

```
from torchaudio_augmentations import (
    RandomResizedCrop,
    RandomApply,
    PolarityInversion,
    Noise,
    Gain,
    HighLowPass,
    Delay,
    PitchShift,
    Reverb,
    Compose,
)
```

pyrubberband

https://github.com/bmcfee/pyrubberband

```
>>> import soundfile as sf
>>> import pyrubberband as pyrb
>>> y, sr = sf.read("myfile.wav")
>>> # Play back at double speed
>>> y_stretch = pyrb.time_stretch(y, sr, 2.0)
>>> # Play back two semi-tones higher
>>> y_shift = pyrb.pitch_shift(y, sr, 2)
```

- **Time stretch**: make it faster/slower without changing the pitch
- **Pitch shift**

- (ps. There is a cool function called “timemap_stretch”; check it out yourself)

Sample Code

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

```python
from torchaudio_augmentations import (
    RandomResizedCrop,
    RandomApply,
    PolarityInversion,
    Noise,
    Gain,
    HighLowPass,
    Delay,
    PitchShift,
    Reverb,
    Compose,
)
```

```python
def get_dataloader(data_path=None,
                   split='train',
                   num_samples=22050 * 29,
                   num_chunks=1,
                   batch_size=16,
                   num_workers=0,
                   is_augmentation=False):
    is_shuffle = True if (split == 'train') else False
    batch_size = batch_size if (split == 'train') else (batch_size // num_chunks)
    data_loader = data.DataLoader(dataset=GTZANDataset(data_path,
                                                       split,
                                                       num_samples,
                                                       num_chunks,
                                                       is_augmentation),
                                  batch_size=batch_size,
                                  shuffle=is_shuffle,
                                  drop_last=False,
                                  num_workers=num_workers)
    return data_loader
```

```python
def _get_augmentations(self):
    transforms = [
        RandomResizedCrop(n_samples=self.num_samples),
        RandomApply([PolarityInversion()], p=0.8),
        RandomApply([Noise(min_snr=0.3, max_snr=0.5)], p=0.3),
        RandomApply([Gain()], p=0.2),
        RandomApply([HighLowPass(sample_rate=22050)], p=0.8),
        RandomApply([Delay(sample_rate=22050)], p=0.5),
        RandomApply([PitchShift(n_samples=self.num_samples, sample_rate=22050)], p=0.4),
        RandomApply([Reverb(sample_rate=22050)], p=0.3),
    ]
    self.augmentation = Compose(transforms=transforms)
```

```python
train_loader = get_dataloader(split='train', is_augmentation=True)
iter_train_loader = iter(train_loader)
train_wav, train_genre = next(iter_train_loader)

valid_loader = get_dataloader(split='valid')
test_loader = get_dataloader(split='test')
```

Sample Code

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

```
for epoch in range(num_epochs):
    losses = []

    # Train
    cnn.train()
    for (wav, genre_index) in train_loader:
        wav = wav.to(device)
        genre_index = genre_index.to(device)

        # Forward
        out = cnn(wav)
        loss = loss_function(out, genre_index)

        # Backward
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        losses.append(loss.item())
    print('Epoch: [%d/%d], Train loss: %.4f' % (epoch+1, num_epochs, np.mean(losses)))
```

```
    # Validation
    cnn.eval()
    y_true = []
    y_pred = []
    losses = []
    for wav, genre_index in valid_loader:
        wav = wav.to(device)
        genre_index = genre_index.to(device)

        # reshape and aggregate chunk-level predictions
        b, c, t = wav.size()
        logits = cnn(wav.view(-1, t))
        logits = logits.view(b, c, -1).mean(dim=1)
        loss = loss_function(logits, genre_index)
        losses.append(loss.item())
        _, pred = torch.max(logits.data, 1)

        # append labels and predictions
        y_true.extend(genre_index.tolist())
        y_pred.extend(pred.tolist())
    accuracy = accuracy_score(y_true, y_pred)
    valid_loss = np.mean(losses)
    print('Epoch: [%d/%d], Valid loss: %.4f, Valid accuracy: %.4f' % (epoch+
```

```
import seaborn as sns
from sklearn.metrics import confusion_matrix


accuracy = accuracy_score(y_true, y_pred)
cm = confusion_matrix(y_true, y_pred)
sns.heatmap(cm, annot=True, xticklabels=GTZAN_GENRES, yticklabels=GTZAN_GENRES, cmap='YlGnBu')
print('Accuracy: %.4f' % accuracy)
```

Exemplar Model: PANNs

https://github.com/qiuqiangkong/audioset_tagging_cnn

- From sound event detection
- Purely convolutional
- Can be used as a pre-trained model
  - Produce audio embeddings that have been used by **CLAP** (https://github.com/LAION-AI/CLAP) in learning audio-text joint embedding space for **text-to-audio** generation

