Slovak University of Technology in Bratislava
Faculty of Informatics and Information Technologies

FIIT-5208-52638

Bc. Lukáš Marták

Modelling Music Structure using Artificial Neural Networks

Master's Thesis

Degree course: Information Systems
Field of study: 9.2.6 Information Systems
Place: Institute of Informatics, Information Systems and Software Engineering
Supervisor: Ing. Márius Šajgalík, PhD.

May 2017

ZADANIE

NAVRH ZADANIA

ČESTNÉ PREHLÁSENIE

Čestne prehlasujem, že záverečnú prácu som vypracoval samostatne s použitím uvedenej literatúry a na základe svojich vedomostí a znalostí.

.....

Bc. Lukáš Marták

POĎAKOVANIE

Predovšetkým by som sa chcel poďakovať všetkým, ktorí ma akokoľvek podporovali nie len počas práce na nielen tomto projekte, ale taktiež počas celej doby môjho štúdia.

Poďakovanie patrí taktiež fakulte za možnosť učiť sa a pracovať na projektoch v doméne, ktorá mi je možno bližšia ako iné.

Špeciálne poďakovanie[1] patrí najmä vedúcemu tejto práce, doktorovi Máriusovi Šajgalíkovi, za otvorenosť novým výzvam, trpezlivosť a edukatívny prístup pri nekončiacich konzultáciách. Taktiež by som rád poďakoval docentke Wande Benešovej za ochotne poskytnuté odborné diskusie, postrehy a pripomienky k mojej práci a jej príbuzným témam.

Na záver tiež patrí nemalá vďaka všetkým, ktorí zdieľali moje nadšenie pre tento projekt a živou diskusiou vo mne často podnecovali nové myšlienky a pohľady na vec.

[1] Táto časť je *stále* náchylná na exponenciálnu expanziu v prípade obhájenia tejto práce.

Anotácia

Slovenská technická univerzita v Bratislave
FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ
Študijný program: Informačné Systémy
Autor: Bc. Lukáš Marták
Diplomová práca: Modelovanie Hudobných Štruktúr pomocou Umelých Neurónových Sietí
Vedúci práce: Ing. Márius Šajgalík, PhD.
Máj 2017

S nástupom éry digitálnych technológií môžeme vidieť dramatický vývoj hudobného priemyslu spolu s radikálnym rastom množstva hudobného obsahu. Virtuálne knižnice sú plné hudby, pripravené podľa potreby poskytnúť komprimované, no stále veľmi kvalitné hudobné nahrávky. S tým ako bohatosť hudobného obsahu rastie, je dôležité mať nové metódy na opis tohto obsahu, navrhnuté pre rôzne účely použitia. Music Information Retrieval je interdisciplinárna veda o získavaní informácií z hudby. V rámci tejto výskumnej oblasti boli identifikované rozličné úlohy, s cieľom vyriešiť rôzne reálne problémy.

V tejto práci sa zameriame na úlohu automatického prepisu hudby, čo je proces získavania hudobnej notácie zo zvukového záznamu hudby. Hlavný problém, ktorý treba riešiť pri tejto úlohe, sa odborne volá Multiple Fundamental-Frequency Estimation, teda odhad viacnásobných základných frekvencií. V minulosti tento problém riešili experti z oblasti spracovania signálov s využitím doménových znalostí a čŕt šitých na mieru danému problému, pre extrakciu informácie zo signálu.

My sa na tento problém pozrieme z kontextu rozvíjajúcej sa oblasti strojového učenia, so zameraním na metódy hlbokého učenia. Aby sme mohli efektívne modelovať štruktúry hudobného obsahu v audio signále, potrebujeme najskôr vybudovať architektúru hlbokej neurónovej siete a potom ju optimalizovať tak, aby nadobudla dostatočnú kapacitu pre modelovanie hudobných signálov.

Annotation

Slovak University of Technology Bratislava
FACULTY OF INFORMATICS AND INFORMATION TECHNOLOGIES
Degree Course: Informatics
Author: Bc. Lukáš Marták
Master thesis: Modelling Music Structure using Artificial Neural Networks
Supervisor: Ing. Márius Šajgalík, PhD.
May 2017

With the era of digital technologies, we can see dramatic evolution of music industry together with radical growth of music content. Libraries are crowded with music, ready to stream compressed, but still great quality audio tracks on demand. As the richness of music content grows, it is crucial to have new methods to describe this content, designed for various purposes. Music Information Retrieval is an interdisciplinary science of retrieving information from music. Various tasks have been identified within the field, which aim to solve different real-world problems.

In this work, we approach the task of Automatic Music Transcription, which is a process of retrieving musical notation from audio piece containing music recording. The main subproblem to be solved here is called Multiple Fundamental-Frequency Estimation. In the past, it has been approached mostly by signal processing domain experts, using handcrafted features to extract information from signal.

We approach this problem within the context of emerging field of machine learning, focusing on deep learning methods. To be able to effectively model the structure of musical content within audio signal, we need to build an architecture of deep neural network and optimize it to gain this modelling capacity.

Obsah

Zoznam obrázkov

1 Introduction

In order to set this work properly to the broader context, we first share some thoughts on how music content can be analyzed differently by various study fields.

We also state some emerging problems and challenges, which motivated some notable amount of research efforts in the field, while focusing on the ones which inspired this work.

1.1 Background

From the history perspective, music has been evolving within cultures, driven by inventions of new tools and instruments, to provide a man with new means of expressive communication.

While music theory captures some rules and dependencies that describe how musically reasonable patterns can be created, our perception of them seems to lie beyond rational reasoning. Some measurements have proven that humans respond to musical signals rather emotionally [56].

However, most of these inventions were probably driven by desire to communicate on various levels of abstraction. Considering hypothesis, that there is causal relationship between abstract thinking and intelligent reasoning, we could find evidence of human evolution, in the evolution of abstract communication channels. One such channel which remains actual and still evolves, is music.

1.2 Motivation

With the rise of digital era, we observe dramatic evolution of music industry together with radical growth of music content. Libraries are crowded with music, ready to stream compressed, but still great quality audio tracks on demand.

