Machine Learning Engineer Nanodegree

Capstone Project: Audio key estimation of digital music with CNNs

Daniel Hellwig

September 12th, 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:

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

with Prec = Precision, Rec = Recall.

II. Analysis

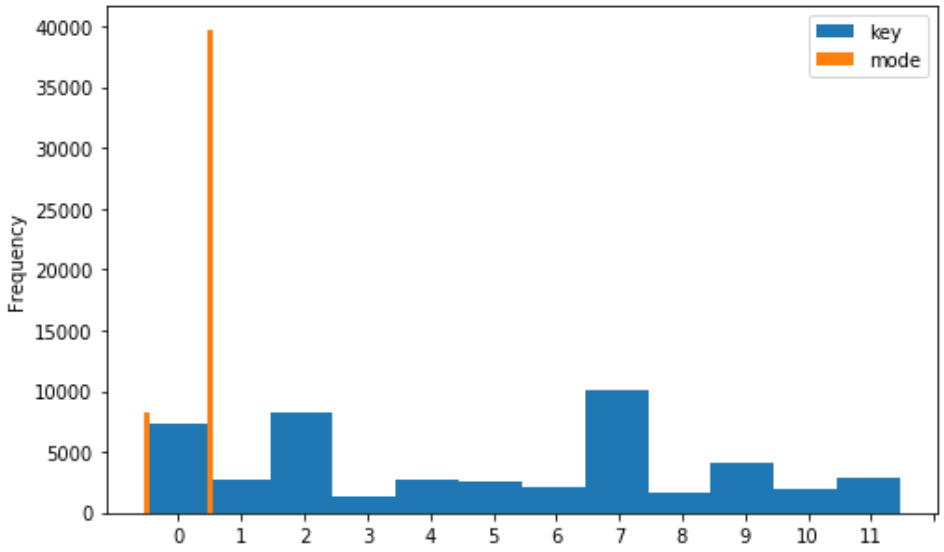
Data Exploration

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 [5] with song metadata of the whole dataset and the track file [6] with all songs and their unique IDs.

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’.

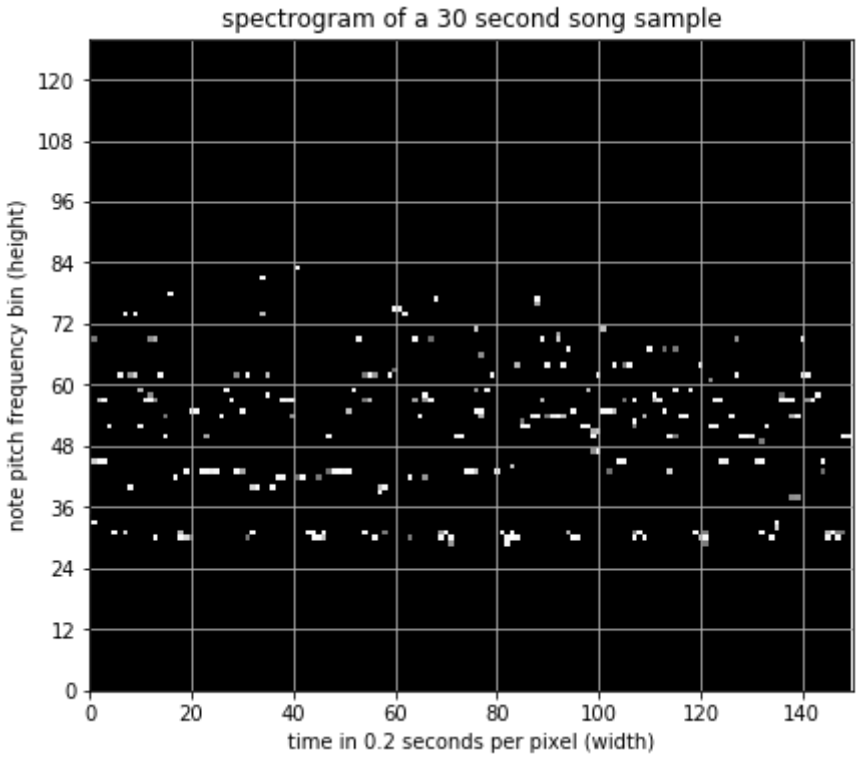
The distribution of tonic note (in the MSD wrongly described as key) and mode is shown below. There exist by far more songs with mode major (1) than minor (0). The distribution of the songs tonic note is unbalanced over the whole 47913 songs.



Out of the generated list are 360 songs manually chosen, resulting in 15 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:

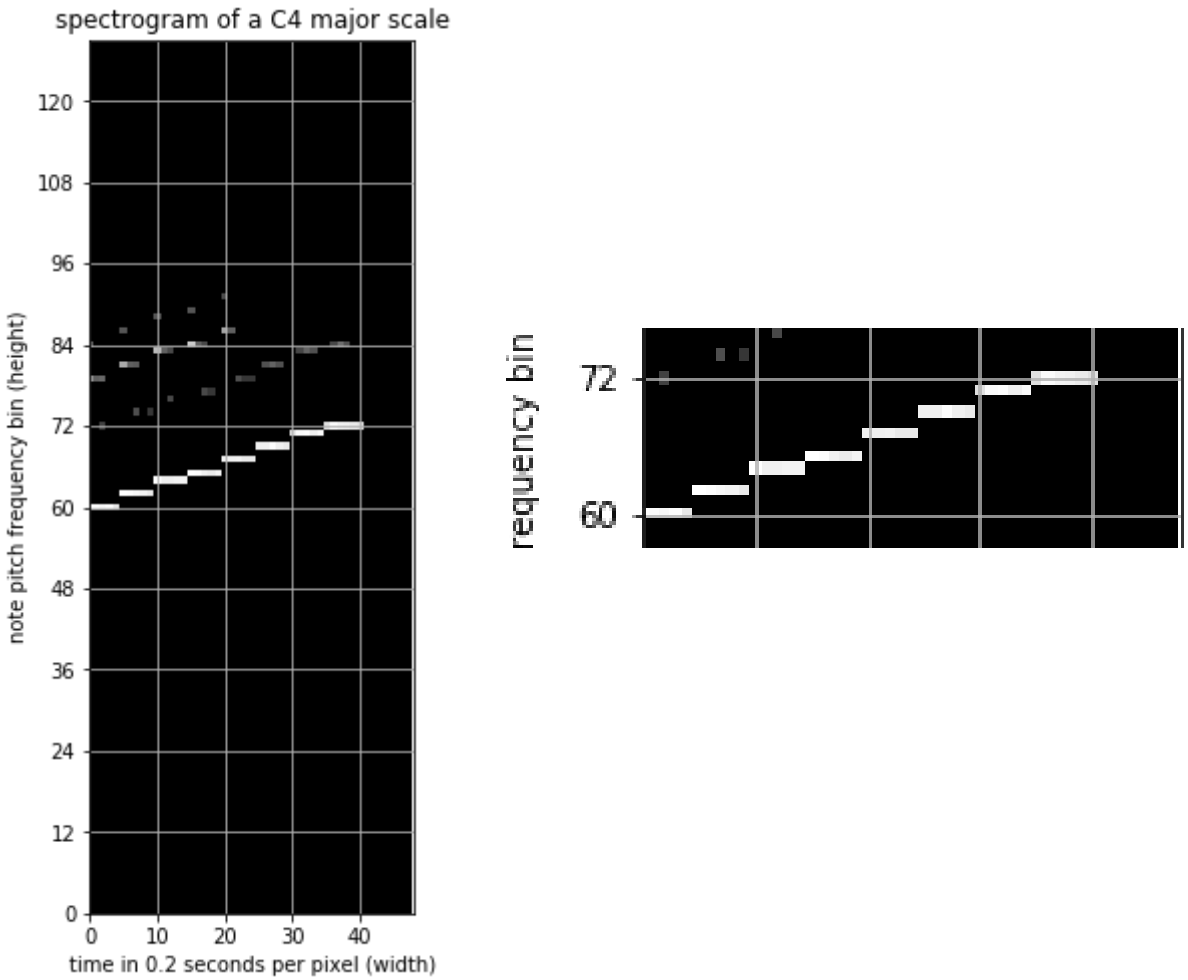


(spectrogram of 30 second song sample ‘Phil Collins - Everyday’)