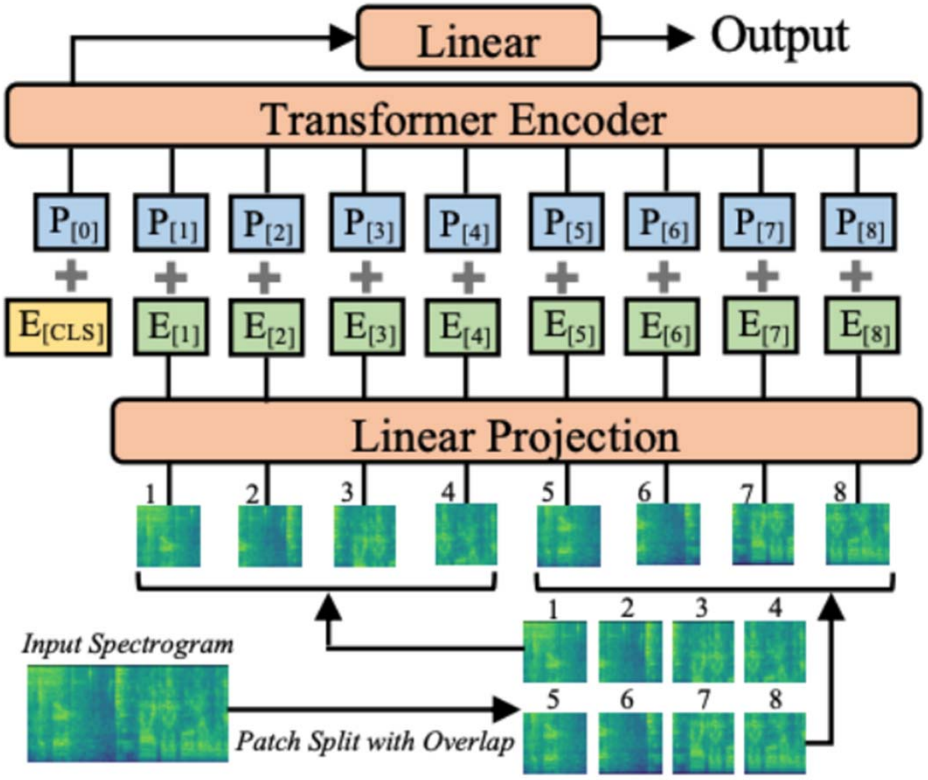
VGGish [1]	CNN6	CNN10	CNN14
Log-mel spectrogram 96 frames × 64 mel bins	Log-mel spectrogram 1000 frames × 64 mel bins		
3 × 3 @ 64 ReLU	5 × 5 @ 64 BN, ReLU	(3 × 3 @ 64 BN, ReLU) × 2	(3 × 3 @ 64 BN, ReLU) × 2
MP 2 × 2	Pooling 2 × 2		
3 × 3 @ 128 ReLU	5 × 5 @ 128 BN, ReLU	(3 × 3 @ 128 BN, ReLU) × 2	(3 × 3 @ 128 BN, ReLU) × 2
MP 2 × 2	Pooling 2 × 2		
(3 × 3 @ 256 ReLU) × 2	5 × 5 @ 256 BN, ReLU	(3 × 3 @ 256 BN, ReLU) × 2	(3 × 3 @ 256 BN, ReLU) × 2
MP 2 × 2	Pooling 2 × 2		
(3 × 3 @ 512 ReLU) × 2	5 × 5 @ 512 BN, ReLU	(3 × 3 @ 512 BN, ReLU) × 2	(3 × 3 @ 512 BN, ReLU) × 2
MP 2 × 2 Flatten	Global pooling		Pooling 2 × 2
FC 4096 ReLU × 2	FC 512, ReLU		(3 × 3 @ 1024 BN, ReLU) × 2
FC 527, Sigmoid	FC 527, Sigmoid		Pooling 2 × 2
			(3 × 3 @ 2048 BN, ReLU) × 2
			Global pooling
			FC 2048, ReLU
			FC 527, Sigmoid

Ref: Kong et al., "PANNs: Large-scale pretrained audio neural networks for audio pattern recognition," arXiv 2020

Exemplar Model: Audio Spectrogram Transformer

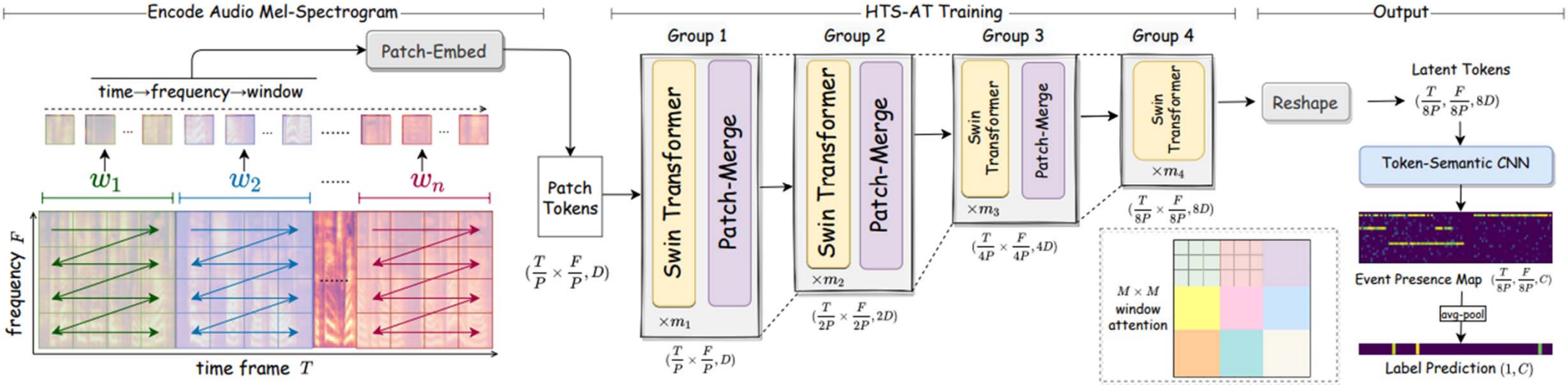
https://github.com/YuanGongND/ast

- From sound event detection
- Use Vison **Transformer** (ViT) based architecture
  - The first convolution-free, purely attention-based model for audio classification
- May need larger amount of training data and compute



Ref: Gong et al., "AST: Audio Spectrogram Transformer," INTERSPEECH 2021

Exemplar Model: HTS-AT



Ref: Chen et al., "HTS-AT: A hierarchical token-semantic audio transformer for sound classification and detection," ICASSP 2022

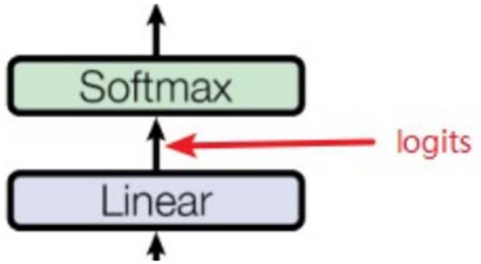
Ref: Wu et al., "Large-scale contrastive language-audio pretraining with feature fusion and keyword-to-caption augmentation," arXiv 2022

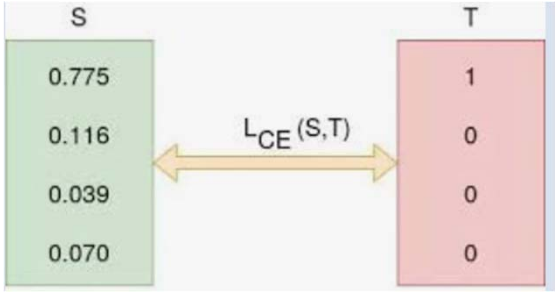
Model	AudioCaps (mAP@10)		Clotho (mAP@10)	
	A→T	T→A	A→T	T→A
PANN+CLIP Trans.	4.7	11.7	1.9	4.4
PANN+BERT	34.3	44.3	10.8	17.7
PANN+RoBERTa	37.5	45.3	11.3	18.4
HTSAT+CLIP Trans.	2.4	6.0	1.1	3.2
HTSAT+BERT	43.7	49.2	**13.8**	**20.8**
HTSAT+RoBERTa	**45.7**	**51.3**	**13.8**	20.4

Table 2: The text-to-audio retrieval result (mAP@10) of using different audio/text encoder on AudioCaps and Clotho.

