Machine Learning Engineer Nanodegree

Capstone Project - Audio key estimation of digital music with CNNs

Daniel Hellwig

August 22nd, 2018

I. Definition

Project Overview

The art of mixing recorded music in real time is known as DJing and performed by a Disc Jockey (DJ). DJs use specialized equipment that can play at least two sources of recorded music simultaneously to create smooth transitions from one song to another [1].

The way of how the transitions are made became extremely versatile nowadays. Hereby one DJ technique experienced a renaissance in 2006: harmonic mixing. The goal of harmonic mixing is to transition between songs of the same or related key, notes of a certain scale that form the basis of a piece of music. This technique enables a DJ to make smooth continuous mixes and prevents unstable tone combinations, known as dissonance [2].

This project deals with the learning task to estimate audio keys of digital music. A mulitclass classifier is trained using samples of digital music files. The Million Song Dataset (MSD) [3] is utilized to select appropriate songs and includes information about their tonic note and mode as well as how confident both are. The learning task is limited to diatonic scales, typically used in western music.

Problem Statement

The task is to estimate the audio key of a digital 30 second sample of a western music piece.

The task includes:

- Data Retrieval
    - retrieve the Million Song Dataset and select appropriate songs
    - create a dataset of selected songs
- Data Preprocessing
    - perform preparatory signal processing tasks
    - extract features of the song dataset, output is spectrogram images
    - create a dataset of song spectrograms
- Model Preparation/Training
    - preprocess the spectrograms
    - build and train a multiclass classifier
- Model Evaluation/Comparison
    - evaluate the classifier with certain metrics
    - benchmark the classifier against another key estimation software with the help of the MIREX evaluation procedure

The resulting classifier can be used to determine the key of western music pieces.

Metrics

The used metric depends on the following question: Out of all estimated keys for given music pieces, how many were classified correctly? This can be calculated by the accuracy for binary classification problems.

Since the songs within each key class are going to be unbalanced due to a high variety of patterns in the spectrograms, it is better to use the F-beta score with beta=1 instead:

$$F_1 = 2 \cdot \frac{Prec * Rec}{Prec + Rec}$$

with Prec = Precision, Rec = Recall.

II. Analysis

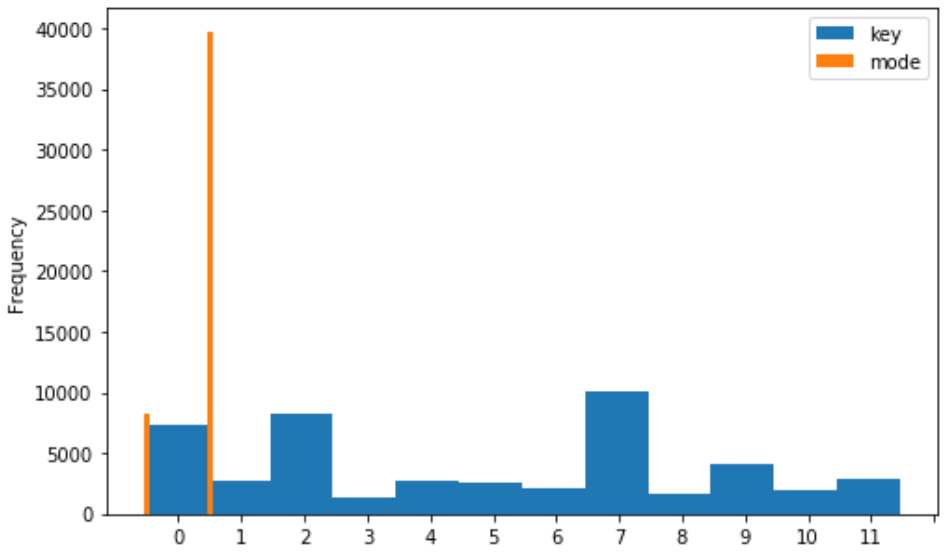
Data Exploration

The feature dataset used for the classifier holds spectrogram images of 30 second song samples. The songs itself are selected beforehand with the help of the Million Song Dataset.