Music is often created by combining not only naturally created and recorded sounds, but rather digitally created ones[1]. As the richness of music content grows, it is crucial to have proper methods and means to describe this content, designed for various purposes.

Some of the problems to drive the invention of such purpose-oriented methods would be e.g. melody pitch tracking, harmonic content description or even automatic lyrics extraction. These would be fundamental for tasks such as genre-based categorization or similarity-based

[1] By *digitally created* sounds, we refer to oscillator based digital sound synthesis.

search for personalized music recommendation, generating necessary meta-data directly from music content.

Also, people who study music professionally (musicians, composers) mostly need to gain deep understanding of its various aspects. Yet, it often implies the need to learn and reproduce existing musical pieces. Therefore, they spend tremendous amounts of time and effort transcribing, analyzing, and reproducing musical content.

Although there is clearly an educational purpose to the repetition of these tasks, at some point, one would appreciate having a choice of skipping at least the routine work of notes transcription, as it is vastly time consuming and the educational gain is not always present or required.

We thus go for a challenging task, which in broader context fits into the discipline of Music Information Retrieval (MIR). Within this research area, we examine multiple tasks and approaches, however, with focus on task of Automatic Music Transcription (AMT). The remainder of this document is structured by content according to following description.

1.3 Document Structure

In Section 2, we analyze and discuss some relevant problems and approaches from broader domain of MIR. Next, in Section 3, we describe some common machine learning methods used within respective MIR works, including some theoretical background of relevant methods.

Since our work is focused on recently very popular and successful, specific kind of machine learning, namely deep learning with artificial neural networks, we dedicate whole Section 4 to its analysis. In this section, we state history and fundamental components of recent deep learning methods, which are preliminary to understand and further apply the method to any problem. In Section 5 we choose the problem that will approached in the scope of our work, outline its fundamental subproblems and define our priorities within given problem set.

We describe our method in Section 6 which is further divided according to two diverse approaches we examine. First, in 6.1 we describe our approach to modelling musical signals in spectral domain. Additionally, we examine novel approach to modelling music in time domain and describe our method in 6.2.

In Section 7, we describe our evaluation methodology and show various results of multiple experiments and evaluations performed with proposed methods.

In Section 8, the last one, we summarize the results of our work, conclude its contribution and discuss possible future directions of this research.

1.4 Terms and Abbreviations

MIR - Music Information Retrieval
DSP - Digital Signal Processing
DAW - Digital Audio Workstation
ACE - Automatic Chord Estimation
AMT - Automatic Music Transcription
MIREX - Music Information Retrieval Evaluation eXchange
MIDI - Musical Instrument Digital Interface
F0 - Fundamental Frequency
ADSR - Attack-Decay-Sustain-Release
FFT - Fast Fourier Transform
STFT - Short Time Fourier Transform
CQT - Constant-Q Transform
PCP - Pitch Class Profile
MFCC - Mel Frequency Cepstral Coefficient
RBM - Restricted Boltzmann Machine
DBN - Deep Belief Network
SVM - Support Vector Machine
HMM - Hidden Markov Model
EM - Expectation Maximization
ANN - Artificial Neural Network
MLP - Multi-Layer Perceptron
PCA - Principal Component Analysis
CNN - Convolutional Neural Network
RNN - Recurrent Neural Network
LSTM - Long Short Term Memory
GRU - Gated Recurrent Unit
ReLU - Rectified Linear Unit

Monophony - Single melodic line

Homophony - One dominant melodic line accompanied by chords

Polyphony - Two or more simultaneous lines of independent melody

Timbre - Sound color, tone color, tone quality

Nyquist Frequency - Maximum frequency that can be carried without distortion, given a fixed sampling rate

Backpropagation - Backward propagation of errors, explained in 4.2

2 Music Information Retrieval

The broad domain of Music Information Retrieval is analyzed in this section. Existing methods for different problems are examined, with focus on methods that extract musical information directly from audio. Open problems are identified and state-of-the art is reviewed. Alternative approaches to these problems are considered in the conclusion.

Music Information Retrieval (MIR) is the interdisciplinary science of retrieving information from music. It is a small field of research, but with many real-world applications. Some methods are common to multiple MIR tasks, so we list those tasks below for further reference.

1. Automatic chord estimation
2. Instrument recognition and audio track separation
3. Automatic music categorization
4. Automatic music transcription

Automatic chord estimation (ACE) consists of annotating audio track in the time domain by chords, typically with onset time, offset time and label for every frame, where single chord sounds. ACE systems are benchmarked in Music Information Retrieval Evaluation eXchange (MIREX) subtask, which measures their accuracy in terms of percentage of correctly identified frames on a set of songs for which the ground truth is known [61].

Audio track separation is about recognizing all various instruments used in a recording, and further separating the audio into multiple tracks, one track per instrument. It is used to extract vocals when generating karaoke tracks from original audio tracks. Though, existing techniques still fail to clearly separate tracks with overlapping frequencies.

When it comes to automatic music categorization, ACE is often used before extraction of abstract properties from musical piece. Although chord progressions are mostly used when studying harmonic content of a piece, they can further be used for automatic key and mood detection, genre classification, audio-to-lyrics alignment, or to measure similarity for cover song identification [61]. But there are other methods for feature extraction, which are preliminary for music categorization systems, that enable categorization based on properties defined by these features, such as handcrafted features, machine learning methods or combination thereof [64].

Automatic music transcription (AMT) is task of annotating an audio track with some symbolic musical notation[1]. Most music transcription systems use Musical Instrument Digital Interface (MIDI) or MusicXML formats for score notation [83]. It is a task, which includes some non-trivial sub-tasks. To mention some of them, there is onset and offset detection (which together form note duration estimation), multi-pitch detection, instrument identification and extraction of rhythmic and tempo information. Difficulty of this task grows with number of instruments and degree of polyphony[2].