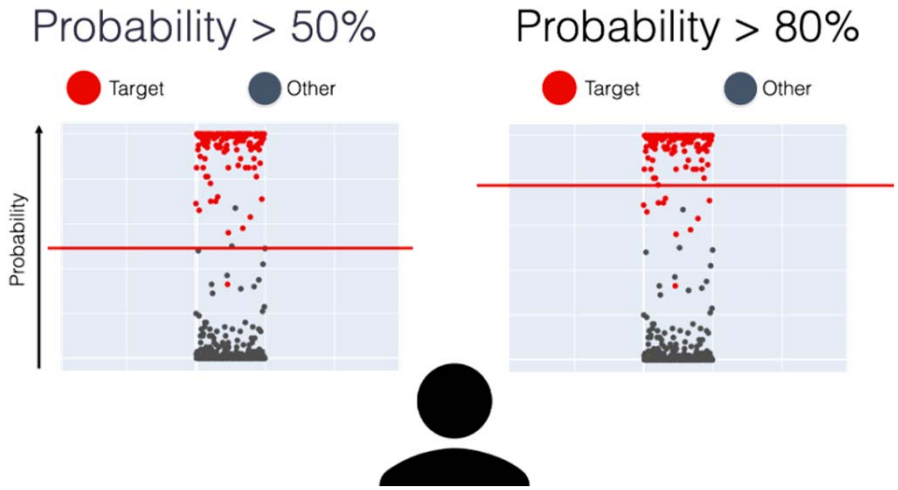
Evaluation Metrics for Music Classification

- The output of DL-based classifiers are usually probabilities
  - multi-class classification: **softmax**
  - multi-label classification: **sigmoid**





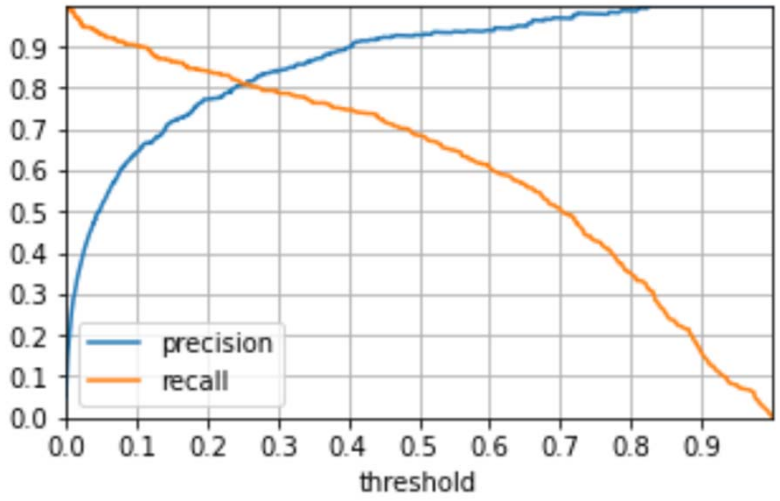
- Probability vs decision
  - outputting probabilities is fine at training time
  - but, at inference time, need to "**make decisions**"
  - usually by thresholding (e.g., at 0.5)



Evaluation Metrics for Music Classification

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

- Classification accuracy (top1, top3)
- Precision, recall, Fscore
  - obtained by varying the threshold

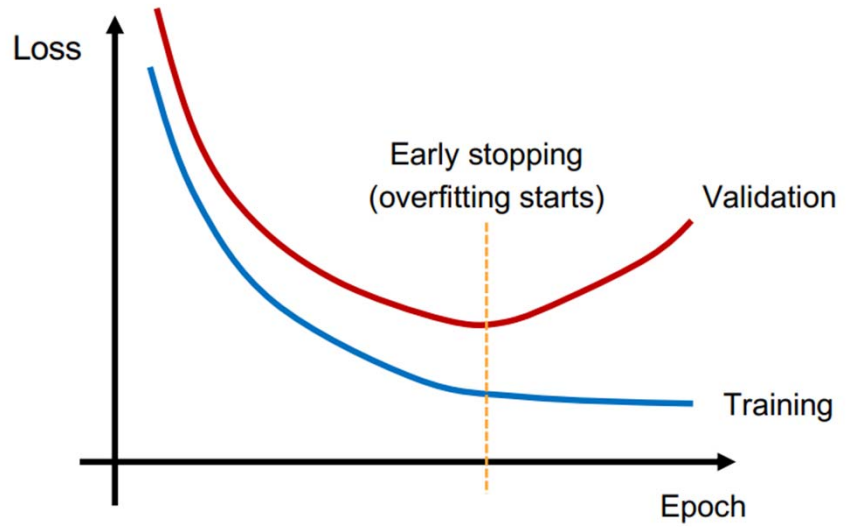


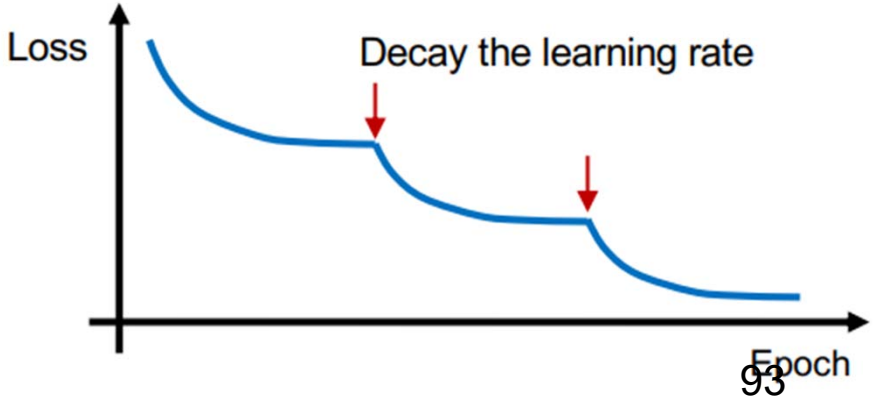
- ROC-AUC
  - "micro" vs "macro" average

average : {'micro', 'macro', 'samples', 'weighted'}

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
  - Then try to mitigate overfitting
- Try different learning rates and optimizers





Outline

- Music classification: Basics
- ML-based music classification (and hand-crafted audio features)
- DL-based music classification
- **Music foundation models**

Music Foundation Models

https://github.com/nicolaus625/FM4Music

- "The term foundation model was coined to describe a multipurpose machine learning model that, **rather than being trained for a single specific task**, serves as the **basis** of multiple derived models that are able to perform a wide range of tasks"
- "Following this **pre-training** phase, foundation models can be **adapted** for various **downstream** tasks via a relatively **lightweight finetuning** or **in-context learning** stage, for example, using a labelled dataset that is orders of magnitude smaller than the pre-training data."
- "FMs for music not only address data scarcity and reduce annotation costs, but also enhance **generalisation** in music information retrieval and creation."

