2023 December 5

Deep Learning for Music Analysis and Generation

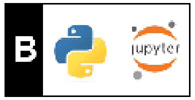
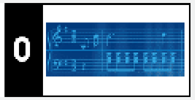
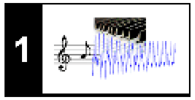
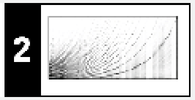
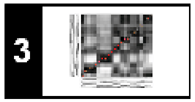
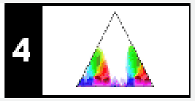
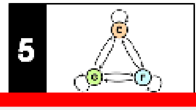
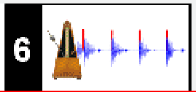
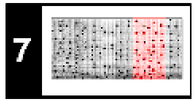
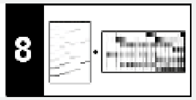
Rhythm

Beat tracking, downbeat tracking & tempo estimation (audio → score)

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

FMP Notebook

https://www.audiolabs-erlangen.de/resources/MIR/FMP/C6/C6.html

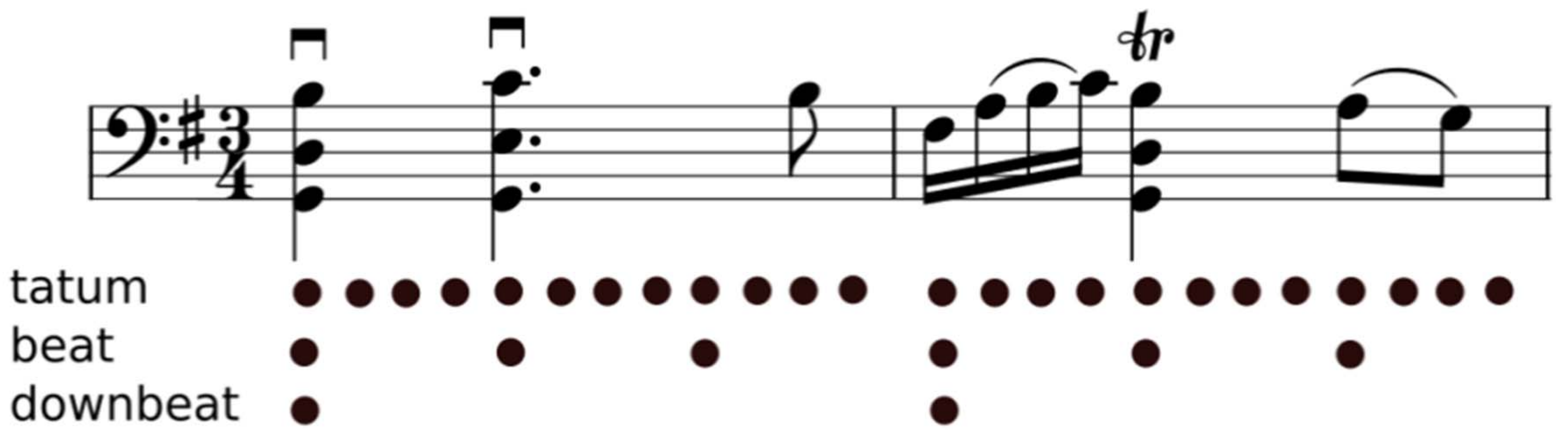
Part	Title	Notions, Techniques & Algorithms	HTML	IPYNB
B 	Basics	Basic information on Python, Jupyter notebooks, Anaconda package management system, Python environments, visualizations, and other topics	[html]	[ipynb]
0 	Overview	Overview of the notebooks (https://www.audiolabs-erlangen.de/FMP)	[html]	[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]
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]