The Million Song Dataset is a collection of audio features and metadata for a million popular songs [4], but it does not contain the songs itself. Two additional files which come along with the MSD are from interest: the MSD summary file with song metadata of the whole dataset [5] and the track file with all songs and their unique IDs [6].

The MSD summary file is used to select all songs which have a key and mode confidence of at least 75%. Subsequently, the selected songs and the track file are joined together and cleaned up. This results in a list of 47913 songs with their attributes ‘key’, ‘key_confidence’, ‘mode’, ‘mode_confidence’, ‘tempo’, ‘track_id’, ‘song_id’, ‘artist_name’, ‘song_title’.

The distribution of tonic note (in the MSD wrongly described as key) and mode is shown below. There exist by far more songs with mode major (1) than minor (0). The distribution of the songs tonic note is unbalanced over the whole 47913 songs.

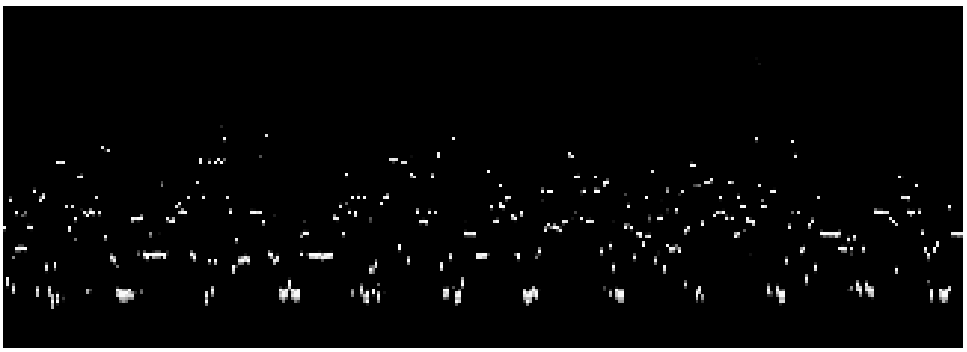


(distribution of tonic note (key) and mode of selected songs)

Out of the generated list are 1200 songs manually chosen, which results in 50 song samples per tonic-mode pair. The reason behind is, that tonic-mode 3-0 consists of the lowest amount of selected songs (142 at all). Furthermore, it is sure that from this amount of songs may only 50% available as a sample (71 at all).

Once the chosen song samples have been recorded, edited and saved in 16 Bit 44100 Hz WAV format, the feature dataset can be created. With the help of certain signal processing tasks and a short-time Fourier transform, spectrograms of the songs are saved as images in directories per tonic-mode pair.

An example of a spectrogram image is shown below:

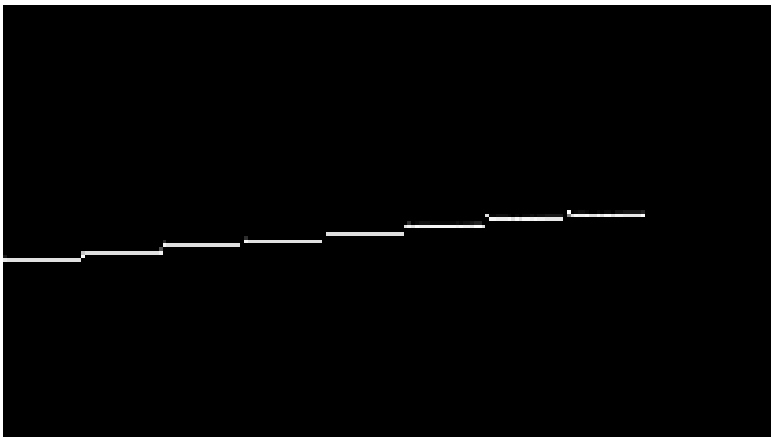


(spectrogram of 30 second song sample ‘Phil Collins - Everyday’)

The images are of dimension 150 x 128 pixel and have 3 channels (RGB).