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 full and magnified)

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:

note	height pixel no. note	next note	height pixel no. next note	distance in pixel
C	60	D	62	2
D	62	E	64	2
E	64	F	65	1

F	65	G	67	2
G	67	A	69	2
A	69	B	71	2
B	71	(C)	72	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

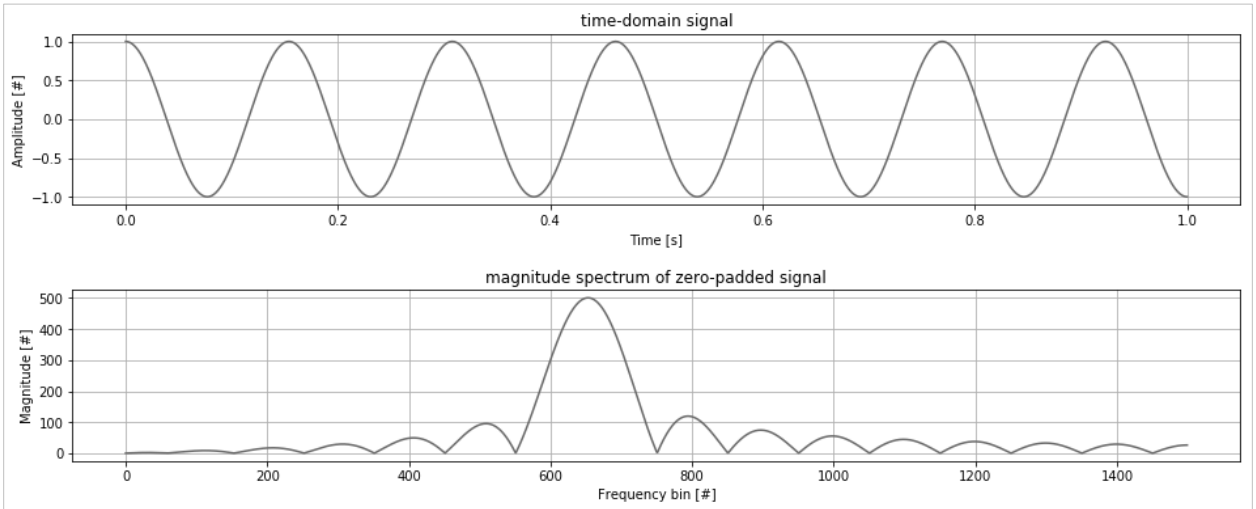
To create the feature dataset of spectrograms, the short-time Fourier transform (STFT) is used to determine the change of frequency in a signal over time.

Digital music pieces represent sampled analog signals in the time-domain, where each sample corresponds to the amplitude of a sound wave. To determine the change of frequency in a signal over time, one uses the discrete Fourier transform (DFT) over short periods of time, known as the short-time Fourier transform (STFT).

In general, the DFT transforms a sequence of N equally-spaced complex samples $\mathrm{x[n]} = \{\mathrm{x}_0, \ldots, \mathrm{x}_{\mathrm{N-1}}\}$ into another sequence of complex samples $\mathrm{X[k]} = \{\mathrm{X}_0, \ldots, \mathrm{X}_{\mathrm{N-1}}\}$ and is defined by:

$$\mathrm{X[k]} = \sum_{\mathrm{n}=0}^{\mathrm{N-1}} \mathrm{x[n]} \cdot \mathrm{e}^{-\mathrm{i}\frac{2\pi}{\mathrm{N}}\mathrm{kn}} = \sum_{\mathrm{n}=0}^{\mathrm{N-1}} \mathrm{x[n]} \cdot \left[\cos\left(\frac{2\pi}{\mathrm{N}}\mathrm{kn}\right) - \mathrm{i}\sin\left(\frac{2\pi}{\mathrm{N}}\mathrm{kn}\right)\right]$$

By that, the outputted sequence of complex samples builds a signal of same length, but in the frequency-domain.

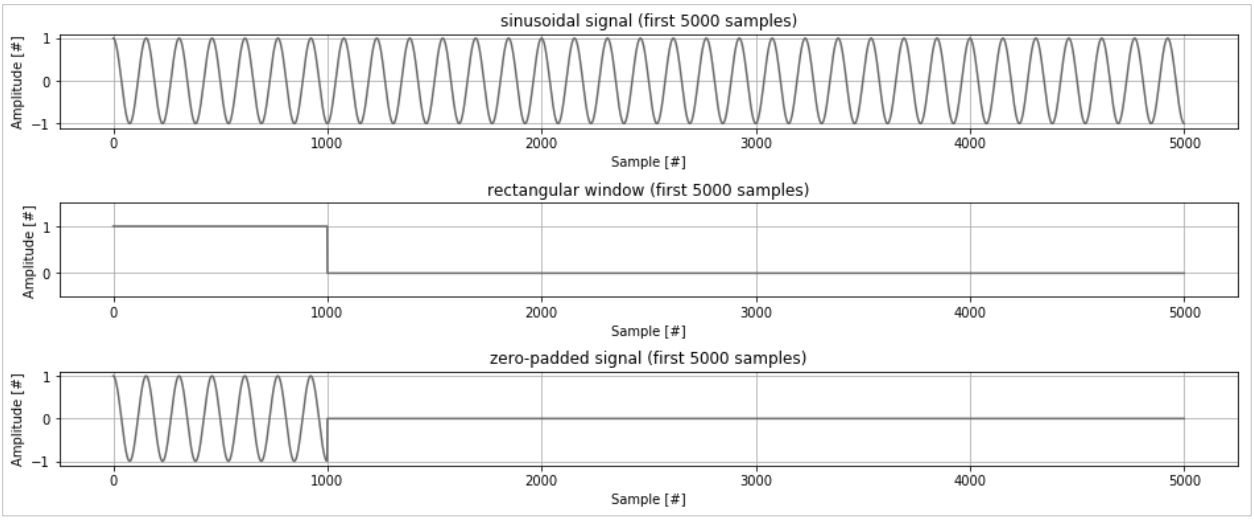


The sinusoidal signal $\mathrm{x(t)} = \cos\left(2\pi \cdot 6.5 \cdot \mathrm{t}\right)$ results in a Fourier transformed signal with maximum magnitude at frequency bin $\mathrm{argmax}\left(\mathrm{X}\right) = 650$.