ISMIR 2021 Tutorial

https://tempobeatdownbeat.github.io/tutorial/intro.html

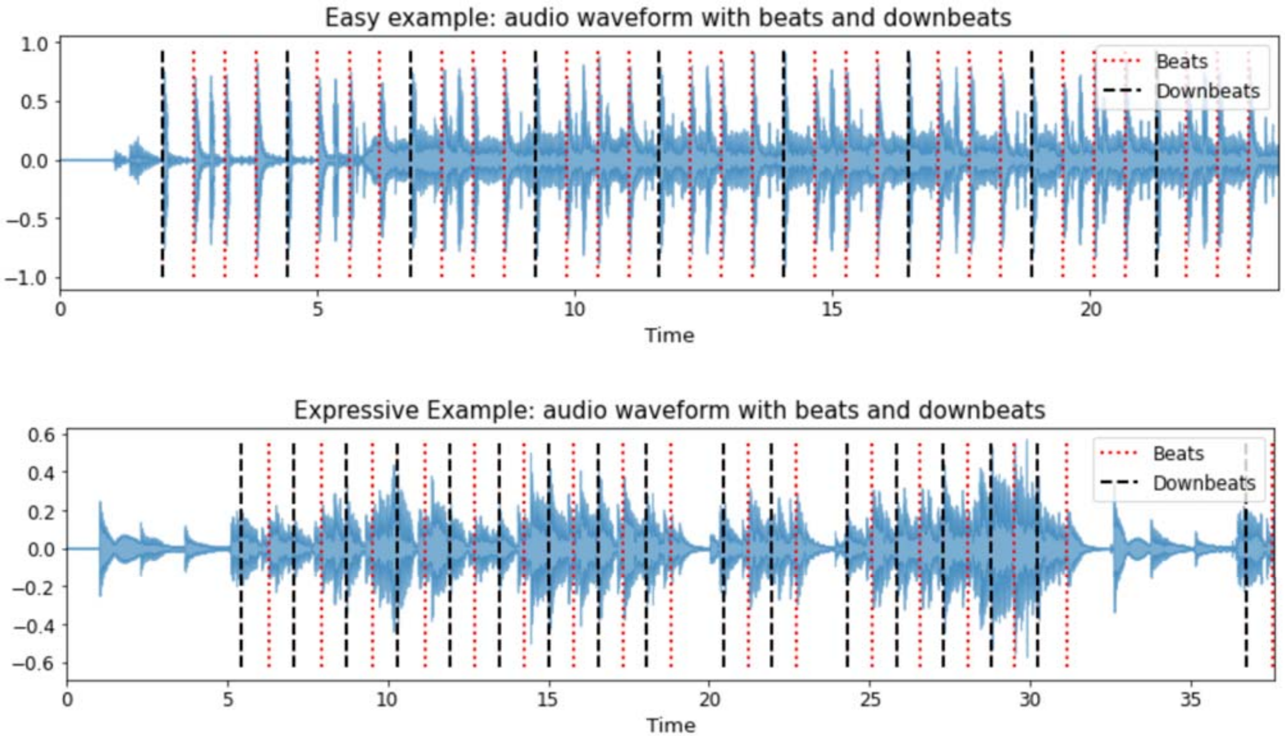
Tempo, Beat, and Downbeat Estimation

By Matthew E. P. Davies, Sebastian Böck, Magdalena Fuentes



What is Beat/Downbeat Tracking?

https://tempobeatdownbeat.github.io/tutorial/ch2_basics/definition.html



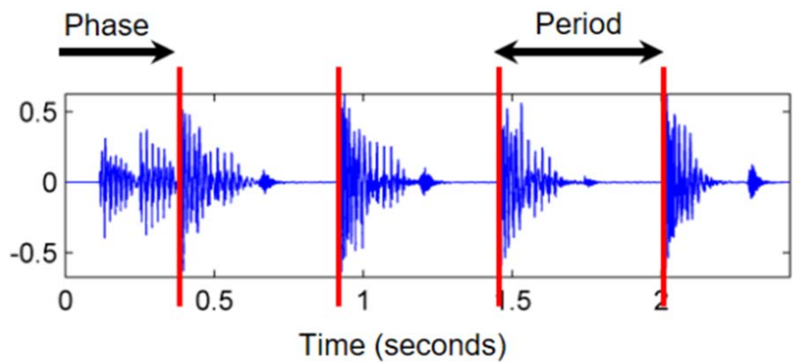


Figure 6.1 from [Müller, FMP, Springer 2015]