A problem of multi-pitch detection is also often denoted as multiple F0 estimation and is major subtask of polyphonic music transcription. Here, F0 stands for fundamental frequency, which in terms of harmonic components represents the 1st harmonic component of the sound. All following harmonic components are called overtones, while 2nd harmonic is 1st overtone, 3rd harmonic is 2nd overtone and so on. In terms of musical sounds, fundamental frequency is the original frequency of the musical tone sounding. Thus, musical notes identification basically consists in correctly estimating all fundamental frequencies contained in the signal.

2.1 Preprocessing of Audio Signal

For different tasks, different audio processing techniques and feature descriptors have been developed. In this subsection the vastly used ones are mentioned.

2.1.1 Fourier Transform

Researchers found evidence that human auditory system performs a transform from time to frequency domain [27]. Consequently, first step of audio processing pipeline in most feature extraction algorithms is transformation of raw audio signal into frequency domain [61]. This is mostly done by some variant of Fast Fourier Transform [26] which is an efficient algorithm for calculation of Discrete Fourier Transform of a time-based audio sequence.

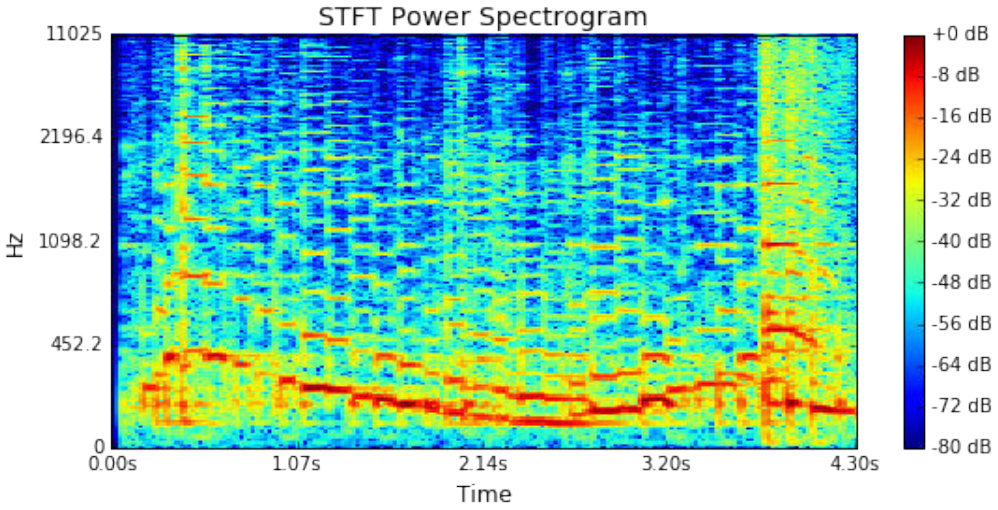
In order to preserve the timing information of the frequency spectral content, Short Time Fourier Transform (STFT) is taken by sliding window over the audio signal. Calculation of STFT is parametrized by the window length in terms of number of samples, and consists of taking the dot product of input signal with the Fourier matrix, which describes all different

[1] Sometimes chord recognition is denoted a subtask of AMT, because chord annotation is only higher-level form of symbolic notation.

[2] Number of simultaneous melodic lines in polyphony.

frequencies containable in a window of specified length. Thanks to special self-similar structure of Fourier matrix, this calculation can be done in $\mathcal{O}(n \log n)$ time complexity.

Results of STFT calculation over time frame of audio signal are stored into column of spectrogram matrix, where each column stores spectral magnitudes for given time frame while rows represent different frequency bands.



Obr. 1: *Example STFT spectrogram of audio signal*

STFT has been successfully used for time-frequency analysis in Automatic Music Transcription [16, 86, 59] and Automatic Chord Estimation [61, 78, 85] tasks. However, there is still issue with precision when describing the audio frequency content. This is because STFT algorithm uses fixed-length window for signal analysis.

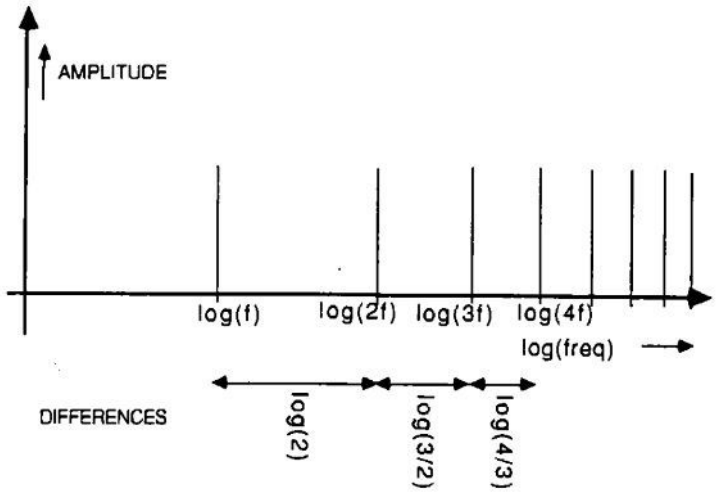
The lower bound of frequency resolution is defined by lowest frequency wavelength which yet fits into the window. Since the sliding window is moving with fixed size overlap, this parameter also defines the temporal resolution of STFT. Therefore, setting length of the window involves trading off resolutions between frequency and time [61].

2.1.2 Constant-Q Transform

This issue has been addressed by researchers at the Center for Computer Research in Music Acoustics (CCRMA) at Stanford, with so called Bounded-Q Transform [46]. This method

introduced variable frequency and time resolution, while maintaining the computational speed of FFT.

Few years later, Constant-Q Transform (CQT) was proposed, having advantage of calculation simplicity and sufficient time-frequency resolution for music analysis. Here Q stands for Q-factors (ratios of center frequencies to bandwidths) of all spectral bins being equal. It means there is frequency-dependent window length. It is therefore appropriate for note identification in music analysis, due to operating in logarithmic frequency scale, where sounds with harmonic frequency content give rise to so called *constant pattern* [45].