Ref: Ma et al., "Foundation Models for Music: A Survey," arXiv 2024

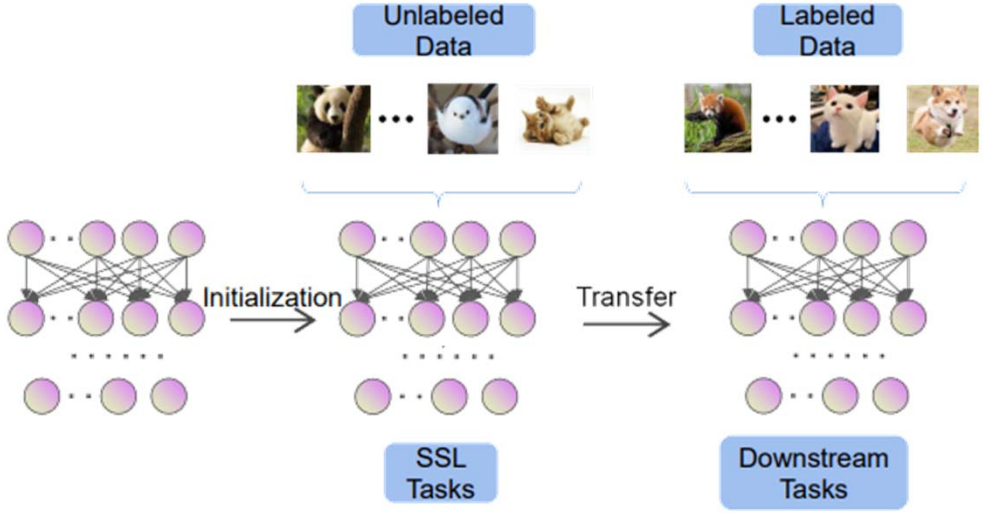
Pre-Training Strategies

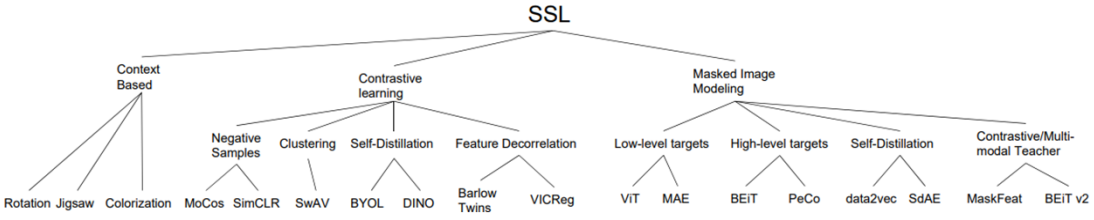
- "Foundation models are pre-trained in a **self-supervised** fashion on **large-scale** datasets, **avoiding or minimising the need for labelled data**"
- Mainstream strategies
  - Contrastive learning and clustering (e.g., SimCLR)
  - Generative pre-training (e.g., AE or VQVAE)
  - Masked modelling (e.g., BERT)

Self-Supervised Learning (SSL)

https://music-classification.github.io/tutorial/part5_beyond/introduction.html

- **Do not** need human labels (**unlabeled**)
- Can learn from **large** amount of data
- Can do "**transfer learning**"
  - ***Pre-train*** the model with SSL
  - ***Fine-tune*** with a few more layers on downstream tasks using supervised learning



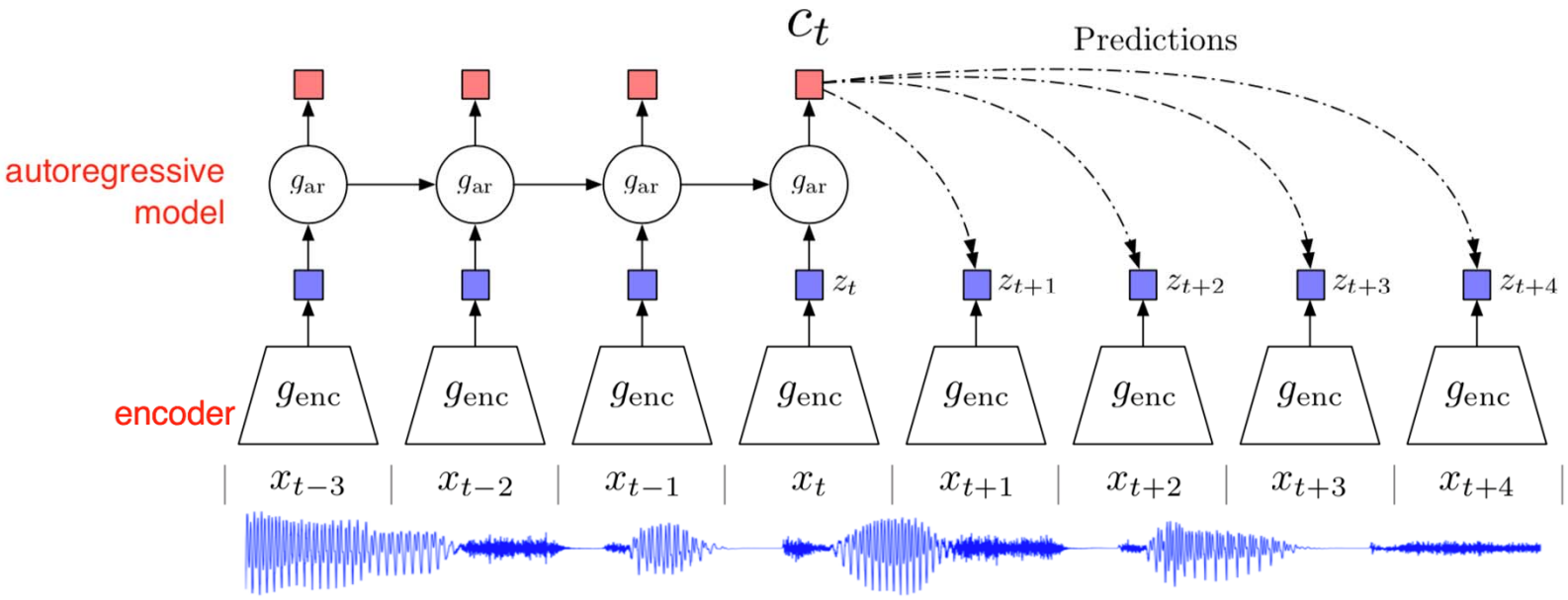


Gui et al., "A survey on self-supervised learning: Algorithms, applications, and future trends," TPAMI 2024