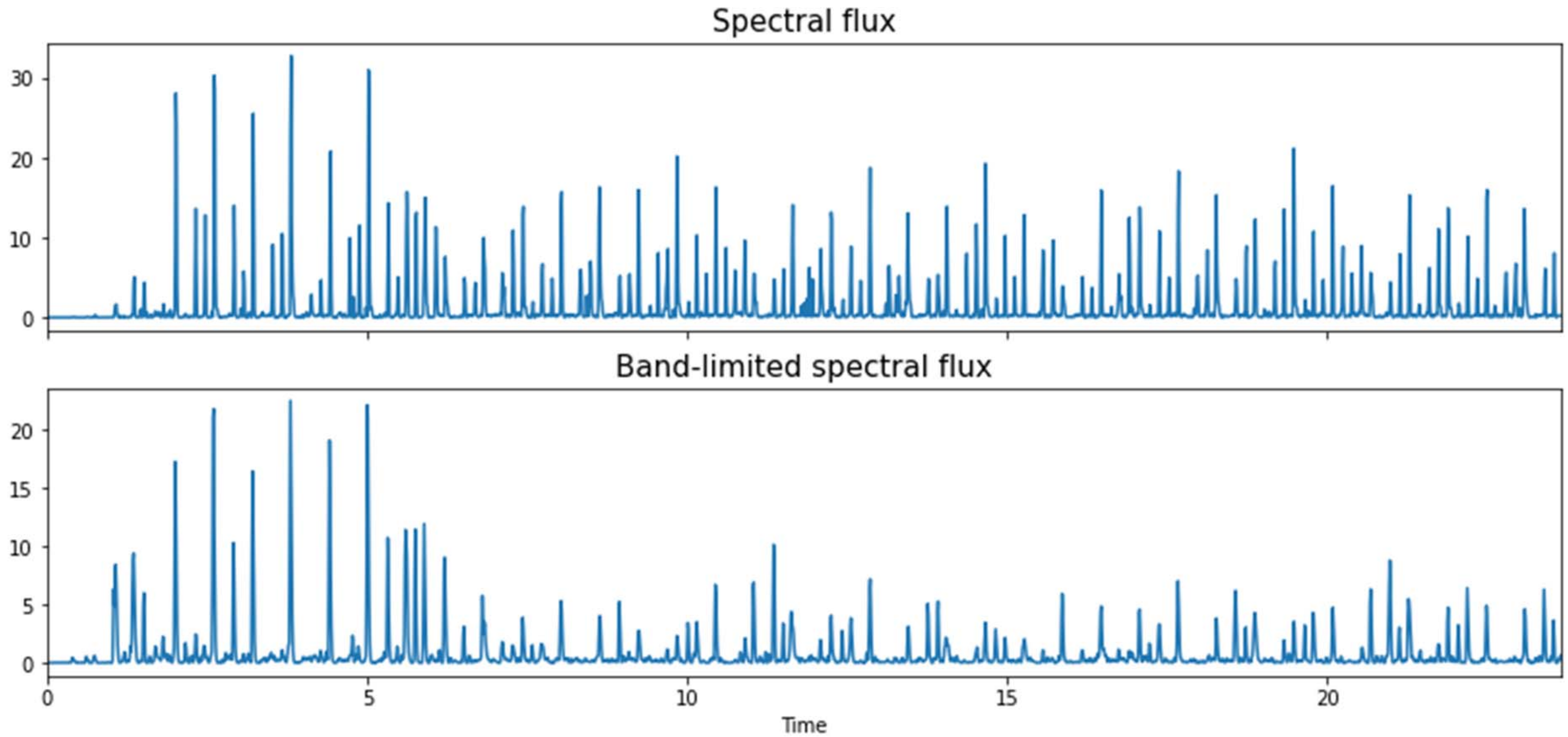
Beat/Downbeat Annotation

https://tempobeatdownbeat.github.io/tutorial/ch2_basics/annotation.html

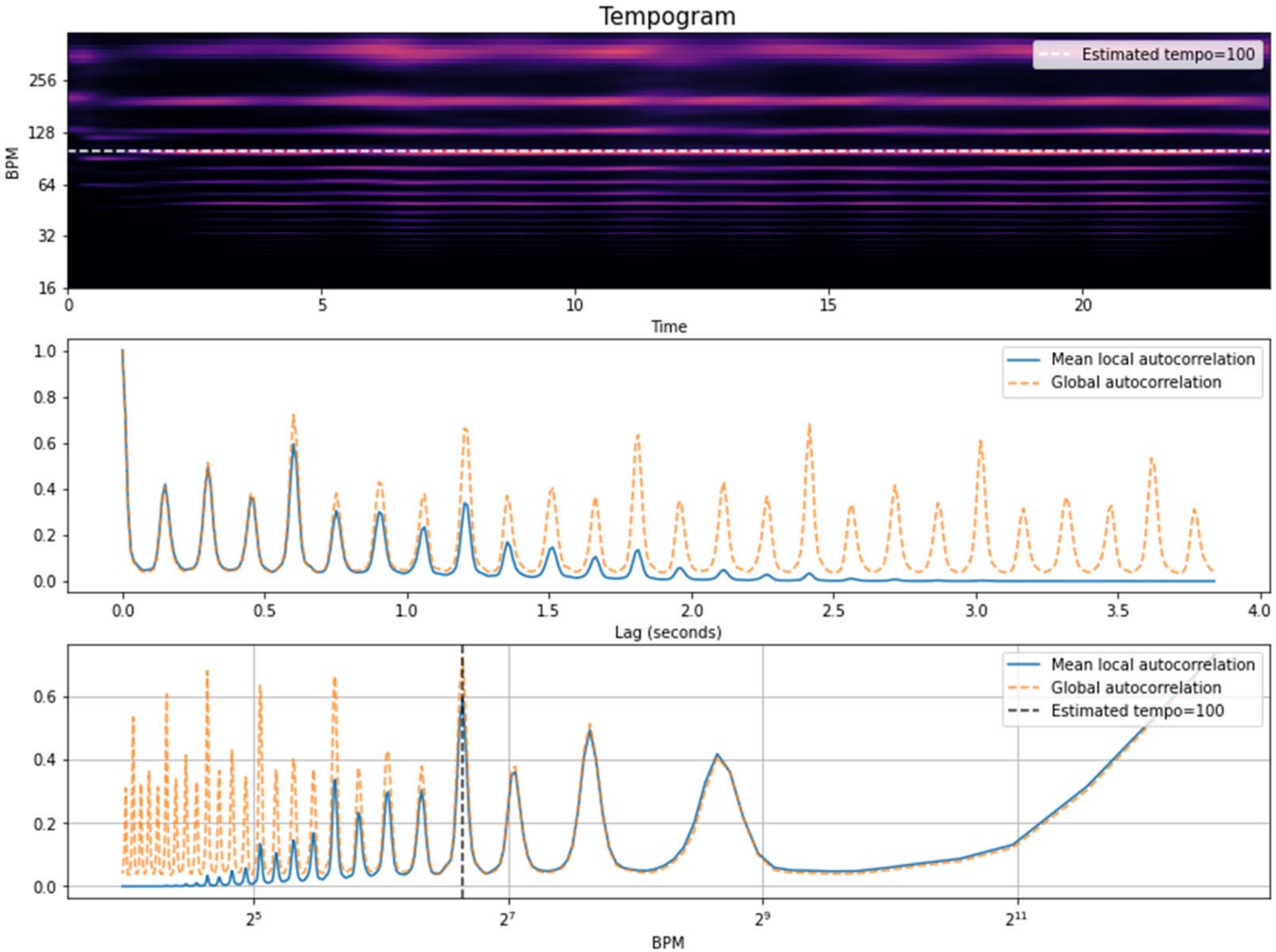
- Annotation is hard!
- It takes a long time, and the more challenging the material to annotate the greater the likelihood of this being helpful for learning.
- On the plus side, annotation is a fantastic way to learn about the task of beat and downbeat estimation so it's a really great excercise.
- We always need more data, so do consider doing some annotating!
- As hard as we try, annotation "mistakes" are made, so they made need correcting.
- This makes comparative evaluation more challenging, so it's always worthwile to ensure you are using the most up to date version of any annotations.