Since the frequency of a digital music piece can considered to be stationary for a short time only, an analysis window is defined. To do so, a windowing function is applied to the signal, which is non-zero for the defined analysis window:

$$X(n) = \sum_{m=-\infty}^{m=\infty} x[m]\, w[n-m] \cdot e^{-i\frac{2\pi}{N}kn}$$

, where $x[m]\, w[n-m]$ is a short time section of the digital music piece $x[m]$.



In practice, the STFT is applied as follows: The discrete audio signal is broken up into time chunks, followed by applying the windowing function. Then each windowed time chunk is Fourier transformed by a DFT. Subsequently all transformed time chunks are composed together to a time-domain signal showing the change of the frequency spectrum.

There is a trade-off to make in the resulting spectrum between the resolution of the shown frequencies and the time at which the frequencies change, since the STFT uses a fixed number of samples per time chunk. To try to overcome this issue, the windowed time chunk is zero-padded prior applying the DFT.

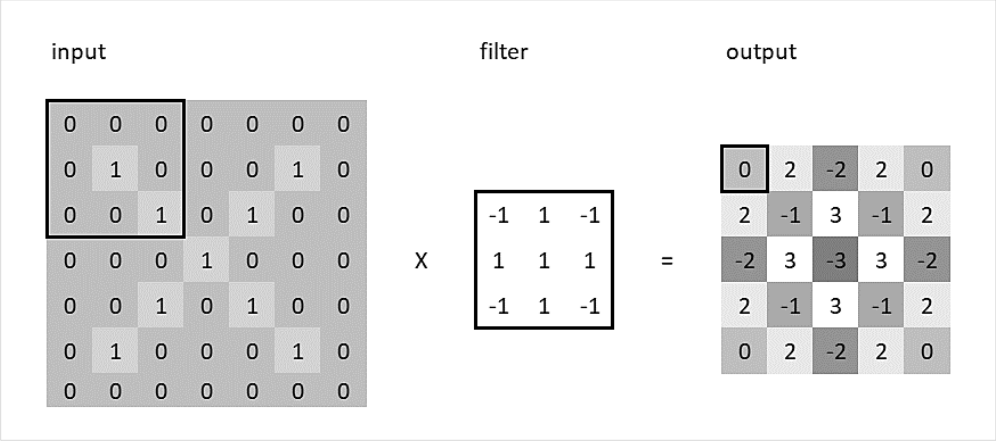
Since the interest lies in the change of specific note pitches, the Fourier transformed audio signal spectrum is modified by putting close frequencies of a time chunk into certain frequency bins. These bins are defined by the Scientific pitch notation (SPN) [8], a method to specify a musical pitch as a combination of musical note name and the pitch's octave. The pitch-note-octave combination ranges from $C_{-1} = 8.176\ \text{Hz}$ (bin 0) to $G_9 = 12544\ \text{Hz}$ (bin 127), whereas each bin's edge to the next bin is $\sqrt[12]{2}$ times the pitch (i.e. $D_{-1} = C_{-1} \cdot \sqrt[12]{2} = 8.662\ \text{Hz}$ (bin 1)). The ratio factor is characteristic for the twelve-tone equal temperament [9], a tuning system commonly used in western music. The bin numbers [0, 127] correspond to the MIDI note number.

Both, the frequency-time resolution trade-off and the 'binification' of the frequency spectrum represent the first convolution of the input data in the whole learning task.

For the learning task a convolutional neural network (CNN) 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.

CNNs are a specific type of neural networks that accept matrices as input. This is a huge advantage for this learning task, since the song spectrograms are 2D-matrices, and usual multilayer perceptron (MLPs) take 1D-vectors as input and make use of fully connected layers only.

The main part of a CNN consists of convolutional layers that have neurons arranged in 3 dimensions (width, height and depth of an image) and its task is to learn filters by moving them over the input image and computing dot products between the filter values and the part of the image the filter is currently at [10]. Below example shows how the layer works:

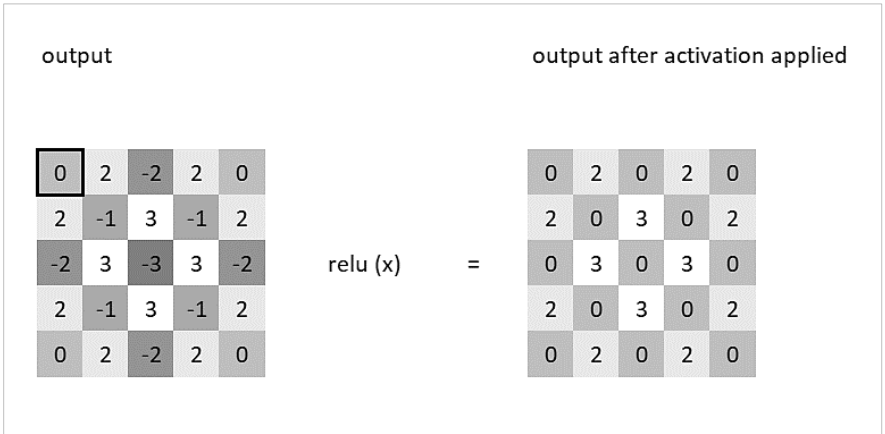


The 3x3 filter is applied on the outlined 3x3 part of the input, the sum of the product is calculated as the output (the outlined value = 0). Then the filter is moved by a stride of 1 column to the right, calculating the next output value (=2). If the right border of the input is reached, the filter moves to the first column and by a stride of 1 row down. This is repeated until the last row and column of the input. Important to notice is that a valid padding is used here (only valid input values are considered by the filter), which leads to a convolution of the 7x7 input to a 5x5 output (3x3 filter can move 7-3+1 = 5 times over the input in both dimensions). There can be as many filters as wanted in one convolutional layer and they are applied on every depth dimension of the input. The filters are learned during the feedforward-backpropagation cycle of a training epoch.

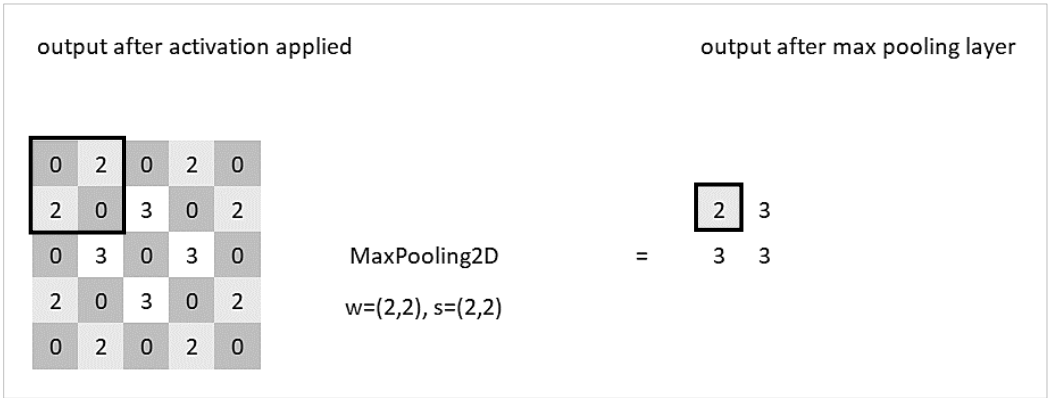
Another part of the CNN is the activation function, used as threshold and normalizer of layer outputs. A common used activation is the rectified linear unit (ReLU):

$$\text{relu}(x) = \begin{cases} x \text{ if } x \geq 0 \\ 0 \text{ if } x < 0 \end{cases}$$

Applied to the output of above example:

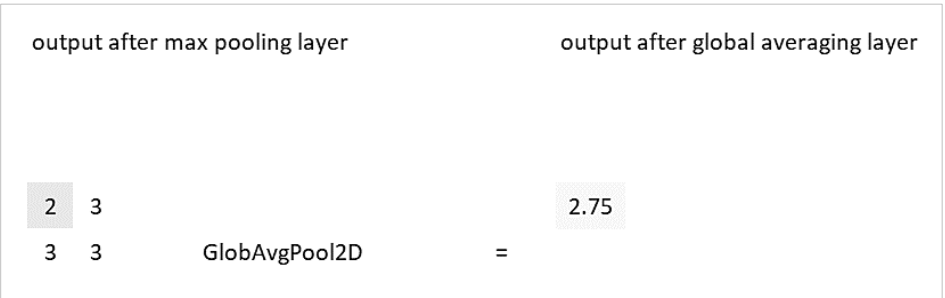


The second most important part of a CNN is a max pooling layer and a way to scale its input down while preserving the most important information [11]. Hereby the input is again windowed as if a filter would be applied, but outputting only the maximum value of the windowed input:



The max pooling layer with window size $w = (2,2)$ calculates the maximum value of array (0, 2, 2, 0), resulting in output value = 2. Then the window moves by a stride s=2 to the right, repeating the calculation (max(0, 2, 3, 0)=3). If there's no space in the input image to fit the window anymore, it moves to the first column and s=2 rows down (max(0, 3, 2, 0)=3). In this case, the downscaling factor is 2, making an 5x5 input to a 2x2 input (5/2 = 2). Important to mention is, that input images not divisible by factor 2 are going to lose information during this process.

Another used part is the global average pooling layer and works similar to the max pooling layer, but calculating the average of all input values at once for each existing filter:



The values of the input are summed up and divided by the number of values. 2+3+3+3 = 11 / 4 = 2.75. This is done per existing filter and leads to a flattened output array which can be then connected to a fully connected layer for classification as known in MLPs.

Benchmark

The classifier is benchmarked against the open source software KeyFinder [12], a key detection tool for DJs. KeyFinder was developed by Ibrahim Sha'ath as part of his MSc in Computer Science back in 2011 and has a pretty good accuracy compared to proprietary software like Rapid Evolution and Mixed In Key. Since KeyFinder is a robust software and made its name in the DJ world, it surely is a good benchmark.

To compare results between the benchmark and the classifier, a new metric is introduced: the MIREX evaluation procedure [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	distance of perfect fifth	relative major/minor	parallel major/minor
1.0	0.5	0.3	0.2

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

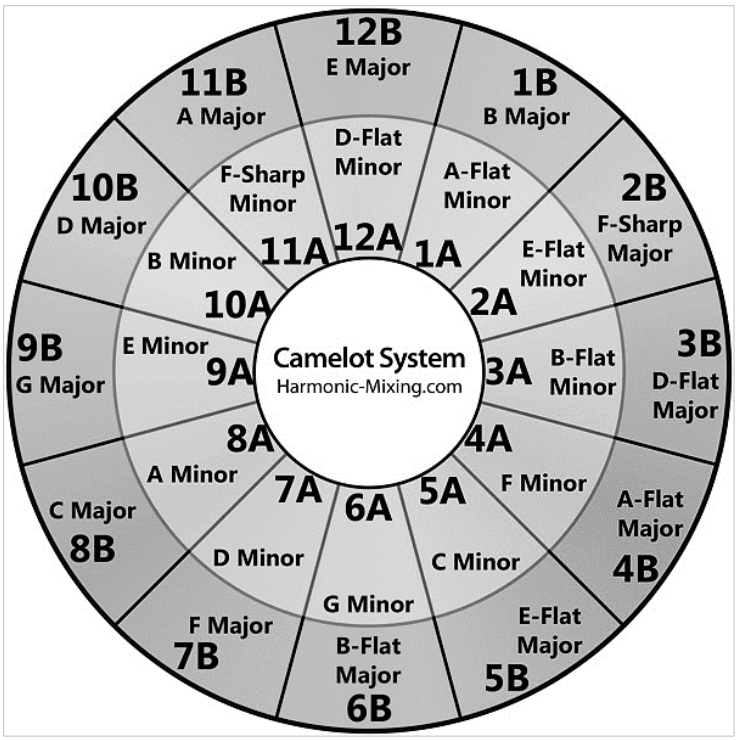
The reasons for taking the MIREX metric rather than mathematical metrics used for learning algorithms are:

- clearly comparable to the benchmark, since it fungates as the baseline
- better fits the applications in real world, since music is a complex combination and makes key estimation a challenging task - marginal errors are allowed and won't be punished too harsch

Below table shows the 24 tonic notes and its distance, relative and parallels notes.

note	dom (fifth)	subdom (fourth)	relative	parallel		camelot wheel	note circle of 5th
1.0	0.5	0.5	0.3	0.2			
C	G	F	Am	Cm		8B	C
G	D	C	Em	Gm		9B	G
D	A	G	Bm	Dm		10B	D
A	E	D	F#m	Am		11B	A
E	B	A	C#m	Em		12B	E
B	F#	E	G#m	Bm		1B	B
F#	C#	B	D#m	F#m		2B	F#
C#	G#	F#	A#m	C#m		3B	Db
G#	D#	C#	Fm	G#m		4B	Ab
D#	A#	G#	Cm	D#m		5B	Eb
A#	F	D#	Gm	A#m		6B	Bb
F	C	A#	Dm	Fm		7B	F
Cm	Gm	Fm	D#	C		5A	cm
Gm	Dm	Cm	A#	G		6A	gm
Dm	Am	Gm	F	D		7A	dm
Am	Em	Dm	C	A		8A	am
Em	Bm	Am	G	E		9A	em
Bm	F#m	Em	D	B		10A	bm
F#m	C#m	Bm	A	F#		11A	f#m
C#m	G#m	F#m	E	C#		12A	c#m
G#m	D#m	C#m	B	G#		1A	g#m
D#m	A#m	G#m	F#	D#		2A	ebm
A#m	Fm	D#m	C#	A#		3A	bbm
Fm	Cm	A#m	G#	F		4A	fm

The following picture shows the Camelot Wheel - a modified version of the circle of fifths. DJs use it for harmonic mixing. If you mix the music by moving from one key to the next in clockwise, counter-clockwise or relative way, dissonances are prevented. It makes clear why the benchmarking is done in above described way.