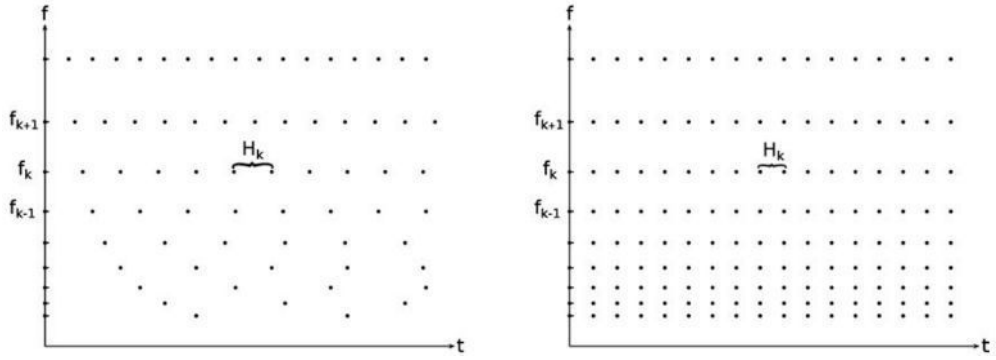
Obr. 2: *Constant pattern of harmonic frequencies; reprinted from [45].*

Using this knowledge, problem of fundamental frequency (F0) identification could be reduced to a problem of recognizing previously determined pattern [45]. But difficulty of this problem also grows with the degree of polyphony, since harmonic frequencies can interfere with fundamental frequencies. Even though, J. Brown proposed an elegant solution to fundamental frequency identification, by means of Constant-Q Transform and simple pattern recognition algorithm.

Although CQT improves the time resolution while maintaining the frequency resolution, due to constant sampling grid[1] for each frequency, there are many overlaps at low frequency windows, causing lots of redundant computations.

[1]Constant sampling grid means that stride of sliding window is equal between frequencies.

This redundancy can be solved by allowing adaptive resolution in frequency, as stated on the left plot from the figure below.



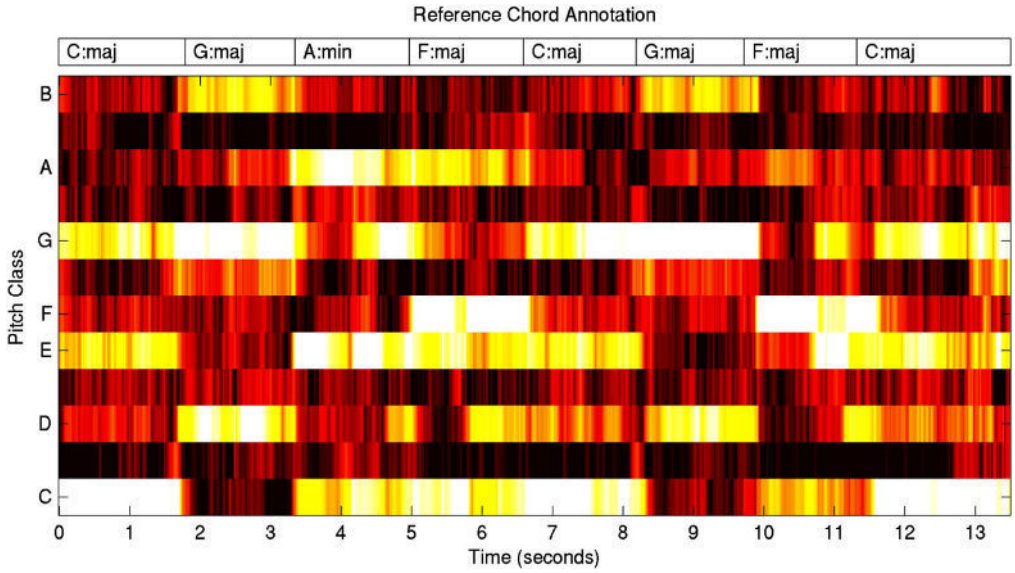
Obr. 3: *Minimum redundancy CQT vs. Rasterized CQT; reprinted from [77].*

This adaptation made it possible to construct an invertible non-stationary Gabor transform with a constant-Q factor on relevant frequency bins which allows modification of CQT-coefficients with subsequent re-synthesis [15]. This can be used in digital signal processing tasks, such as real-time frequency masking or key transposition.

2.1.3 Pitch Class Profile

Although spectrograms are mostly used for note identification in AMT, they are not optimal for ACE, since they still contain lot of information irrelevant for the task. Fortunately, Fujishima defined new feature representation called *Pitch Class Profile* [61].

Pitch Class Profile (PCP) is often visualized via chromagrams, where each pitch class represents one semi-tone of western musical scale, often referred to as *chroma feature*. Example is shown in figure below.



Obr. 4: *Chromagram of opening to* Let It Be (McCartney)*; reprinted from [61].*

It is created by removing redundant information from spectrogram feature, such as background noise, percussive elements of the music or harmonic frequencies. Frequencies which are close to each pitch class are collected and collapsed to form a 12–dimensional chroma vector for each time frame. That includes identification of salient frequencies for pitch class salience calculation, summing energy of pitch classes over octaves and smoothing which is also often stated as beat synchronization [61].

2.1.4 MFCC

Another feature representation of audio signal called Mel Frequency Cepstral Coefficients (MFCCs) was designed to accurately describe the shape of a vocal tract.

The basic idea is to compute a frequency analysis based upon a filter bank with approximately critical band spacing of the filters and bandwidths [69].

MFCC features are designed to describe envelope of short time power spectrum and have been successfully applied in Automatic Speech Recognition (ASR) tasks, e.g. keyword spotting [68]. In addition, it has also been successfully applied in MIR for music annotation and classification tasks [64].

2.2 Previous Work

The motivation for MIR research is big. Even human performance in this tasks is − in terms of accuracy − constrained by experience, training and musical talent. Therefore, human resources are very valuable and limited. Various approaches have been tried in order to develop new or improve the accuracy of existing solutions to different MIR tasks. We briefly describe some of them. Several machine learning approaches to MIR tasks are mentioned as well, so for some overview description of those methods please refer to Section 3.

2.2.1 Polyphonic Music Transcription

Different analytical approaches have been developed to address the problem of note identification in polyphonic music. Early methods for note identification and source separation consisted mostly in acoustic analysis based on expert domain knowledge [23, 24]. Although they were limited in terms of practical application, they served very well as a building blocks for future development of music analysis systems.