Exploratory Visualization

To understand how specific patterns shall be recognized by the classifier, the spectrogram of a clean audio signal in C major is taken for reference:



(C4 major audio spectrogram)

The image is exactly 128 pixels high, caused by the number of displayed note pitches within the image. Each height pixel represents the frequency of a note, beginning at the bottom with note $C_{-1} = 8.176\ Hz$ and ending at the top with note $G_9 = 12544\ Hz$. Since there are 11 octaves displayed each 12 notes, except of the last octave showing only 8 notes, the sum of height pixels = 12*11-4 = 128. To get a better vision of the height pixels and their corresponding notes/frequencies, you can imagine a piano rotated by 90 degrees next to the image, where each piano key represents one height pixel.

The C major scale is based on tonic C and incorporates the pitches D, E, F, G, A, B (, C). Although, a C major audio piece can only consist of the pitches C, E, G and is then known as the C major chord.

Another unique pattern to see in this example is given by the distance between each note pixel and the next one:

(C4 major spectrum, magnified)

note	height pixel no. note	next note	height pixel no. next note	distance in pixel
C	48	D	50	2
D	50	E	52	2
E	52	F	53	1
F	53	G	55	2
G	55	A	57	2
A	57	B	59	2
B	59	(C)	60	1

The distance pattern 2-2-1-2-2-2-1 shows the number of semitones between each note (interval) and this one is characteristic for a major scale. In contrast, the minor scale has the pattern 2-1-2-2-1-2-2.

The feature dataset has per tonic-mode pair the same amount of spectrogram images. However, the dataset is considered as being unbalanced since spectrograms of one tonic-mode pair can strongly differ from each other to the human eye.

The width of the spectrogram image represents the time in seconds of the song sample. The width is 150 pixel which results in 0.2 seconds per width pixel.

Algorithms and Techniques

To create the feature dataset of spectrograms, the short-time Fourier transform (STFT) is used to determine the change of frequency in a signal over time.
There are several parameters that affect the outputted spectrogram of an audio signal:

- decimation parameters: low-pass filter function and downsampling factor
- length of a signal time chunk
- used window function on the signal and length of the window
- number of zeros to pad at the end of a time chunk (affects the resolution in the frequency domain)

For the learning task a convolutional neural network is used which is commonly applied to visual imagery. The output of the network is the probability of each tonic-mode pair in a music piece, whereby the maximal probable tonic-mode pair represents the key of it.
The parameters of the convolutional network are:

- number of epochs = training cycles of the network
- batch size = number of input samples to take for one weight update
- type of optimizer for a gradient step
    - learning rate
    - specific parameters (dependent on used optimizer)
- type of loss function to evaluate a gradient step
- type of regularizer and its parameters
- the architecture of the network itself: how many convolutional layers, pooling layers, fully connected layers and its parameters (window size, stride, padding of the input)

Benchmark

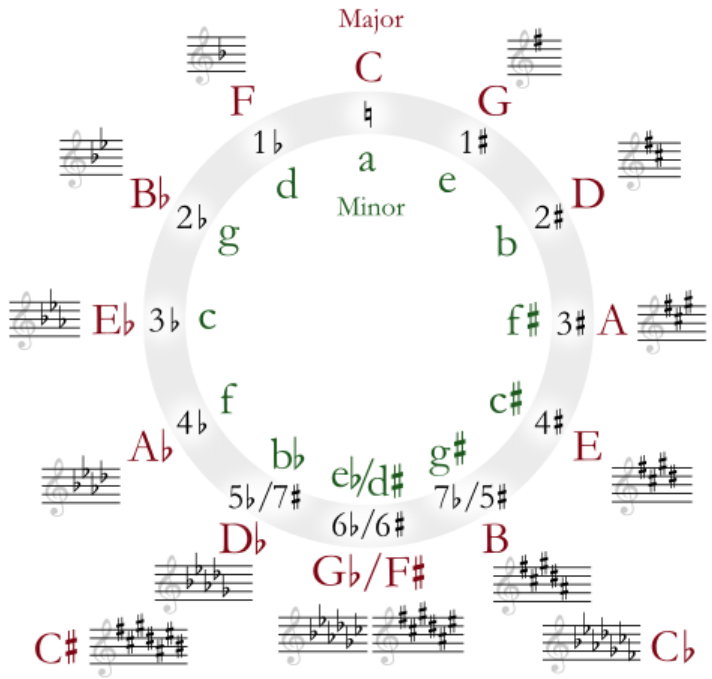
For the comparison to another key estimation software, a second metric named MIREX evaluation procedure is used [7].