(Camelot Wheel, taken from Harmonic Mixing - Mixed In Key [http://harmonic-mixing.com/HowTo.aspx]

III. Methodology

Data Preprocessing

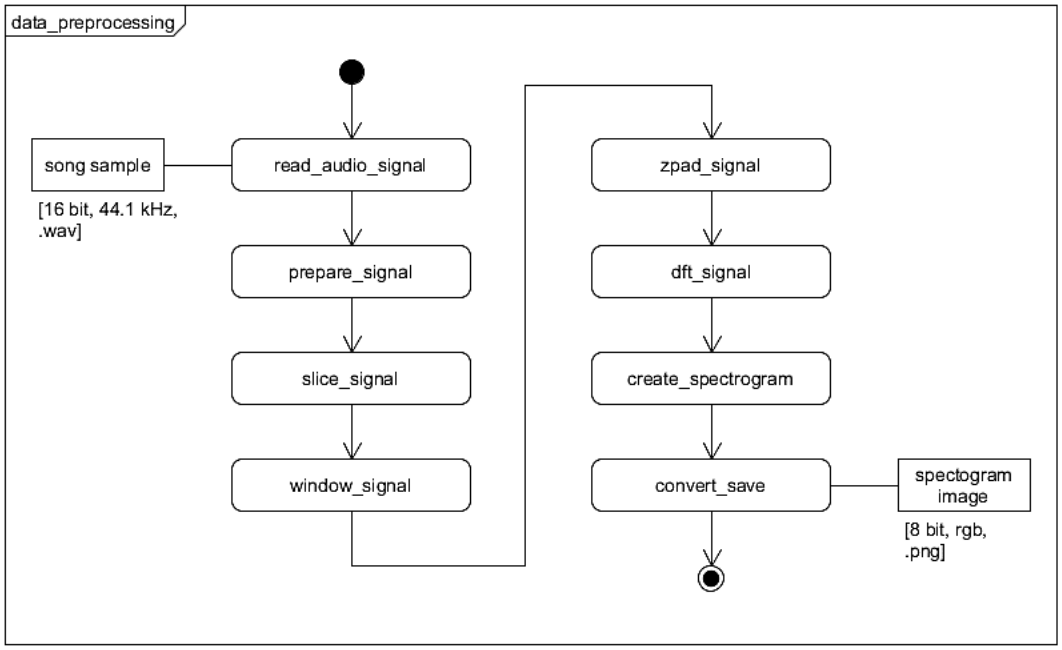
This section is divided in two parts:

- construction of the feature dataset (digital music pieces converted to spectrograms)
- data preprocessing of the feature dataset (spectrograms modified to fit the needs of the classifier)

Construction of the feature dataset

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:



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:

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

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:

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

Frequency bin distance of the Fourier transformed signal:

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

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

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

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 `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.

Data preprocessing of the feature dataset

Data splitting:

As mentioned in the section Data Exploration, there are 360 spectrograms of manually chosen digital song samples. Since benchmarking is done against an external software which analyses audio signals, the feature dataset is split up in a training set and testing set beforehand.

The training set contains 312 spectrograms for training and validation of the classifier. The testing set contains 72 spectrograms.

Data augmentation with the help of transposition:

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

There are 312 spectrograms in the training set and this is by far not enough data to train any neural network and therefor data augmentation is used. Since the position of the height pixels in the spectrogram are very important for the analysis, you can't just vary the images by scaling, rotation and translation since it would lead to a change in the key of the song. Fortunately, there exists a technique in music theory which comes in handy: Transposition is a process of moving a collection of notes up or down in pitch by a constant interval.

Due to the twelve-tone equal temperament, transposing a pitch x by n semitones is given by a linear equation using modulo 12, known as transpositional equivalence [13]:

$$T_n(x) = x + n \bmod 12$$

Further important is the octave equivalency which states, that notes an octave apart are given the same name in the western system of music notation, they are musically equivalent. By that, any transposition by octaves will not change the key of the music piece [14].

This results in translating the height pixels of a spectrogram by a multiple of 12 pixels up or down to augment the data.

By that, the size of the training set was increased from 312 to 936 spectrograms (39 per tonic-mode pair)

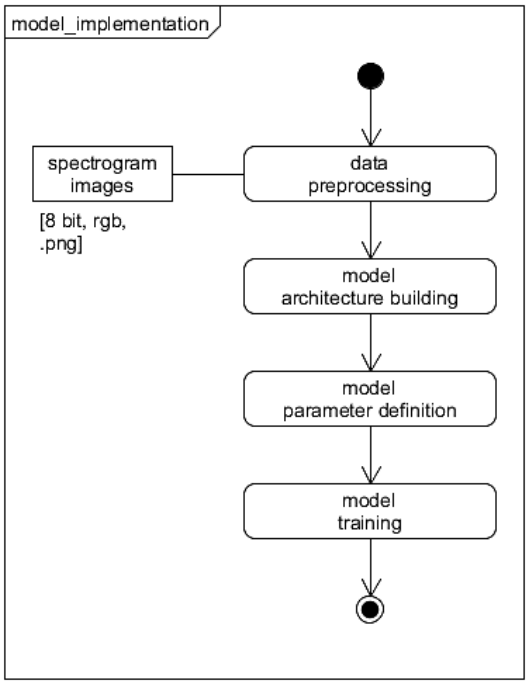
Image resizing:

CNNs work best with images that are divisible by 2 multiple times. The feature dataset images have a size of 128 x 150 pixels. They are resized to 160 x 160 pixels.

Implementation

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

The following outline shows the tasks of the implementation:



The feature dataset is loaded and converted into 4D tensors, suitable for a convolutional neural network built upon Keras with TensorFlow backend. Standardization of the feature dataset is not implemented (raw image values are used).