Contrastive Predictive Coding

https://music-classification.github.io/tutorial/part5_beyond/methods.html

- Learn by predicting the future in a learned latent space

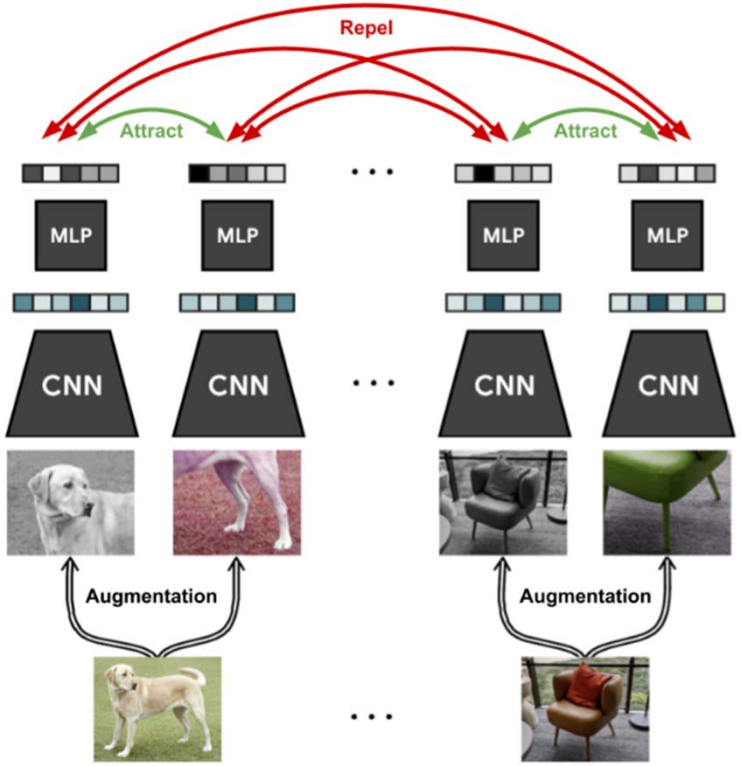


Ref: van den Oord et al., "Representation learning with contrastive predictive coding," arXiv 2018

SimCLR

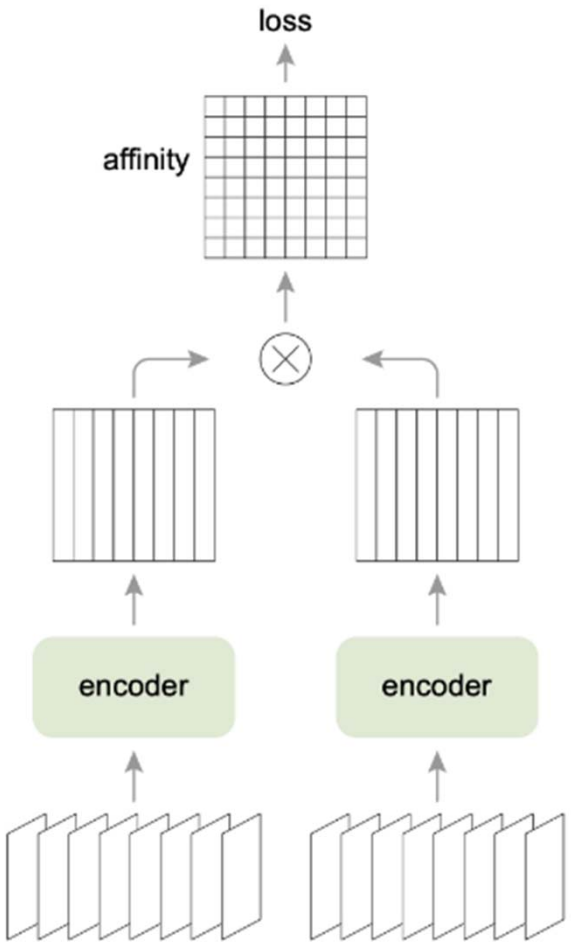
https://music-classification.github.io/tutorial/part5_beyond/methods.html

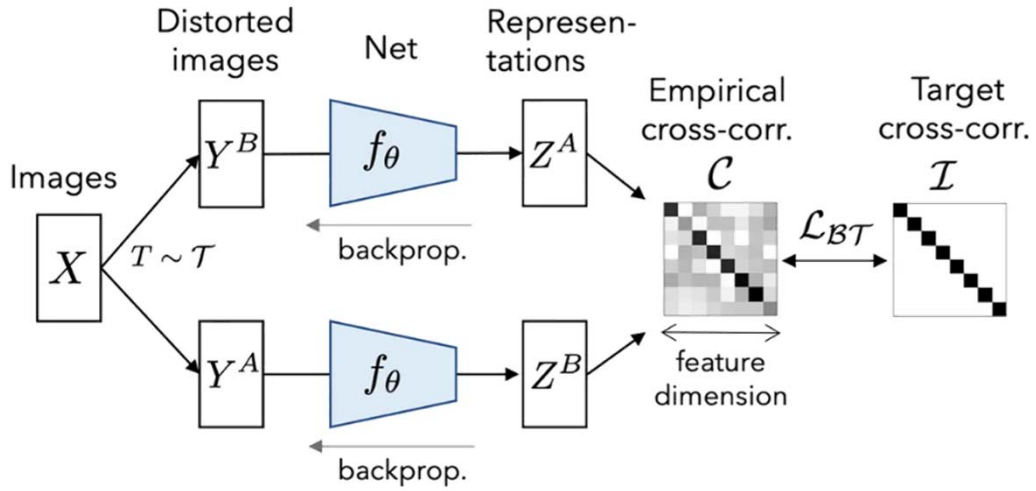
- "Draw closer" positive data pairs while "push away" negative data pairs
  - **Positive pairs**: different "augmented views" of the same instance (done by *data augmentation*)
  - **Negative pairs**: different instances
- **Metric learning**
  - shared encoder (CNN+MLP), output an *embedding vector*
  - learn a similarity metric discriminatively



Ref: Chen et al., "A simple framework for contrastive learning of visual representations," ICML 2020