The metric compares the identified key by the algorithm against the actual key of the piece and gives points dependent on their relationship:

relation to correct key points:	
same	1.0
distance of perfect fifth	0.5
relative major/minor	0.3
parallel major/minor	0.2

The distance of perfect fifth can be either the dominant (fifth) or subdominant (fourth) from the tonic note of the actual key.

The circle of fifths is shown below to better understand how the evaluation works:



(circle of fifths)
attribution: By Just plain Bill [GFDL (http://www.gnu.org/copyleft/fdl.html) or CC-BY-SA-3.0 (http://creativecommons.org/licenses/by-sa/3.0/)], from Wikimedia Commons

Example: A music piece may have the actual key C major. The dominant is G major, the subdominant is F major. The relative key is A minor. The parallel key is C minor.

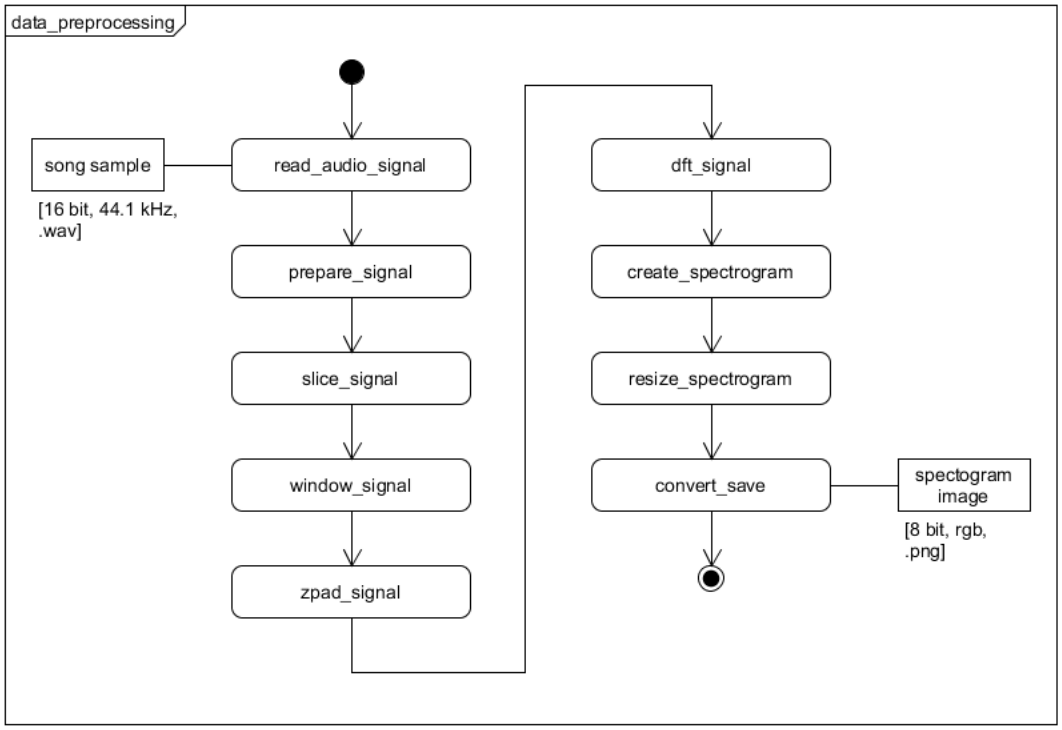
III. Methodology

Data Preprocessing

This section discusses the construction of the feature dataset only, since it is the input data of the classifier.

The implementation can be found in directory /00.hlp/fft/fft.ipynb (Jupyter notebook).

The following outline shows the tasks to get from an audio signal to its spectrogram:



(outline data preprocessing)

Function `read_audio_signal (filename)`

Reads in the WAV audio file and returns the signal (NumPy 2D-ndarray) as well its sampling rate. The function checks the length of the signal to be 30 seconds and corrects it if necessary.