Baseline Approach

https://tempobeatdownbeat.github.io/tutorial/ch2_basics/baseline.html



Baseline Approach

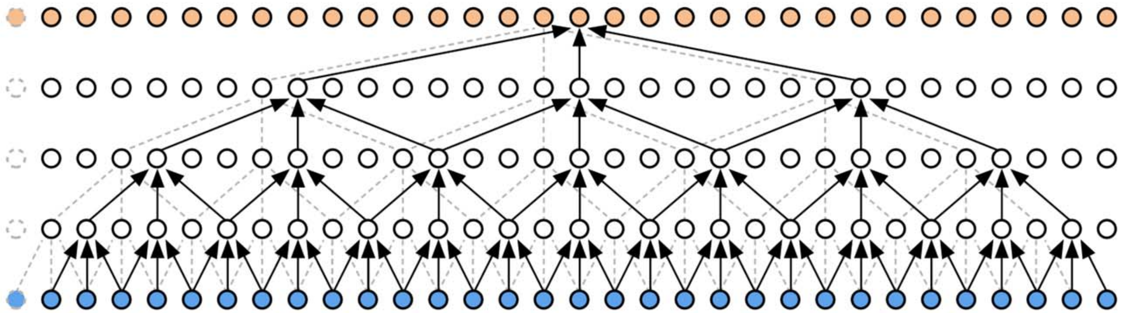


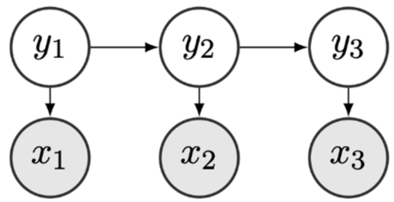
Deep Learning Approaches

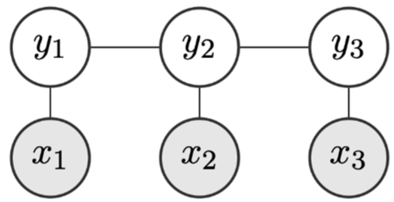
https://tempobeatdownbeat.github.io/tutorial/ch3_going_deep/overview.html

https://tempobeatdownbeat.github.io/tutorial/ch3_going_deep/dnns.html

https://tempobeatdownbeat.github.io/tutorial/ch3_going_deep/postprocessing.html







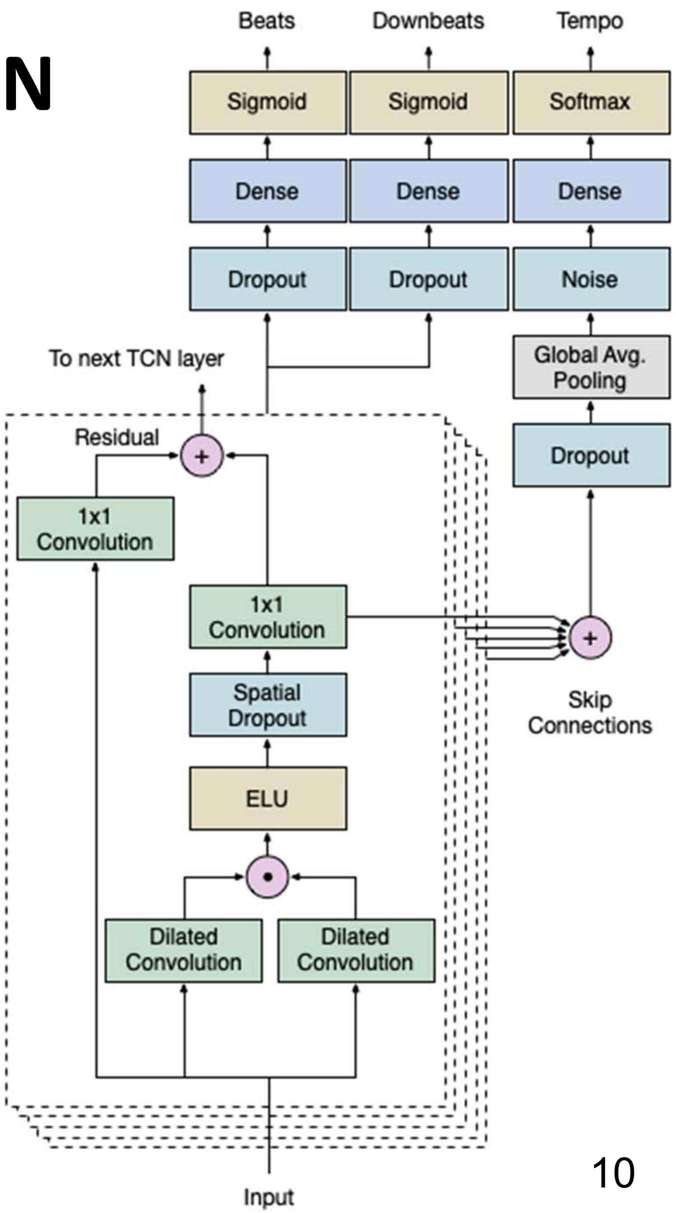
Hands on!