SimCLR



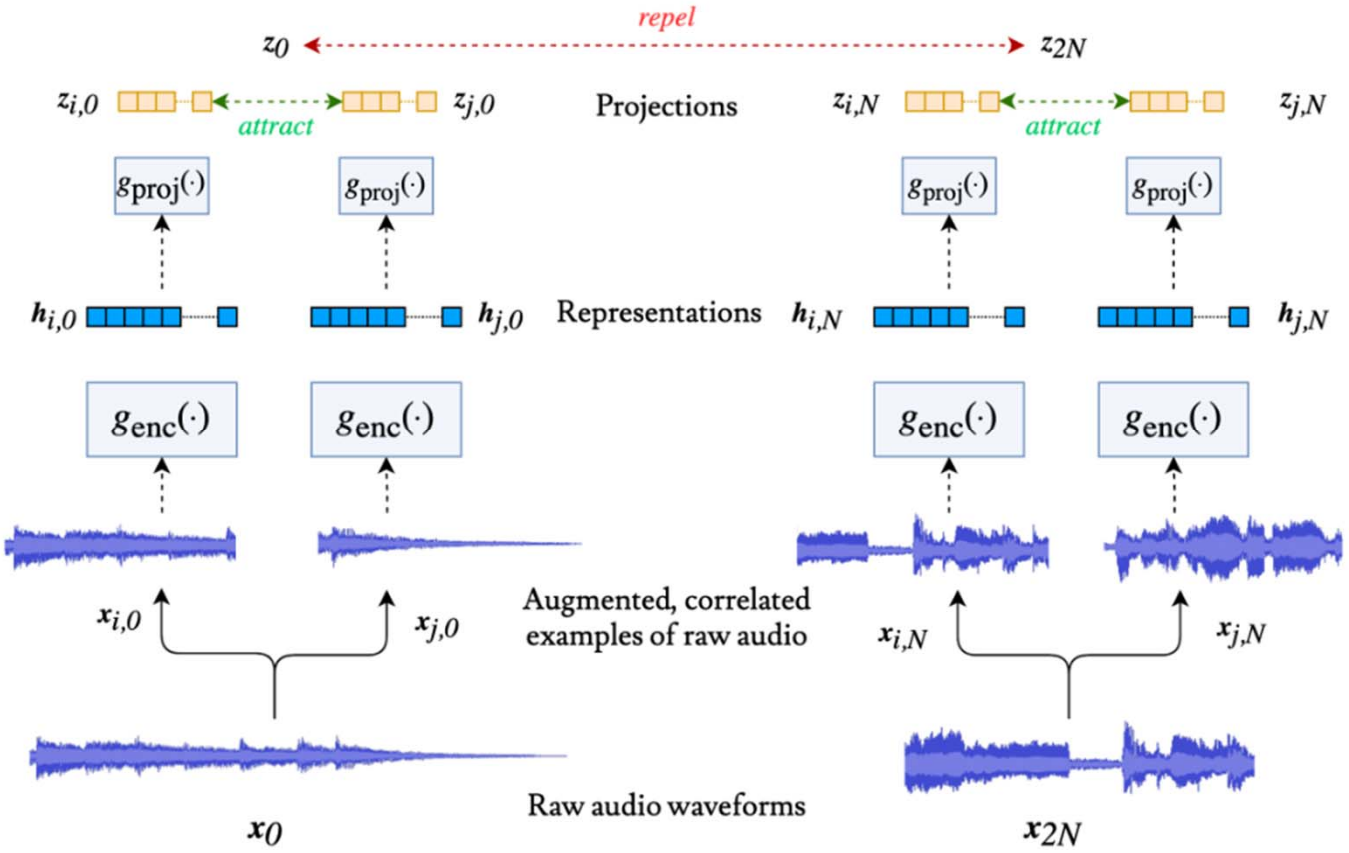


- Training details
  - given a mini-batch of *N* samples, we will have *2N* samples after data augmentation
  - given a positive pair, view the other 2(*N*-1) pairs as negative
  - infoNCE loss

$$\ell_{i,j} = -\log \frac{\exp(\text{sim}(\boldsymbol{z}_i, \boldsymbol{z}_j)/\tau)}{\sum_{k=1}^{2N} \mathbb{1}_{[k \neq i]} \exp(\text{sim}(\boldsymbol{z}_i, \boldsymbol{z}_k)/\tau)}$$

Exemplar Model: CLMR

https://music-classification.github.io/tutorial/part5_beyond/self-supervised-learning.html

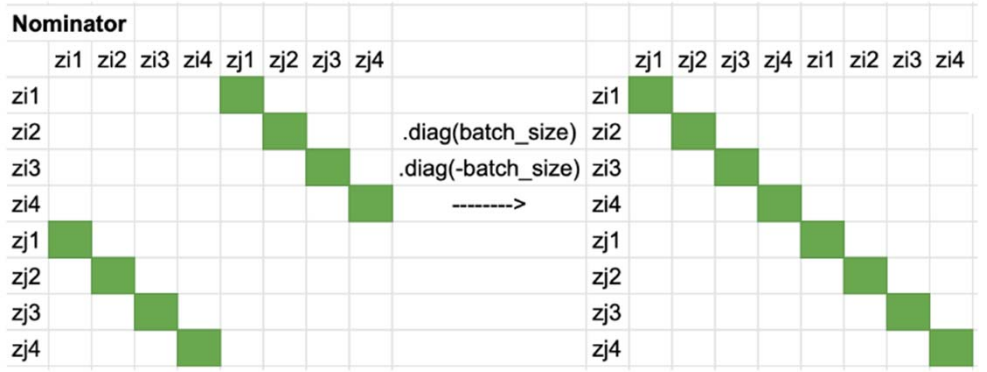


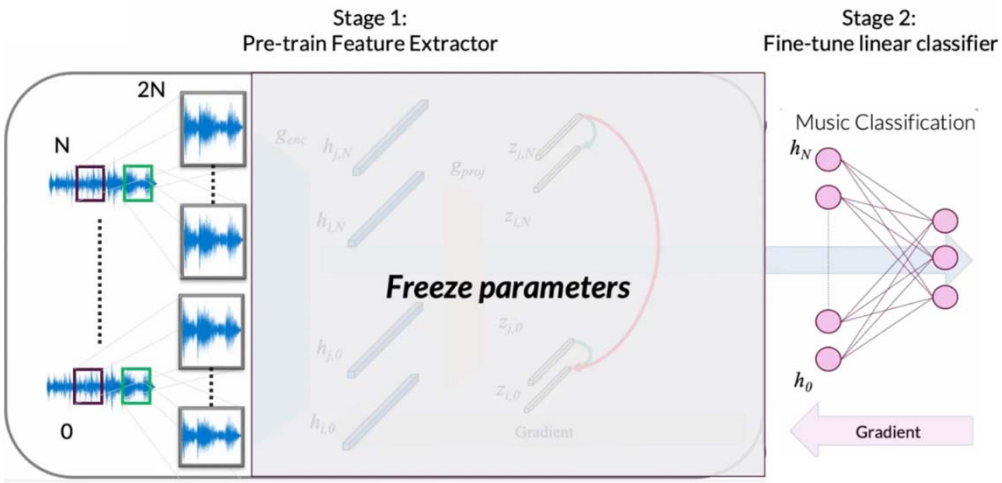
Ref: Spijkervet & Burgoyne, "Contrastive Learning of Musical Representations," ISMIR 2021

Sample Code

https://music-classification.github.io/tutorial/part5_beyond/self-supervised-learning.html