Function `prepare_signal (sig_fs, sig) / PARAM_DEC_FAC = 2`

Reduces the signal to one channel (mono) by building the mean over all signal channels. Subsequently the signal is decimated which includes applying a FIR low-pass filter (finite impulse response) and downsampling the signal by a factor of 2, i.e. throwing away every second sample of the signal. The function returns the new signal and new sampling rate. This preparation does not affect the quality of the signal since it is the result of the Nyquist-Shannon sampling theorem.

Function `slice_signal (sig) / PARAM_N = 4410`

Splits the signal into defined time chunks, each 4410 samples long = 0.2 seconds. The length is derived from the maximum song tempo in the dataset = 248.323 beats per minute (bpm), safely rounded to 300 bpm. This results in the following resolution to catch the beats in every song:

$$300\ bpm = \frac{300}{60}\ bps = 5\ bps = 1\ beat\ per\ 0.2\ s = 0.2\ s \cdot 22050\ Samples\ per\ s = 4410\ Samples$$

The function returns a NumPy 2D-ndarray with shape (num_slices, PARAM_N).

Function `window_signal (sig_slices) / PARAM_WIN = scipy.signal.hann (Hanning window)`

Multiplies the signal slices with a window function. The Hanning window is used. Returns the windowed signal slices.

Function `zpad_signal (sig_slices) / PARAM_N_ZEROS = 32768 - PARAM_N`

Appends a defined number of zeros to every signal slice, since the number of samples of a time-domain signal going to be Fourier transformed dictates the resolution in the frequency-domain. The number of zeros to append are derived from the smallest difference in note frequency to be visualized:

$$freq(C\#_0) - freq(C) = (17.324 - 16.351)\ Hz = 0.973\ Hz \rightarrow round\ down\ (0.973\ Hz) = 0.9\ Hz$$

Frequency bin distance of the Fourier transformed signal:

$$dist_{freq-bins} = \frac{1}{PARAM\_N + PARAM\_N\_ZEROS} \cdot sampling\ rate \leq 0.9\ Hz$$

$$PARAM\_N\_ZEROS = \frac{sampling\ rate}{0.9\ Hz} - PARAM\_N = \frac{22050}{0.9} - PARAM\_N = 24500 - PARAM\_N$$

$$PARAM\_N\_ZEROS + PARAM\_N \geq 24500\ Samples\ \rightarrow next\ power\ of\ 2 = 32768\ Samples$$

The STFT expects a number of samples to the power of 2. The function returns the zero-padded signal slices.

Function `dft_signal (sig_fs, sig_slices)`

Performs the short-time Fourier transform on the signal slices. Returns the absolute magnitude of the Fourier coefficients as well as the frequency bins.

Function `create_spectrogram (sig_fft_mags, freq_bins_f)`

Creates the spectrogram by taking the first 32 maximum magnitudes per signal slice and converts those into a range of [0, 255] (bit range of an image). Returns a spectrogram as NumPy 2D-ndarray, where

- rows = maximum pitch frequencies, corresponds to the width of the current image
- columns = time chunk, corresponds to the height of the current image

Function `resize_spectrogram (spectro)`

Resizes the spectrogram to useful dimensions for the convolutional neural network. Currently unused.

Function `convert_save (spectro)`

Makes an image of the spectrogram, rotates it by 90 degrees (columns = pitch frequencies = image height, rows = time chunk = image width), converts the image to RGB space (= 3 channels) and saves the image in PNG format.

The described data preprocessing tasks are applied to every song sample and the outputted spectrogram images are saved in a defined directory tree:

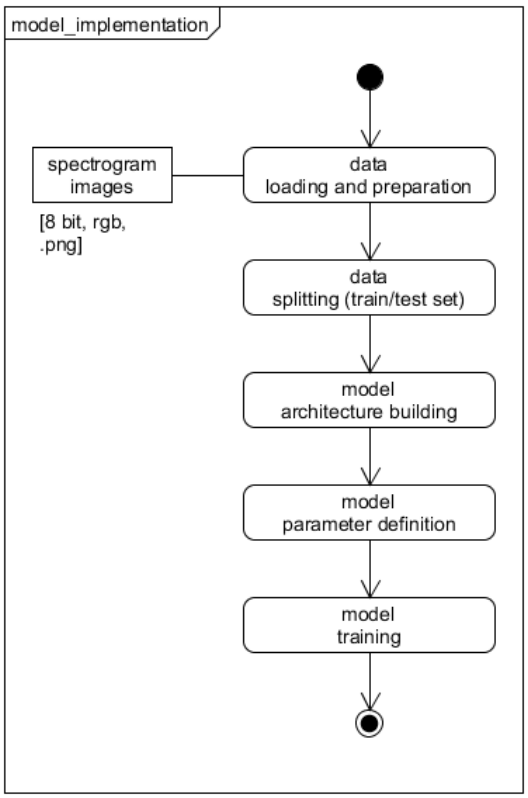
```
src_spectro                          main directory
  0-0                                tonic-mode sub-directory
    sample_spectrogram.png           spectrogram image
  ...
  11-1
```

The feature dataset src_spectro is then used as input for the convolutional neural network.

Implementation

The implementation can be found in the main directory, keyestcnn.ipynb (Jupyter notebook).

The following outline shows the tasks of the implementation:



(implementation)

The feature dataset is loaded and converted into 4D tensors, suitable for a convolutional neural network built upon Keras with TensorFlow backend.

The model architecture is shown below:

Layer (type)	Output Shape	Param #
input (InputLayer)	(None, 128, 150, 1)	0
conv2d_1 (Conv2D)	(None, 128, 150, 16)	32
conv2d_2 (Conv2D)	(None, 128, 150, 32)	2080
maxp_2 (MaxPooling2D)	(None, 64, 75, 32)	0
conv2d_3 (Conv2D)	(None, 64, 75, 64)	8256
maxp_3 (MaxPooling2D)	(None, 32, 37, 64)	0
avg_flatten (GlobalAveragePo	(None, 64)	0
output (Dense)	(None, 24)	1560

Total params: 11,928
Trainable params: 11,928
Non-trainable params: 0

There are three convolutional layers, two max pooling layers and a global average pooling layer right before the output to flatten the processed data.

Since the pitch frequencies of the spectrogram are pixel accurate, the intention is to keep the original dimensions as long as possible and applying more and more filters step by step. Therefore, the network starts with two convolutional layers and then downsizes the image by a factor of 2. The third convolutional layer applies more filters and trying to get deeper insights of spectrogram patterns. All convolutional layer nodes are activated by a ReLU.

The output layer reflects the 24 tonic-mode classes (2 modes, each 12 tonics) and has a Softmax activation.

The applied loss function is the mean squared error. A stochastic gradient descent optimizer is used, with a learning rate = 0.0001 and a momentum = 0.8.

The applied metric to evaluate the model is the F-beta score, with beta = 1.

Model training is done in 100 epochs and a batch size of 10 spectrogram images. The model weights are saved, if the validation step shows improvement.

Refinement

I can’t say any refinements yet.

I tried several things already, but without documenting yet:

- Data Preprocessing:
  - changing the time chunk length (1024, 4096, 8096 Samples)