One of such systems, meant to recognize rhythm, chords and source-separated musical notes, used complex hierarchy of components to analyze the musical scene. Markov Random Field (MRF)-based hypothesis network was compared to Bayesian Network (BN), while system also employed computationally intensive simulated annealing algorithm, for edge-detection in MRF-based signal modelling [48]. MRF-based system outperformed BN-based system in terms of note recognition rates by $\approx 10\%$.

Another heuristic-based system used knowledge about auditory physiology, physical sound production and musical practice [59]. This system is very limited in many different terms. Some of those limitations are, that it can only transcribe piano music, it can't recognize multiple tones on different octaves and note hypothesis rating function has problems identifying higher notes in piano ranges.

One decade later, researchers started employing machine learning methods for AMT task [67, 66]. Generally, Support Vector Machines (SVM) were trained on spectral features to generate note presence hypothesis note-wise, meaning 87 hypothesis for 87 notes. These hypothesis were then treated as posterior probabilities[1] for further 'smoothing' by Hidden Markov Models (HMM).

[1] `http://www.investopedia.com/terms/p/posterior-probability.asp`

A completely different and novel approach to polyphonic note transcription comprised of designing genetic algorithm (GA) for MIDI generation from audio. Yet, this approach seem to be too computationally intensive, because for simple chord progression, it took over 1 hour to find the 100% accurate transcription [86].

Extensive work has also been done for the task of musical source identification in polyphonic music [13].

2.2.1.1 Polyphonic Piano Transcription

The problem of polyphonic music transcription can also occur in an form where only single sound source (musical instrument) appears musical audio signal. For instance, polyphonic piano transcription is a task of transcribing solo piano music. It thus belongs to the simpler subtasks of AMT, since no instrument identification is required. Juhan Nam and colleagues approached this problem with combination of several machine learning methods [63].

They first reviewed the task of multiple F0 estimation by listing 3 major state-of-the art approaches:

1. iterative F0 search [51],
2. joint source estimation [36],
3. classification-based approach.

They chose the use of classification-based approach, because it addresses polyphonic transcription in a completely different way $-$ as a pattern-recognition problem [63].

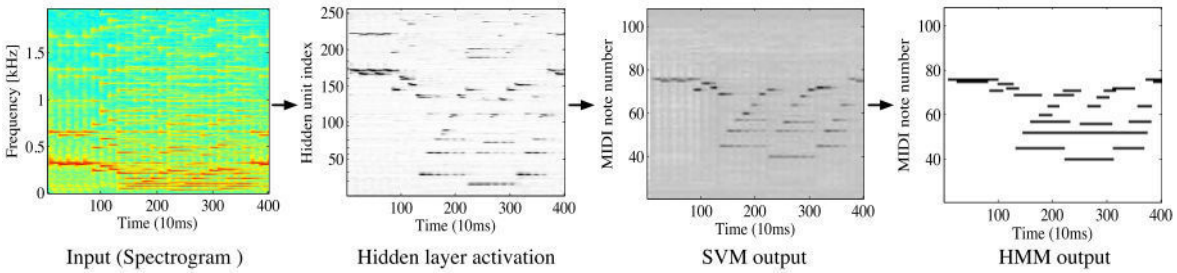
First, they applied unsupervised feature learning with DBNs to normalized, PCA-whitened spectrograms. This choice is reasoned by stating its previous successful applications to music classification tasks [37, 55]. They trained 2 layers of RBMs in a greedy layer-wise manner. This training comes in two phases which are often called *unsupervised pre-training* and *supervised fine-tuning*.

In second phase, they build upon Poliner and Ellis' piano transcription model, consisting of 88 independent SVM binary classifiers, predicting presence of corresponding 88 piano notes.

One improvement is, that rather than feeding spectrogram frames directly into SVM, they use features extracted by DBN instead. Another improvement is that instead of training single classifier at a time, they employ so called 'multi-note training', by feeding the features jointly

to all SVMs and treating their output as a single binary 88-dimensional vector. Thus, training SVMs as a single classifier.

In last phase, post-processing is done with Hidden Markov Model to *smooth* the SVM predictions, which were converted to posterior probabilities, before feeding to HMM.



Obr. 5: *Signal transformation through the processing pipeline; reprinted from [63].*

Figure above shows how signal is processed through the whole pipeline. To sum it all up: Spectrogram of audio frame is first calculated and fed directly into DBN. Activations on hidden layer of DBN are then fed into the set of SVM classifiers. Outputs from SVMs (representing distance to the boundary for each piano note) are converted to posterior probabilities and fed into HMM, which outputs the predicton of notes present in the signal.

Based on experiments with this setup on different datasets, authors conclude several things. First, fine-tuning generally improves accuracy. Second, multi-note training improved not only accuracy, but also training speed as well [63].

2.2.2 Music and Speech Categorization

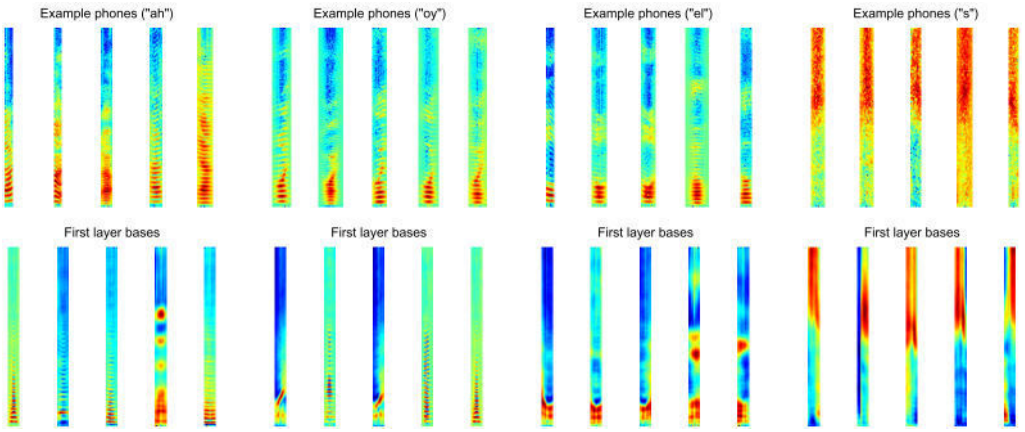
Another task where DBNs were used for feature extraction is music categorization [37]. Authors used activations of trained DBN as inputs to SVM classifier. Features were learned to solve the task of genre recognition on dataset which contained 10 different genres.