https://colab.research.google.com/drive/1tuOqNyO9gdMmYJsj33fP_QOfpRsm2tmt?usp=sharing

Ref: Böck & Davies, "Deconstruct, analyse, reconstruct: How to improve tempo, beat, and downbeat estimation," ISMIR 2020

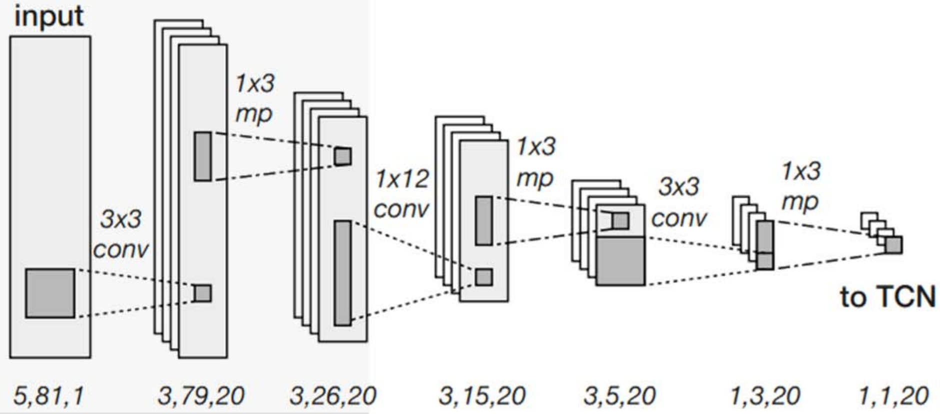
Main Architecture - TCN

```
def residual_block(x, i, activation, num_filters, kernel_size, padding, dropout_rate=0, name=''):
    # name of the layer
    name = name + '_dilation_%d' % i
    # 1x1 conv. of input (so it can be added as residual)
    res_x = Conv1D(num_filters, 1, padding='same', name=name + '_1x1_conv_residual')(x)
    # two dilated convolutions, with dilation rates of i and 2i
    conv_1 = Conv1D(
        filters=num_filters,
        kernel_size=kernel_size,
        dilation_rate=i,
        padding=padding,
        name=name + '_dilated_conv_1',
    )(x)
    conv_2 = Conv1D(
        filters=num_filters,
        kernel_size=kernel_size,
        dilation_rate=i * 2,
        padding=padding,
        name=name + '_dilated_conv_2',
    )(x)
    # concatenate the output of the two dilations
    concat = keras.layers.concatenate([conv_1, conv_2], name=name + '_concat')
    # apply activation function
    x = Activation(activation, name=name + '_activation')(concat)
    # apply spatial dropout
    x = SpatialDropout1D(dropout_rate, name=name + '_spatial_dropout_%f' % dropout_rate)(x)
    # 1x1 conv. to obtain a representation with the same size as the residual
    x = Conv1D(num_filters, 1, padding='same', name=name + '_1x1_conv')(x)
```