The model architecture is shown below:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input (InputLayer)           (None, 160, 160, 1)       0
_________________________________________________________________
fex.conv2d_1 (Conv2D)        (None, 160, 160, 64)      320
_________________________________________________________________
fex.maxp_1 (MaxPooling2D)    (None, 80, 80, 64)        0
_________________________________________________________________
fex.conv2d_2 (Conv2D)        (None, 80, 80, 128)       32896
_________________________________________________________________
fex.maxp_2 (MaxPooling2D)    (None, 40, 40, 128)       0
_________________________________________________________________
fex.conv2d_3 (Conv2D)        (None, 40, 40, 256)       131328
_________________________________________________________________
fex.maxp_3 (MaxPooling2D)    (None, 20, 20, 256)       0
_________________________________________________________________
avg_flatten (GlobalAveragePo (None, 256)               0
_________________________________________________________________
dropout_1 (Dropout)          (None, 256)               0
_________________________________________________________________
output (Dense)               (None, 24)                6168
=================================================================
Total params: 170,712
Trainable params: 170,712
Non-trainable params: 0
_________________________________________________________________
```

There are three sets of convolutional layers with max pooling layers and a global average pooling layer. Right before the fully connected output layer is a dropout as kind of a regularizer with a drop probability of 25%.

The filters per convolutional layer increase from 64 to 256 in the last layer. The max pooling layers always downsize the image by a factor of 2. 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 categorial cross-entropy. A stochastic gradient descent optimizer is used, with a learning rate = 0.1 and a momentum = 0.3.

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

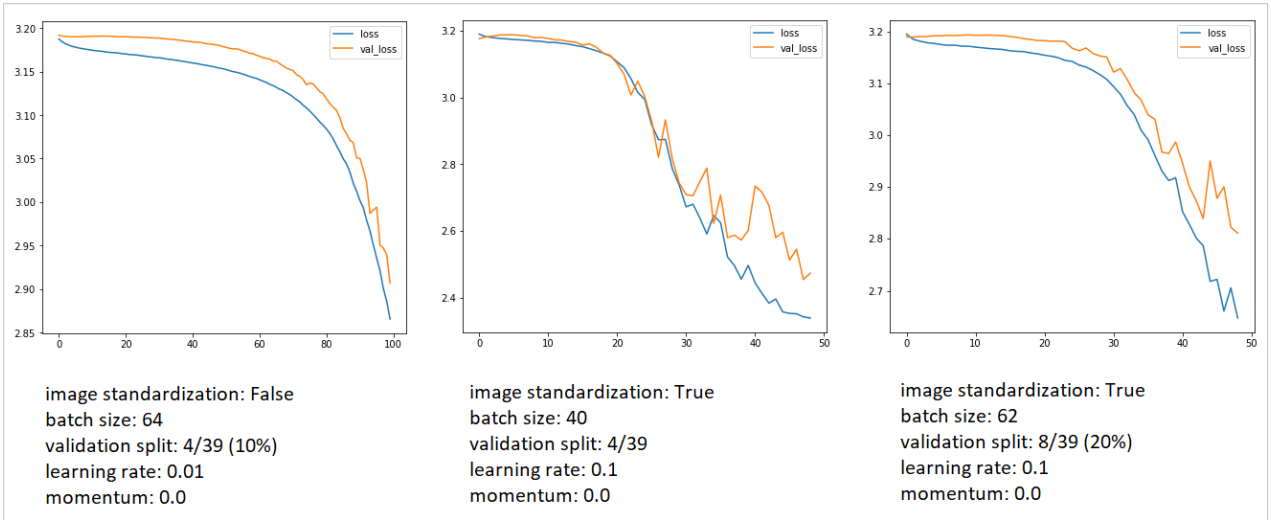
Model training is done in 450 epochs and a batch size of 44. The model weights are saved, if the validation step shows improvement.

Refinement

Data augmentation

The most important improvement has been made through data augmentation. The quantity of spectrogram images massively increased by a factor of 3 via translating the spectrogram image pixels up and down (octave transposition). Hence the dataset was big enough to finally let the classifier learn.

Image standardization, learning rate, batch size, validation split



Above graphs show 50/100 training epochs of the classifier with 3 different parameter settings. Conspicuous is when applying image standardization by scaling each image to have zero mean and unit norm (stdev = 1), the learning rate had to be 10 times higher (0.01 > 0.1) to get the same results. Furthermore, a smaller batch size works better since training and validation loss stay closer together and the delta of loss per epoch increased faster. Changing the size of the validation set didn't show any noticeable changes. To smoothen the loss curves, momentum has been applied in the final optimizer of the classifier.

The final parameters were set to: image standardization = True, batch size = 44, validation split = 6/39, learning rate = 0.1, momentum = 0.3.

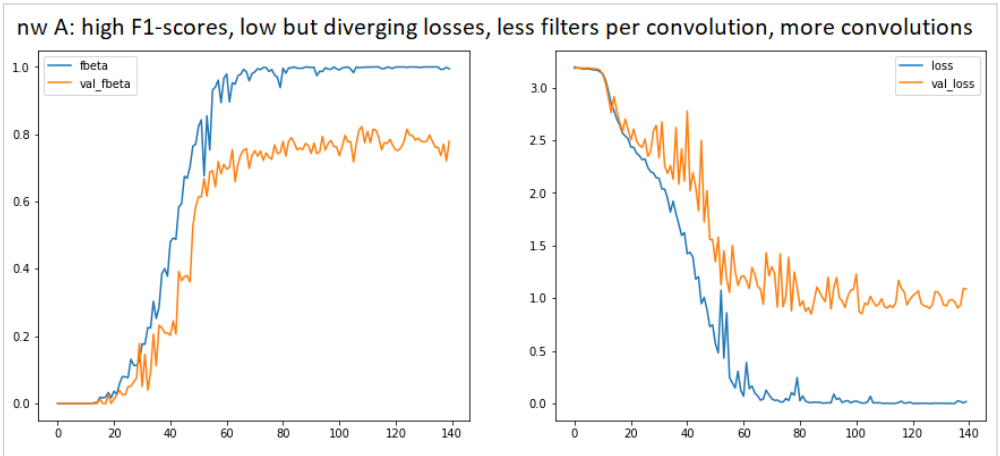
Model architecture

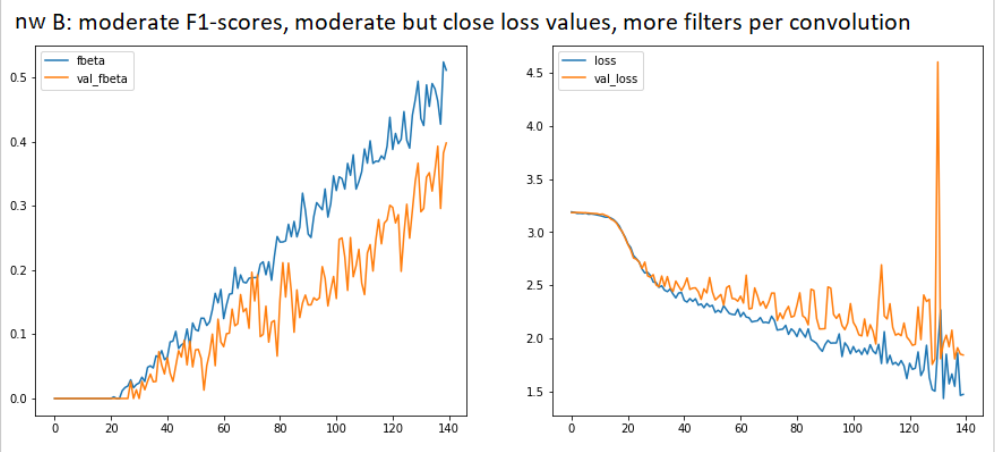
The first network feature extractors built began with 16 or 32 filters in the first layer and ended up on 128 filters in the last. While these were pretty good in learning and gained high F-beta scores, they began to overfit because the difference between training and validation loss increased. The evaluation of the overfitting models with the testing dataset confirmed it.

Even if you let network A have a feature extractor architecture
conv32+maxp - conv64+maxp - conv128+maxp - conv256+maxp, with 170,712 params in total

and network B have a feature extractor architecture
conv64+maxp - conv128+maxp - conv256+maxp, with 178,808 params in total,

network B always performed better, but with lower F-beta scores.

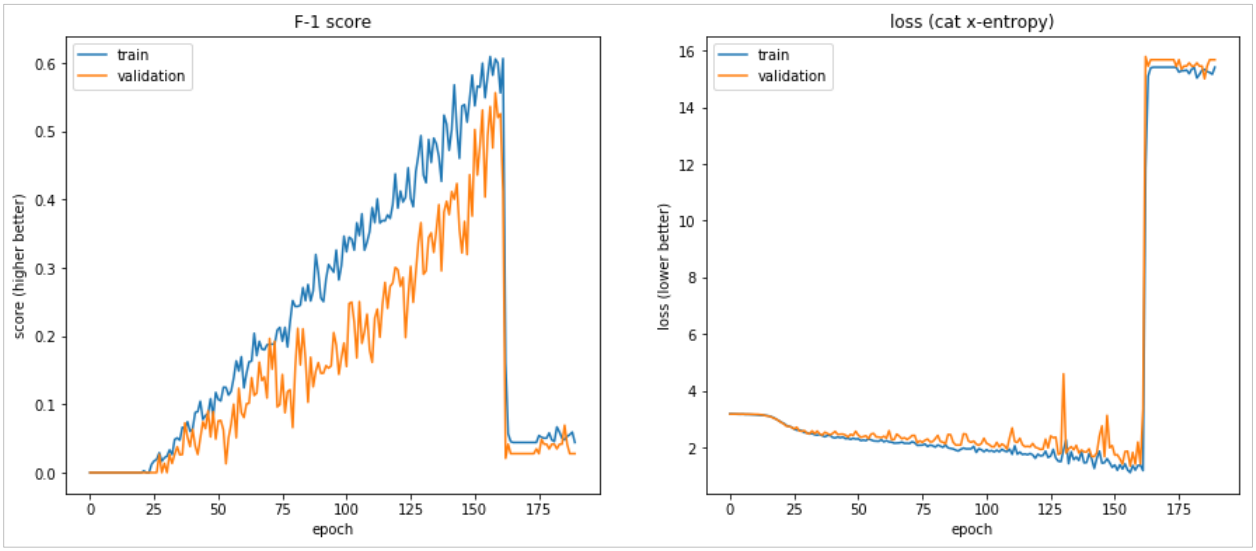




IV. Results

Model Evaluation and Validation

Below graph shows the progress of F-beta score and loss over the first 190 epochs:



The classifier learned features and the training / validation curves for both F-1 score and loss stay close together. The network didn't do anything for the first 20 epochs with marginal changes in loss.

Then the classifier forgets everything as of epoch 163. This behavior is reproducible, yet limitable to the feature extractor.

Score & loss value of epoch with best validation error:

```
 val_loss :  1.3516860637399886
     loss :  1.1193079418606229
val_fbeta :  0.5365167260169983
    fbeta :  0.6096924808290269
```

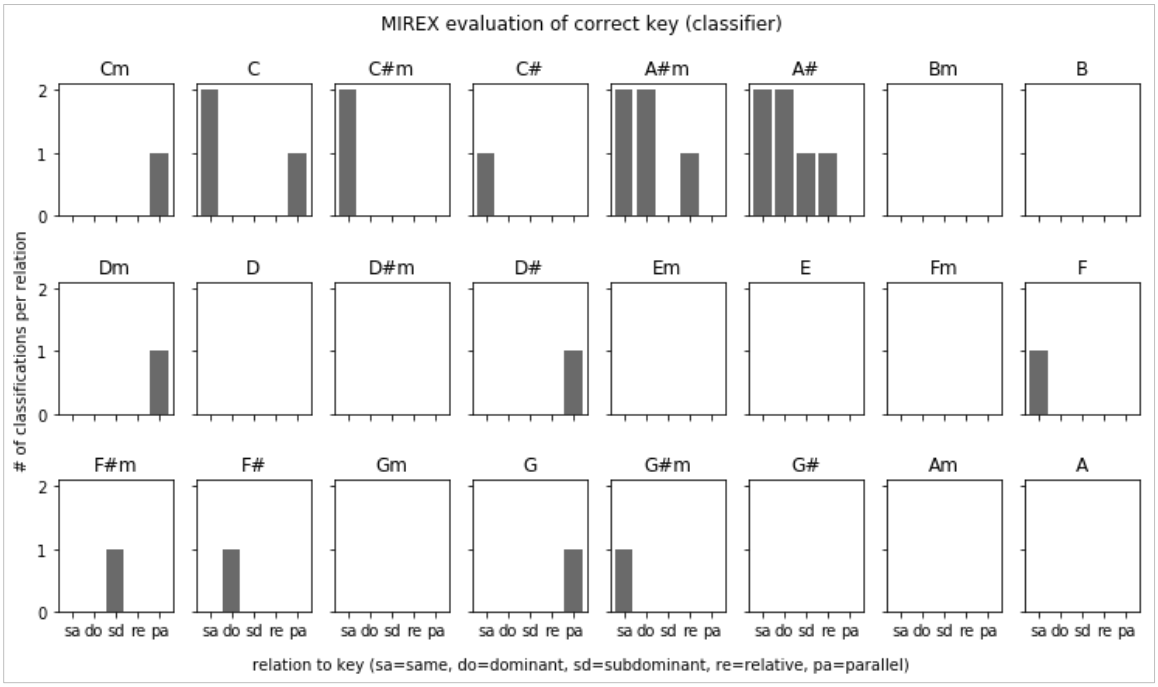
The evaluation of the classifier with 72 samples gave the following results:

```
     loss :  4.199851724836561
    fbeta :  0.145185194648268
```

Justification

The classifier was tested against the software KeyFinder with the MIREX evaluation procedure. Points are given with respect to the relation to the correct key.

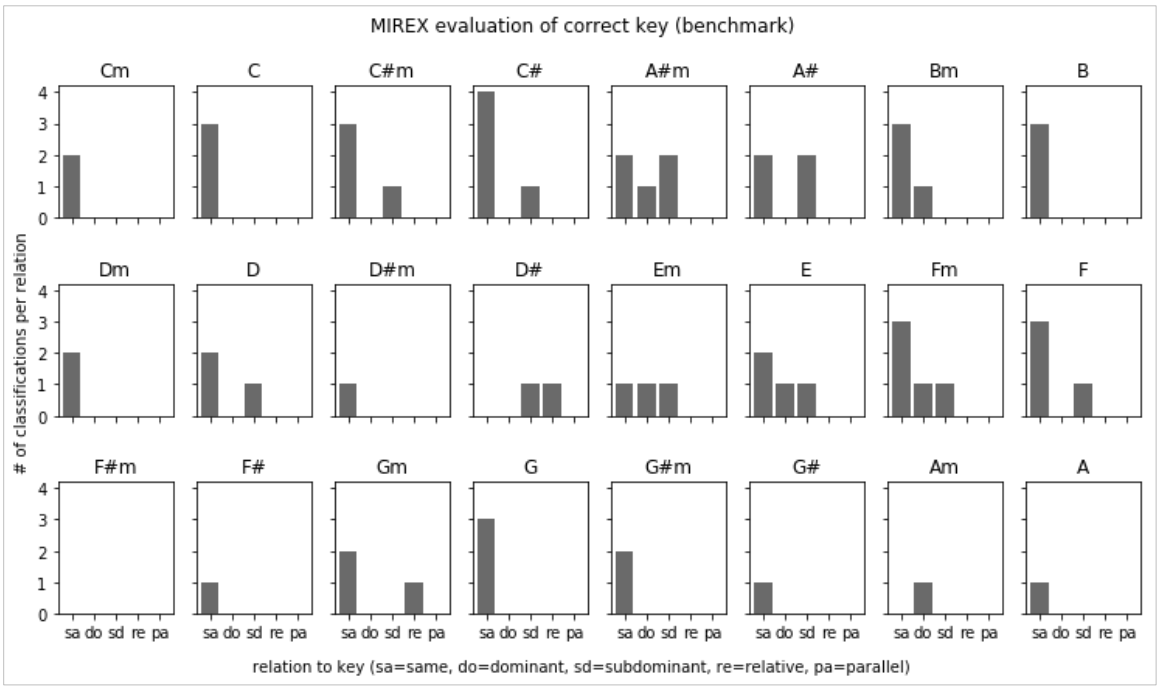
Classifier results:



The classifier reached best results for keys A#, A#m, C and C#m. For 11 keys there's no match at all.

Classifier key points: 16.1

Benchmark results:



Benchmark key points: 55.6

Out of 72 samples in the test dataset, the classifier got 16.1 MIREX key points and the benchmark got 55.6 MIREX key points. The classifier estimated 11 samples correctly and 7 in distance to the key, whereas the benchmark computed 46 samples correctly and 18 in distance to the key - that is 4 times more than the classifier.

By that, the classifier did not solve the learning task to estimate the audio key of a digital 30 second sample of a western music piece.

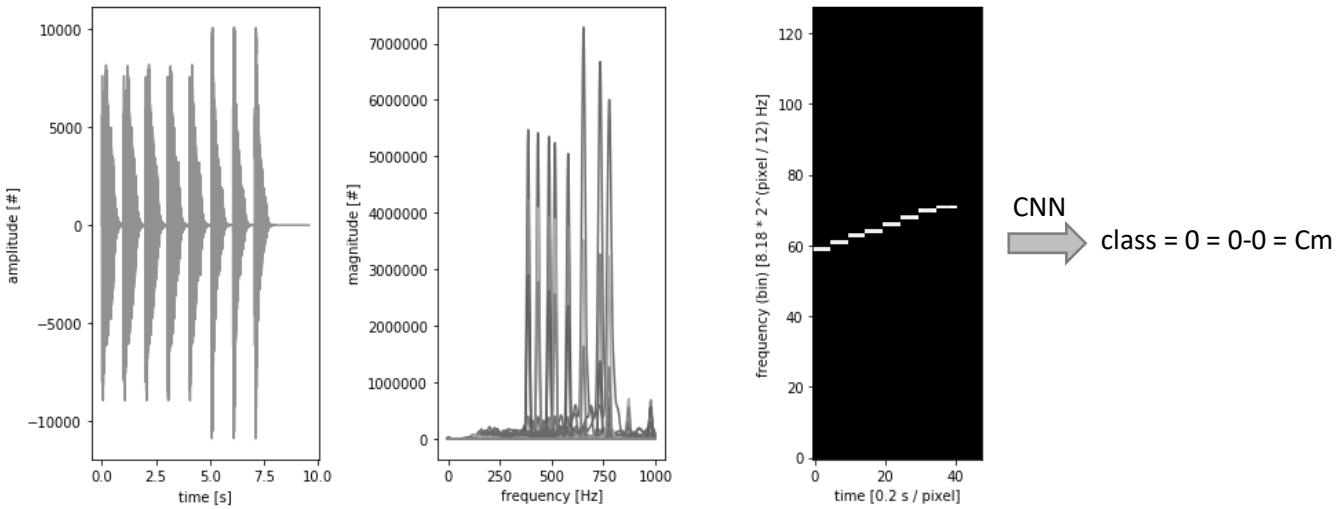
V. Conclusion

Free-Form Visualization

Following graphs compactly visualize the whole learning task beginning from the time-domain signal, via the frequency-domain signal and then the STFT spectrogram which is fed into the classifier giving the resulting tonic-mode class.

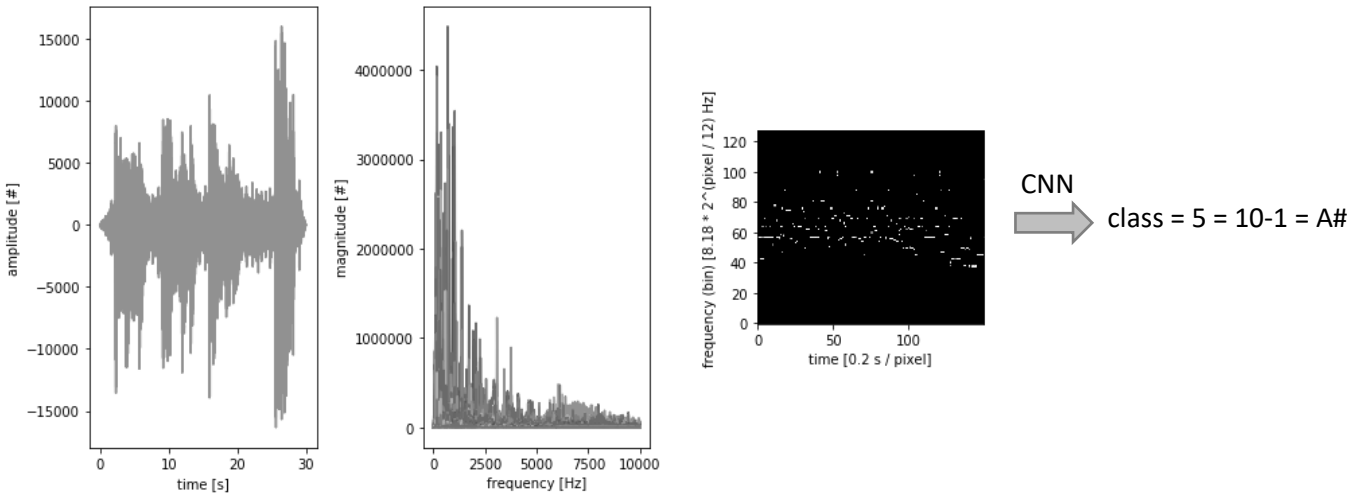
As simple example, the C major scale is used again since it is comprehensible by humans.

C major scale (true class = 1 = 0-1 = C):



The second example shows a music piece used in the learning task.

30 second song sample (true class = 11 = 3-1 = D#):



The difference in both frequency-domain signal and spectrogram of the examples reflects how challenging the learning task is due to an unbalanced frequency band and noise in the frequencies.

Reflection

The whole project to estimate the audio key of a digital 30 second sample of a western music piece by a CNN must be divided into several subtasks and is challenging in many ways.

The data preprocessing is the most important subtask, makes up 80% of the overall project work and is crucial for success or failure. Hereby the data retrieval was heavily time-consuming - it turned out that the MSD is the only available, hurdle-free and appropriate source for this learning task. On the other hand, it was very exciting to combine mathematical foundations, especially FFT, with music theory to create a dataset for a machine learning algorithm. During this project, the knowledge in both fields of competence rose in short time from almost zero to a solid base.

The biggest difficulty for me is represented by the evaluation of the model training to improve the learning algorithm. It is known that the theory of neural networks is behind their practical usage - many behaviors can't be described yet. Therefor a lot of experience is needed, and analytical skills must be applied to make proper decisions for the next model training. It was emphasized with pressure in your mind caused by the statement "The next training takes at least 6 hours, so wisely decide what to change for the next run".

All in all, a very exciting project keeping me busy all day and sometimes night. I am going on to improve the learning algorithm to make it solve the task one day.

Improvement

The data preprocessing subtask is the most important and needs to be improved to make the learning task a success.

The first thing to notice is the size of the input dataset - 312 song samples for training and validation is by far not enough. More songs need to be retrieved with the help of the MSD.

A second and widely used technique to increase the size of the feature dataset is data augmentation. Here translation in height was applied only, due to restrictions by the music theory. Other transformational functions like change in brightness and contrast can be applied too as both augmentation and filter technique to increase the quality of the spectrograms.

Furthermore, a quick look at a random spectrogram shows kind of chaotic information - as a human being it is hard to tell if there's any structure behind each tonic-mode pair and this may apply to the CNN too. Although the STFT is widely used and fast implementations exist, other algorithms may be more useful to create the spectrograms, amongst others the wavelet transform and multiresolution analysis.

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

[8] https://en.wikipedia.org/wiki/Scientific_pitch_notation#Table_of_note_frequencies

[9] https://en.wikipedia.org/wiki/Equal_temperament

[10] http://cs231n.github.io/convolutional-networks/#conv

[11] https://brohrer.github.io/how_convolutional_neural_networks_work.html

[12] http://www.ibrahimshaath.co.uk/keyfinder/

[13] https://en.wikipedia.org/wiki/Transposition_%28music%29#Transpositional_equivalence

[14] http://brebru.com/musicroom/theory/lesson18/octavetrans.html