Overall classification accuracy was significantly better for features aggregated over 5-second frames than for frame-level features. Learned features also outperformed MFCCs in both cases. [37].

Researchers from Computer Science Department on Stanford University also decided to study deep learning approaches for auditory data [55]. In their work, authors applied

Convolutional Deep Belief Networks (CDBN) for unsupervised learning to extract relevant features from audio and consequently use them for various audio classification tasks.

To reduce high dimensionality of spectrograms, PCA whitening[1] was applied first. Unlabeled speech audio data was used to learn CDBN features. Empirical evaluation of CDBN features revealed, that for speech data, learned features closely correspond to phonemes, as is observable in the figure below.



Obr. 6: *Visualization of the four different phonemes and their corresponding first-layer CDBN bases. For each phoneme: the spectrograms of the five randomly selected phones; five first-layer bases with the highest average activations on the given phoneme; reprinted from [55].*

Further experiments showed, that CDBN features can easily outperform MFCC features in tasks such as speaker identification and speaker gender classification.

In addition, musical audio data was used for genre and artist identification. Features learned by CDBN on classical music data also outperformed MFCC features in both of these tasks [55].

2.2.3 Music Annotation

Juhan Nam and colleagues also tried various combinations of different feature learning algorithms with different classifiers for the task of music annotation [64].

They propose using Mel-frequency spectrogram instead of MFCC features as input to feature learning algorithm and use both these alternatives in experiments for comparison.

[1] `http://ufldl.stanford.edu/tutorial/unsupervised/PCAWhitening/`

Further, three different feature learning algorithms are described and employed for comparison: K-Means Clustering, Sparse Coding (SC) and Sparse Restricted Boltzmann Machine (SRBM).

For classification, Linear SVM is compared to neural network with single hidden layer.

They furthermore apply several different pre-processing and post-processing steps before and after feature learning, until features are fed to classifier. Detailed description of this processing pipeline is to be found in their work [64].

Different configurations of the setup have been evaluated on CAL500 dataset. For annotation and retrieval tasks, RSBM features learned from Mel-frequency spectrograms provided best results, so they were further used for classifiers evaluation. In this setup, neural network performed better than linear SVM in task of music retrieval [64].

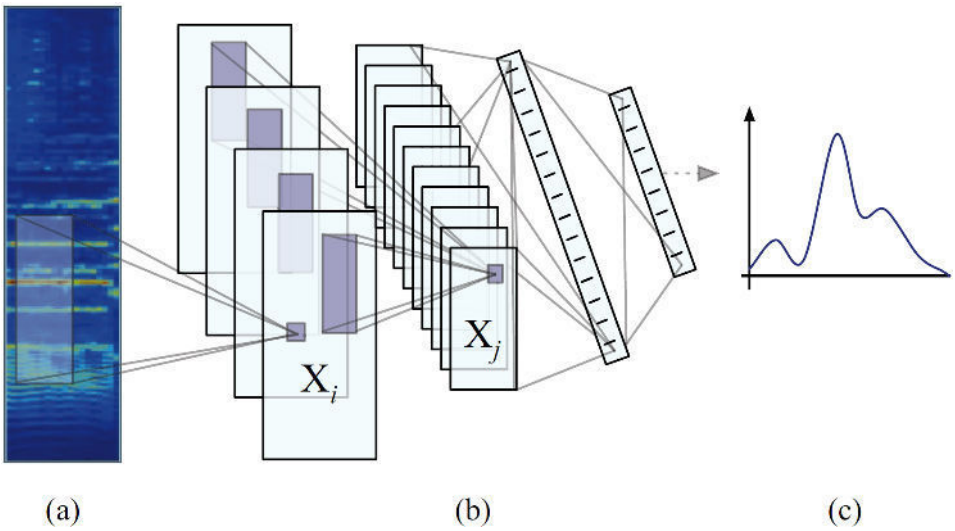
2.2.4 Automatic Chord Estimation

There has been an intensive ongoing research effort for the ACE task, initiated by the design of Pitch Class Profile [61]. First methods used template matching to identify chords in chromagrams. Some works relied purely on the abilities of HMMs to model temporal dependencies in musical chord progressions [72], others relied on handcrafted pitch-tracking and preprocessing of chroma features [85].

Development of expert-driven systems continued concurrently with data-driven ones, while in data-driven systems, Hidden Markov Models and Dynamic Bayesian Networks seem to be most frequently used.

Although, one alternative approach to ACE has been adopted by Humphrey and Bello in [40]. They state that major effort in previous ACE research was spent on tuning system components while developing better hand-crafted features, instead of developing system itself. They further point out that deep learning methods has already been employed to successfully produce robust Tonnetz features [41].

They mixed songs from several POP song datasets, used CQT for time-frequency transform and trained Convolutional Neural Networks (CNN) on 5 second tiles of pitch spectra to estimate an ongoing chord class, producing a jointly optimized chord recognition system.



Obr. 7: *The CNN Chord Recognition Architecture; reprinted from [40].*

In the figure above, spectrogram tile (a) is input to a CNN (b) which outputs probability distribution (c) between chord labels.

Using transposition of Constant-Q Gabor filtered representation and shifting the chord labels accordingly, they literally increased amount of data by factor of 12. They denote this data manipulation as extended training data (ETD). With such labeled data for supervised learning, their approach achieved state-of-the art performance.

2.2.5 Extraction of Instrumental Controls

Specific kind of information retrieval from music has been done in [21]. In this work, learning to extract violin instrumental controls from audio signal was performed with tree-based models and multilayer perceptrons.

However, this work required construction of database of multimodal data from violin performances. This database was constructed through acquisition of instrumental gestures from live performances of musical scores, monitored by expensive sensors in complex setups. That is also motivation to create system for indirect instrumental controls acquisition (by processing the audio signal).