PreNet

```python
def create_model(input_shape, num_filters=20, num_dilations=11, kernel_size=5, activation='elu', dropout_rate=0.15):
    # input layer
    input_layer = Input(shape=input_shape)

    # stack of 3 conv layers, each conv, activation, max. pooling & dropout
    conv_1 = Conv2D(num_filters, (3, 3), padding='valid', name='conv_1_conv')(input_layer)
    conv_1 = Activation(activation, name='conv_1_activation')(conv_1)
    conv_1 = MaxPooling2D((1, 3), name='conv_1_max_pooling')(conv_1)
    conv_1 = Dropout(dropout_rate, name='conv_1_dropout')(conv_1)

    conv_2 = Conv2D(num_filters, (1, 10), padding='valid', name='conv_2_conv')(conv_1)
    conv_2 = Activation(activation, name='conv_2_activation')(conv_2)
    conv_2 = MaxPooling2D((1, 3), name='conv_2_max_pooling')(conv_2)
    conv_2 = Dropout(dropout_rate, name='conv_2_dropout')(conv_2)

    conv_3 = Conv2D(num_filters, (3, 3), padding='valid', name='conv_3_conv')(conv_2)
    conv_3 = Activation(activation, name='conv_3_activation')(conv_3)
    conv_3 = MaxPooling2D((1, 3), name='conv_3_max_pooling')(conv_3)
    conv_3 = Dropout(dropout_rate, name='conv_3_dropout')(conv_3)

    # reshape layer to reduce dimensions
    x = Reshape((-1, num_filters), name='tcn_input_reshape')(conv_3)

    # TCN layers
    dilations = [2 ** i for i in range(num_dilations)]
    tcn, skip = TCN(
        num_filters=[num_filters] * len(dilations),
        kernel_size=kernel_size,
        dilations=dilations,
```



Data Sequence Handling & Target Widening

```python
# infer (global) tempo from beats
def infer_tempo(beats, hist_smooth=15, fps=FPS, no_tempo=MASK_VALUE):
    ibis = np.diff(beats) * fps
    bins = np.bincount(np.round(ibis).astype(int))
    # if no beats are present, there is no tempo
    if not bins.any():
        return NO_TEMPO
    intervals = np.arange(len(bins))
    # smooth histogram bins
    if hist_smooth > 0:
        bins = madmom.audio.signal.smooth(bins, hist_smooth)
    # create interpolation function
    interpolation_fn = interp1d(intervals, bins, 'quadratic')
    # generate new intervals with 1000x the resolution
    intervals = np.arange(intervals[0], intervals[-1], 0.001)
    tempi = 60.0 * fps / intervals
    # apply quadratic interpolation
    bins = interpolation_fn(intervals)
    peaks = argrelmax(bins, mode='wrap')[0]
    if len(peaks) == 0:
        # no peaks, no tempo
        return no_tempo
    else:
        # report only the strongest tempo
        sorted_peaks = peaks[np.argsort(bins[peaks])[::-1]]
        return tempi[sorted_peaks][0]
```

```python
# pad features
def cnn_pad(data, pad_frames):
    """Pad the data by repeating the first and last frame N times."""
    pad_start = np.repeat(data[:1], pad_frames, axis=0)
    pad_stop = np.repeat(data[-1:], pad_frames, axis=0)
    return np.concatenate((pad_start, data, pad_stop))
```

```python
def widen_beat_targets(self, size=3, value=0.5):
    for y in self.beats.values():
        # skip masked beat targets
        if np.allclose(y, MASK_VALUE):
            continue
        np.maximum(y, maximum_filter1d(y, size=size) * value, out=y)

def widen_downbeat_targets(self, size=3, value=0.5):
    for y in self.downbeats.values():
        # skip masked downbeat targets
        if np.allclose(y, MASK_VALUE):
            continue
        np.maximum(y, maximum_filter1d(y, size=size) * value, out=y)

def widen_tempo_targets(self, size=3, value=0.5):
    for y in self.tempo.values():
        # skip masked tempo targets
        if np.allclose(y, MASK_VALUE):
            continue
        np.maximum(y, maximum_filter1d(y, size=size) * value, out=y)
```