- Use **sample-level** CNN as the feature extractor
  - 59,049 (3^10) samples at 22kHz (~2.68 seconds)
- Use SimCLR for learning
- Freeze the pre-trained feature extractor, **fine-tune *linear classifier***
  - supervised SampleCNN: 49.6%
  - SSL CLMR + SampleCNN: 55.2%





Sample Code

https://music-classification.github.io/tutorial/part5_beyond/self-supervised-learning.html

```python
def train_linear_model(encoder, linear_model, epochs, learning_rate):

    # we now use a regular CrossEntropy loss to compare our predicted genre labels with the ground truth labe
    criterion = nn.CrossEntropyLoss()

    losses = []
    for e in range(epochs):
        epoch_losses = []
        for (x, _), y in tqdm(train_loader):

            optimizer.zero_grad()

            # we will not be backpropagating the gradients of the SampleCNN encoder:
            with torch.no_grad():
                h = encoder(x)

            p = linear_model(h)

            loss = criterion(p, y)

            loss.backward()
            optimizer.step()

            # print(f"Loss: {loss}")
            epoch_losses.append(loss.detach().item())

        mean_loss = np.array(epoch_losses).mean()
        losses.append(mean_loss)
        print(f"Epoch: {e}\tMean loss: {mean_loss}")
    return losses
```

```python
class LinearModel(nn.Module):
    def __init__(self, hidden_dim, output_dim):
        super().__init__()
        self.hidden_dim = hidden_dim
        self.output_dim = output_dim
        self.model = nn.Linear(self.hidden_dim, self.output_dim)

    def forward(self, x):
        return self.model(x)
```

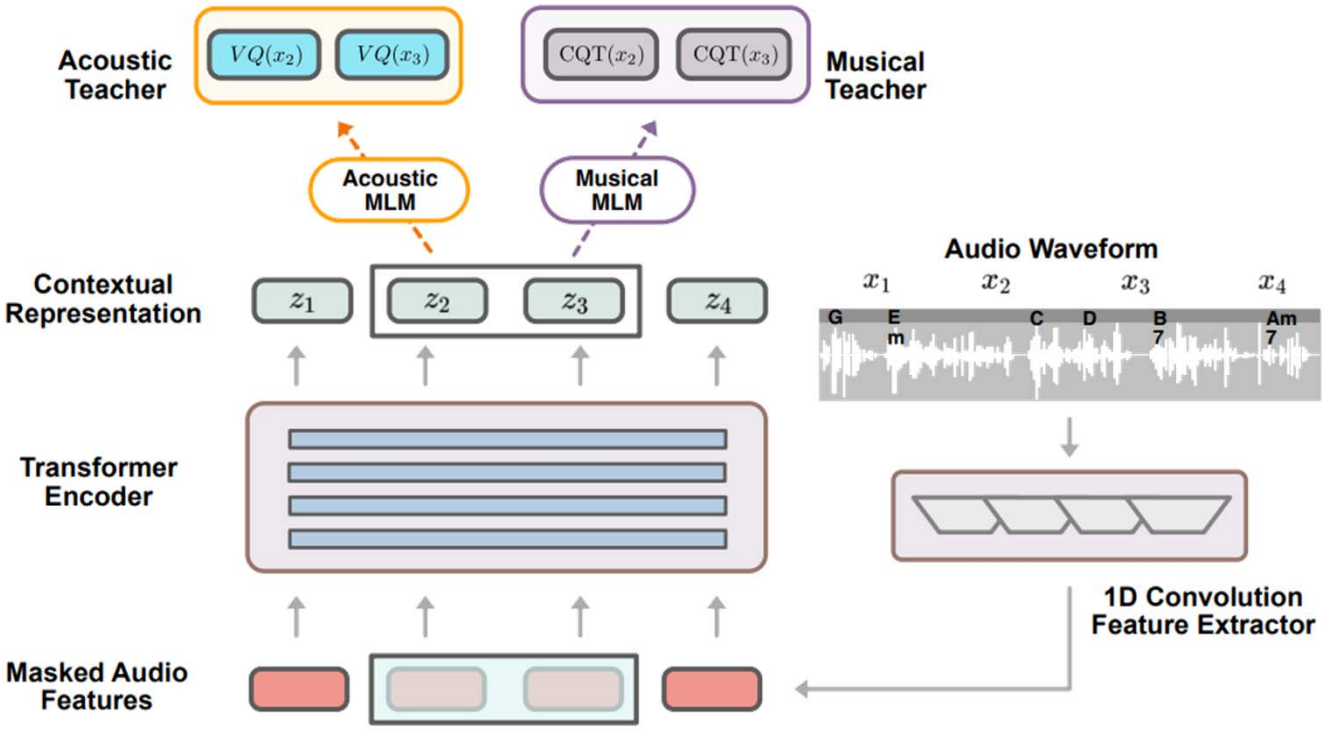
```python
linear_model = LinearModel(n_features, args.n_classes)
```

Exemplar Model: MERT

https://github.com/yizhilll/MERT

- Use **sample-level** CNNs
  - **5-second** clips as input
  - 75 embeddings per second
- Masked language modeling
  - acoustic teachers ($\mathcal{L}_H$)
    - predict HuBERT or Encodec
  - musical teachers ($\mathcal{L}_{CQT}$)
    - predict CQT
  - do both

$$\mathcal{L} = \alpha \cdot \mathcal{L}_H + \mathcal{L}_{CQT}$$



Ref: Li et al., "MERT: Acoustic music understanding model with large-scale self-supervised training," ICLR 2024