Six different instrumental controls were monitored, namely, string being played, finger position on that string, bowing force, bowing velocity and bow tilt. Database contained

spectral features of audio recorded with special pickup and instrumental controls data.

With low-level spectral descriptors (pitch and MFCCs) fed to multilayer perceptron classifier seemed to give best prediction results. Although, automatic acquisition from any violin recording would be more difficult, due to specific recording conditions and violin causing different spectral properties of audio signal [21]. For such task, automatic calibration would be necessary.

2.3 Open Issues

We notice, that most approaches to different MIR tasks have problems with robustness against acoustic variations in musical signal, causing them poor accuracy when evaluated on real-world data. This issue is common to both hand-crafted expert-driven systems and data-driven systems.

In task of polyphonic music transcription, major problem is to correctly identify fundamental frequencies in a musical signal, thus to correctly identify notes in polyphonic texture. Whether those systems rely on hand-crafted features or learned ones, the filtering of harmonic components in musical sounds is still the most critical factor influencing accuracy. In worst case, the information necessary to distinguish fundamental frequencies between different sound sources is lost or damaged during conversion to representations being analyzed. Hopefully, and also more probably, this information is contained in the signal, but we still miss the ëncryption key"so far.

Therefore, one idea is to study the processes underlying human perception of music. This could inspire formation of mechanism that would realistically mimic the brain processes of musical expert when manually extracting musical information from audio.

Considering supervised learning approaches, their performance depends strongly on richness of the training data. Fortunately, thanks to vast libraries of musical instrument samples, audio data can be generated directly from MIDI notation. This is a huge advantage, though another fact is, that MIDI generated music lacks some kinds of acoustic variations which are specific for live played music.

Unlike polyphonic music transcription, for other tasks, such as music annotation or classification, the disadvantage is that we can't generate heaps of labeled data automatically. With song annotation, the problem is a bit easier, since expert knowledge is not necessary in most cases. Mostly, crowdsourcing methods and tools can be used to help us gather labels, as

demonstrated in [49, 52].

In task of automatic chord estimation, we observe that most systems in research efforts are evaluated on the same benchmark datasets, which could globally lead to overfitting [61]. The occurrence of different chords in music data also seems to correlate with their harmonic complexity. Adding more examples of complex chords and chord progressions in different musical and acoustic variations to music datasets could probably improve the accuracy of existing machine learning based ACE systems.

3 Machine Learning in Music Information Retrieval

In previous century, machine learning emerged as a new subfield of computer science. It studies construction of algorithms with ability to build a model of computational logic just through experience, i.e., instead of following strictly defined set of instructions, to learn the rules by observing data. It has been evolving for several decades now and currently it is used in vast majority of approaches in autonomous systems [88] and artificial intelligence generally.

We briefly review some machine learning methods which have been applied to problems in MIR. Since this work aims to explore the potential of neural networks applied to our domain, we rather analyze the progressive field of deep learning within dedicated Section 4.

3.1 Bayesian Networks

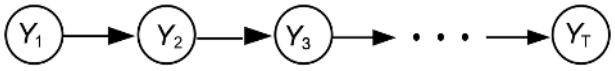
Bayesian networks are inferred from concept of Bayesian probability. It builds on Bayes' theorem, which provides inference method to compute posterior probability.

Posterior probability, also denoted as conditional probability, can be interpreted as probability of A conditioned by previous observation of B, or informally, probability P of A given that B was observed. Formula definition of posterior probability:

$$P(A \mid B) = \frac{P(B \mid A)\, P(A)}{P(B)}$$

Bayesian networks are used to model conditional independencies between set of random variables. Specifically, when represented by (acyclic) graph, each variable is a node in this graph, while arcs represent conditional independence.

Dynamic Bayesian networks are subclass of Bayesian networks for modelling time series data [34]. They are just Bayesian networks for dynamic processes. One of the simplest Bayesian network models is the first-order Markov process, where each variable is directly influenced only by previous variable [34].



Obr. 8: *Bayesian network representing first-order Markov process; reprinted from [34].*

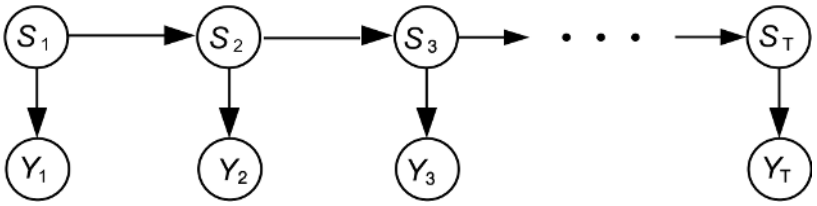
Bayesian networks have been employed for MIR tasks such as musical scene analysis [48] or music transcription [22, 35, 47]. Dynamic Bayesian networks have also been crucial in ACE system which achieved best performance on MIREX evaluation in 2012 [61].

3.2 Hidden Markov Models

A hidden Markov models are special kind of Bayesian networks. They are used to model probability distributions over sequences of observations, which are sampled at discrete, equally-spaced time intervals [34].

They are special by advantage that they can model *latent* or *hidden* dependencies of these observations. They gain this ability by modelling hidden states behind these observations.

In other words, hidden Markov model assumes that observation at time *t* was generated by some process in state S_t which is *hidden* from the observer [34]. It also assumes, that each state S_t is independent from all previous states except from S_{t-1}, which is called first order Markov property.



Obr. 9: *Bayesian network of conditional independence relations for first-order HMM; reprinted from [34].*

Higher order Markov models also exist, but they must satisfy corresponding order Markov property, which means, assuming that state S_t is only dependent on order-number of previous states, and that state at some time point encapsulates all we need to know about the history of the process in order to predict the future of the process [34].

Higher order HMMs can be used to model different properties of musical harmony. Example of such model could see chromagram as observation in time frame *t* with underlying latent states modelling key, chord and bass annotations for that time frame [61].

To learn the parameters of HMM, Expectation-Maximization (EM) algorithm is used in most applications. To learn the hidden parameters of HMM with EM, necessary expectations are computed using Forward-Backward algorithm [34].