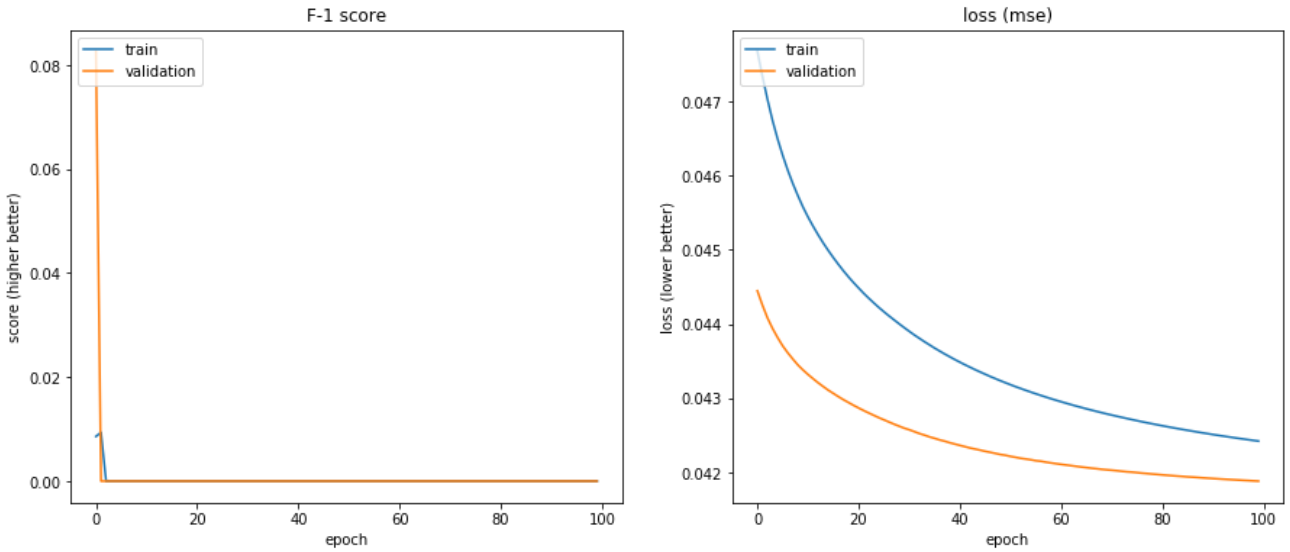
    - changing the number of zeros to pad = the length of the signal to be processed by STFT (double of the time chunk length, no zeros padded), but after some thinking it turned out, that the best is to determine it by the resolution wanted in the frequency-domain
    - changing downsampling factor to 10 (showed no improvement)
    - changing the size of the outputted spectrogram (88 px height, 116 px height, same width and height): the classifier showed better results at bigger pictures with same width and height
- Model Training
    - optimizer changed to Adamax: shows no improvement yet
    - changed SGD optimizer parameters: lower learning rate and a bit momentum formed a nicer learning curve
    - different network architectures: shows no significantly better results

IV. Results

Model Evaluation and Validation

TOPIC ONGOING

Below graph shows the progress of F-beta score and loss over all epochs:



(score and loss of classifier)

loss per epoch:

- gradient steps start with a loss of 0.052, end by 0.042 and show a smooth concave curve. The curve couldn't be better except a faster drop in the first 10 epochs
- the worse: mse after 1st epoch = 0.052 - the CNN learns very slow and in tiny steps

F-beta score:

- evaluation metric immediately drops to zero after some epochs - the CNN doesn't learn anything yet

The model was tested with random song samples, not part of the dataset. There was no match at all.

Justification

TOPIC ONGOING

Yet no benchmarking done.

V. Conclusion

Free-Form Visualization

TOPIC ONGOING

Reflection

TOPIC ONGOING

Improvement

input data

(1) The used dataset only has 240 samples for training, validation and test. This is by far nothing for the CNN.

Todo: retrieve more samples for the dataset

(2) A quick look at random spectrograms show kind of chaotic information - as a human being it is hard to tell if there's any structure behind each key-mode pair. This may apply to the CNN too.

Todo: find additional filter techniques / methods to clearly bring out structures for the CNN

(3) Songs can change in key over their whole length.

Todo: take appropriate sample of a song - ommit bridges, refrains, silent passages, noisy songs

_

model training

The model was trained for 100 epochs, each in batches of 10 samples per feedfwd-backprop step. To make sure that the architecture is well suited, more epochs shall be run.

Todo: increase epochs, change batch size

_

model architecture

Todo: To better understand the insight of the CNN, visualize the filter of the convolutions. May there be enlightenment what kind of architecture works best.

References

[1] https://en.wikipedia.org/wiki/Disc_jockey#History

[2] https://en.wikipedia.org/wiki/Harmonic_mixing

[3] https://labrosa.ee.columbia.edu/millionsong/

[4] https://labrosa.ee.columbia.edu/millionsong/

[5] http://labrosa.ee.columbia.edu/millionsong/sites/default/files/AdditionalFiles/msd_summary_file.h5

[6] http://labrosa.ee.columbia.edu/millionsong/sites/default/files/AdditionalFiles/unique_tracks.txt

[7] http://music-ir.org/mirex/wiki/2005:Audio_and_Symbolic_Key_Finding#Evaluation_Procedures