Data Augmentation

- “We use a simple approach of adding the same training examples with **changed hop-size when computing the STFT**. This results in the same song being represented with a different numbers of frames. This way the **beat positions are "streched" or "squeezed"** and the tempo changes accordingly.”

```
[ ] for fps in [95, 97.5, 102.5, 105]:
        ds = DataSequence(
            tracks={f'{k}_{fps}': v for k, v in tracks.items() if k in train_files},
            pre_processor=PreProcessor(fps=fps),
            pad_frames=pad_frames,
        )
        ds.widen_beat_targets()
        ds.widen_downbeat_targets()
        ds.widen_tempo_targets(3, 0.5)
        ds.widen_tempo_targets(3, 0.5)
        train.append(ds)
```

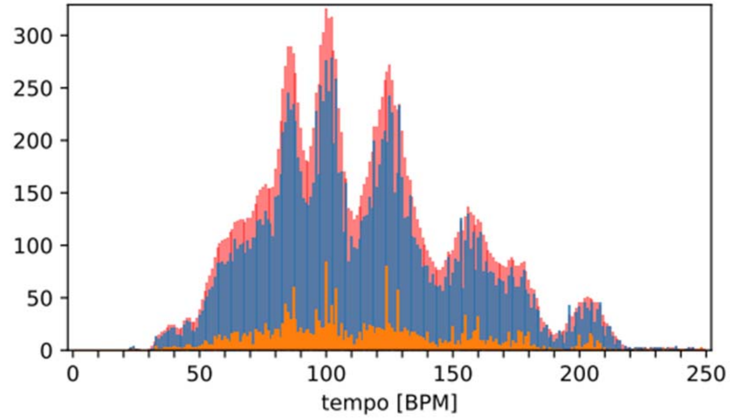


Figure 2: Tempo distribution of original tempo annotations (orange, foreground), after data augmentation (blue) and target widening (red, background).

Post-processing

```
[ ] # track beats with a DBN
    beat_tracker = madmom.features.beats.DBNBeatTrackingProcessor(
        min_bpm=55.0, max_bpm=215.0, fps=FPS, transition_lambda=100, threshold=0.05
    )

    # track downbeats with a DBN
    # as input, use a combined beat & downbeat activation function
    downbeat_tracker = madmom.features.downbeats.DBNDownBeatTrackingProcessor(
        beats_per_bar=[3, 4], min_bpm=55.0, max_bpm=215.0, fps=FPS, transition_lambda=100
    )

    # track bars, i.e. first track the beats and then infer the downbeat positions
    bar_tracker = madmom.features.downbeats.DBNBarTrackingProcessor(
        beats_per_bar=(3, 4), meter_change_prob=1e-3, observation_weight=4
```

```
def detect_tempo(bins, hist_smooth=11, min_bpm=10):
    min_bpm = int(np.floor(min_bpm))
    tempi = np.arange(min_bpm, len(bins))
    bins = bins[min_bpm:]
    # smooth histogram bins
    if hist_smooth > 0:
        bins = madmom.audio.signal.smooth(bins, hist_smooth)
    # create interpolation function
    interpolation_fn = interp1d(tempi, bins, 'quadratic')
    # generate new intervals with 1000x the resolution
    tempi = np.arange(tempi[0], tempi[-1], 0.001)
    # apply quadratic interpolation
    bins = interpolation_fn(tempi)
    peaks = argrelmax(bins, mode='wrap')[0]
    if len(peaks) == 0:
        # no peaks, no tempo
        tempi = np.array([], ndmin=2)
    elif len(peaks) == 1:
        # report only the strongest tempo
        ret = np.array([tempi[peaks[0]], 1.0])
        tempi = np.array([tempi[peaks[0]], 1.0])
    else:
        # sort the peaks in descending order of bin heights
        sorted_peaks = peaks[np.argsort(bins[peaks])[::-1]]
        # normalize their strengths
        strengths = bins[sorted_peaks]
        strengths /= np.sum(strengths)
```

Library: Madmom

https://github.com/CPJKU/madmom