MARBLE Benchmark

Music Audio Representation Benchmark for universaL Evaluation

https://marble-bm.shef.ac.uk/

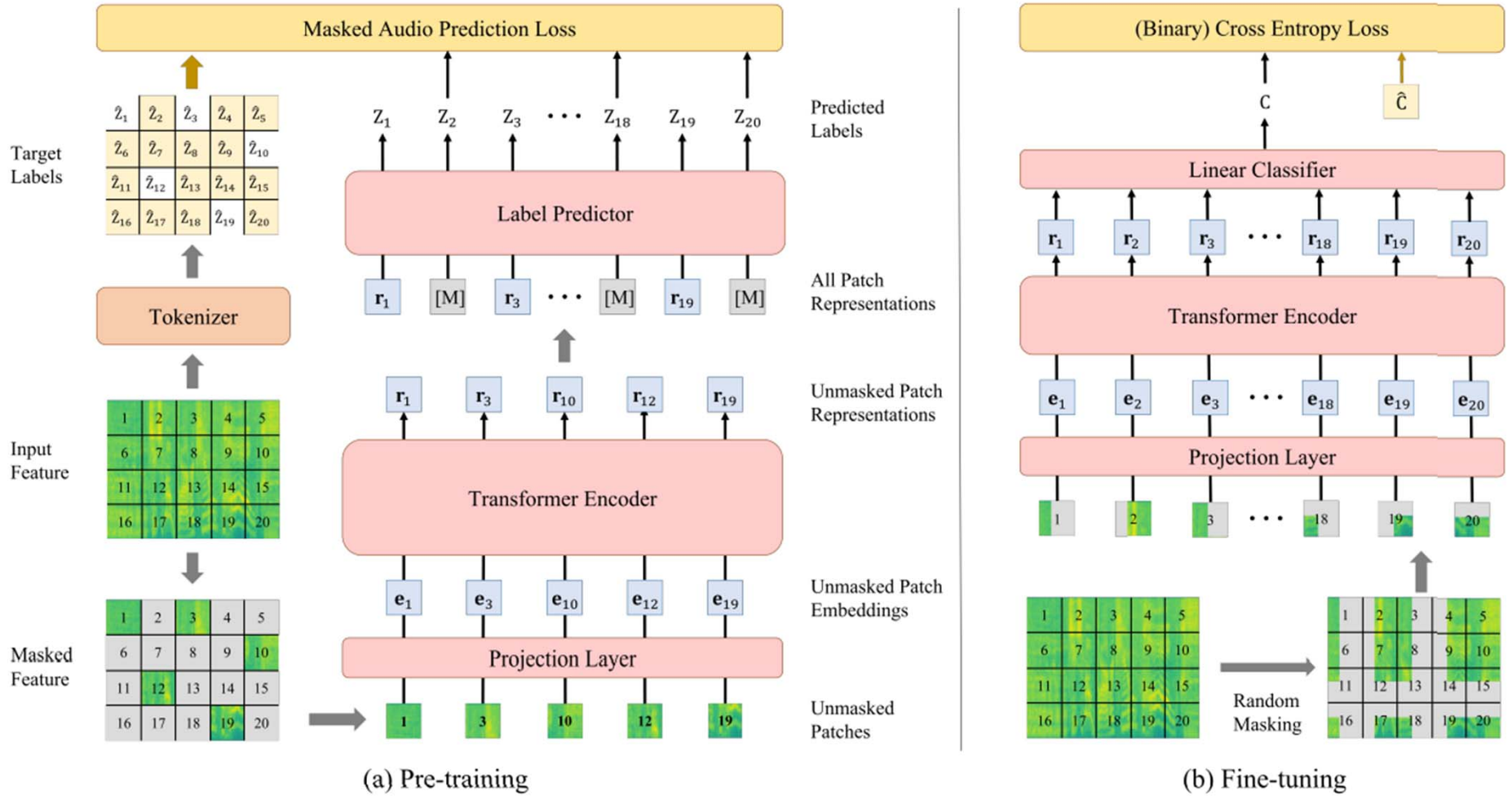
Dataset	MTT		GS	GTZAN	GTZAN	EMO		Nsynth	NSynth	VocalSet	VocalSet
Task	Tagging		Key	Genre	Rhythm	Emotion		Instrument	Pitch	Tech	Singer
Metric	ROC	AP	$Acc^{Refined}$	Acc	$F1^{beat}$	$R2^{V}$	$R2^{A}$	Acc	Acc	Acc	Acc
MAP-MERT-v0-95M	90.7	38.2	64.1	74.8	**88.3**	52.9	69.9	70.4	92.3	73.6	77.0
MAP-MERT-v0-95M-public	90.7	38.4	**67.3**	72.8	88.1	59.1	72.8	70.4	92.3	75.6	78.0
MAP-MERT-v1-95M	91.0	39.3	63.5	74.8	**88.3**	55.5	**76.3**	70.7	92.6	74.2	83.7
MAP-MERT-v1-330M	91.1	39.5	61.7	77.6	87.9	59.0	75.8	72.6	**94.4**	**76.9**	87.1
MAP-Music2Vec	90.0	36.2	50.6	74.1	68.2	52.1	71.0	69.3	93.1	71.1	81.4
MusiCNN	90.3	37.8	14.4	73.5	-	44.0	68.8	72.6	64.1	70.3	57.0
CLMR	89.5	36.0	14.8	65.2	-	44.4	70.3	67.9	47.0	58.1	49.9
Jukebox-5B	**91.4**	**40.6**	63.8	**77.9**	-	57.0	73.0	70.4	91.6	76.7	82.6
MULE	91.2	40.1	64.9	75.5	-	**60.7**	73.1	**74.6**	88.5	75.5	**87.5**

MERT Outperforms CLMR

https://marble-bm.shef.ac.uk/

Dataset	MTT		GS	GTZAN	GTZAN	EMO		Nsynth	NSynth	VocalSet	VocalSet
Task	Tagging		Key	Genre	Rhythm	Emotion		Instrument	Pitch	Tech	Singer
Metric	ROC	AP	$Acc^{Refined}$	Acc	$F1^{beat}$	$R2^{V}$	$R2^{A}$	Acc	Acc	Acc	Acc
MAP-MERT-v0-95M	90.7	38.2	64.1	74.8	**88.3**	52.9	69.9	70.4	92.3	73.6	77.0
MAP-MERT-v0-95M-public	90.7	38.4	**67.3**	72.8	88.1	59.1	72.8	70.4	92.3	75.6	78.0
MAP-MERT-v1-95M	91.0	39.3	63.5	74.8	**88.3**	55.5	**76.3**	70.7	92.6	74.2	83.7
MAP-MERT-v1-330M	91.1	39.5	61.7	77.6	87.9	59.0	75.8	72.6	**94.4**	**76.9**	87.1
MAP-Music2Vec	90.0	36.2	50.6	74.1	68.2	52.1	71.0	69.3	93.1	71.1	81.4
MusiCNN	90.3	37.8	14.4	73.5	-	44.0	68.8	72.6	64.1	70.3	57.0
CLMR	89.5	36.0	14.8	65.2	-	44.4	70.3	67.9	47.0	58.1	49.9
Jukebox-5B	**91.4**	**40.6**	63.8	**77.9**	-	57.0	73.0	70.4	91.6	76.7	82.6
MULE	91.2	40.1	64.9	75.5	-	**60.7**	73.1	**74.6**	88.5	75.5	**87.5**

BEATS



Ref: Chen et al., "BEATs: Audio Pre-Training with Acoustic Tokenizers," ICML 2023

Copyright-free Music Audio Datasets

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

- Free Music Archive (FMA)
  - https://freemusicarchive.org/
  - https://github.com/mdeff/fma
- Jamendo dataset
  - https://www.jamendo.com/
  - https://github.com/MTG/mtg-jamendo-dataset
- FreeSound
  - https://freesound.org/
  - https://labs.freesound.org/datasets/