In audio processing tasks such as ASR but also ACE, *Viterbi algorithm*[87] is very useful. It is used instead of forward-backward algorithm, to more effectively compute single most probable sequence of states.

Besides applications in ACE tasks [61, 78, 72], HMMs have been very useful in polyphonic music transcription systems for post-processing discrete classifier outputs. By performing temporal smoothing on these independent classifications, they have been able to significantly influence the transcription accuracy [67, 66, 63]. This motivated HMM tuning as a subject to further research interests.

3.3 Support Vector Machines

Support vector machines (SVMs) are supervised learning methods used for both classification and regression problems. Their training algorithm builds a model from set of labeled training data to classify new, previously unseen examples.

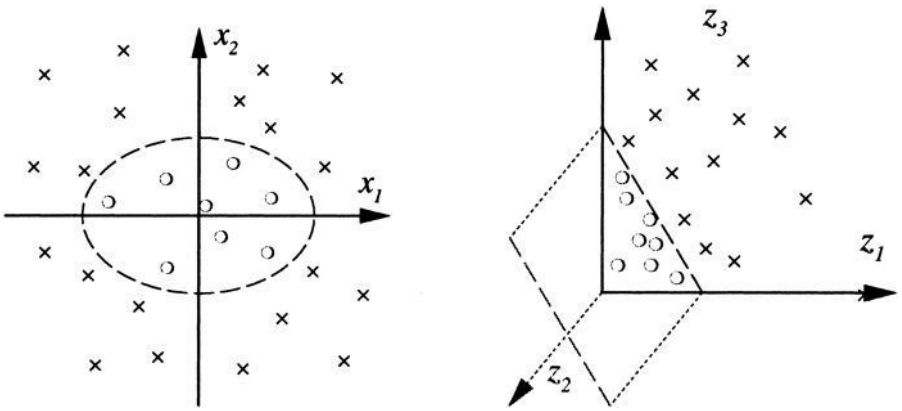
Originally, SVMs were developed to implement *statistical learning* theory, to estimate a function from set of examples, while minimizing *empirical risk* that estimated function will differ from the actual one [79]. Since it assumes data to be *linearly separable*, this type of SVM is referred to as linear SVM, being a non-probabilistic linear binary classifier.

Assuming that training set consists of *n*-dimensional feature vectors and is separable by a hyperplane in *n*-dimensional space, *margin* of a hyperplane is the minimal distance of training examples from *decision surface* represented by this hyperplane. Training SVM consists in finding hyperplane with maximal margin that correctly separates training examples. Choice of this hyperplane is supported by set of training examples, thereby called *support vectors* [79].

One key innovation associated with SVMs is the *kernel trick* [42]. Learning of linear SVM is driven by function $w^\top x + b$. SVM classifier discriminates between classes based on the sign of the reslut. It has been shown [42] that this function can be rewritten as:

$$b + \sum_i \alpha_i k(\mathbf{x}, \mathbf{x}^{(i)})$$

where α is vector of coefficients and $k(\mathbf{x}, \mathbf{x}^{(i)})$ is our *kernel* function. This function can be non-linear with respect to $\mathbf{x}$. Applying non-linear kernel function to input actually transforms it to a higher-dimensional space. When kernelized learning function is non-linear with respect to $\mathbf{x}$, it is learning in new non-linear feature space [42].



Obr. 10: *Transformation to higher dimensional feature space by kernelized SVN allowing the construction of separating hyperplane there; reprinted from [79].*

Intuitively, it can be thought of as training classifier on new, linearly separable representation of data, which were not linearly separable in original feature space.

SVM classifiers have been employed widely in MIR for tasks such as annotation [64], tagging and genre classification [37] and music transcription [67, 66, 63].

3.3.0.1 Conclusion

Although there are other machine learning algorithms, not mentioned in this section, we reviewed the most relevant ones, with respect to state-of-the art in MIR research.

Neural networks are of course as a machine learning method very relevant as well. In MIR tasks, their applications have been mostly classification for note detection [32, 58, 33, 63], chord recognition [65, 40] and genre/artist/key detection [55, 44, 37, 64]. Another meaningful and successful use of neural nets was for feature extraction with unsupervised learning [55, 37, 64]. Recently, some effort also led to success with purely audio-based neural network music generation [84].

4 Neural Networks and Deep Learning

The historical context of deep learning[1] is bound to development of computational models called Artificial Neural Networks (ANNs). Terminology in ANN research has been undergoing rich variations due to different perspectives of scientists from various disciplines contributing to the field. However, knowing this context is certainly useful to understand some causalities of contemporary state-of-the art.

Different learning algorithms and optimization techniques have been developed in order to improve the cognitive capabilities of neural network models. We examine existing architectures, their main characteristics and review the major progress that has been done in past research of artificial neural networks.

4.1 History

Few trends in history of deep learning have been identified [42]:

- Popularity of deep learning noticed significant fluctuation as only few people actually understood what was going on.
- Usability of deep learning increased with the amount of available training data.
- Deep models and their capabilities have been growing in size together with hardware and software infrastructure.
- Deep learning has been solving problems with growing complexity but too growing accuracy over time.

These trends are also observable in this brief selection from the major milestones of the deep learning research journey.

1943 - Warren McCulloch and Walter Pitts created a computational model for neural networks based on threshold logic.

1958 - Frank Rosenblatt introduced the perceptron.

1980 - Kunihiko Fukushima proposed the Neoconitron, a hierarchical, multilayered artificial neural network used for pattern recognition problems.

[1] Though the term *deep learning* has interdisciplinary scope such as cognitive sciences, computer science or neurobiology, we use it in the context of computer science and artificial intelligence.

1989 - Deep neural networks with training times measured in days, making them impractical for real-world use.

2006 - Geoffrey Hinton and Ruslan Salakhutdinov showed how many-layered neural network could be pre-trained layer-wise as unsupervised restricted Boltzmann machines.

2009 - NIPS Workshop on Deep Learning for Speech Recognition discovered that with a large enough data set, the neural networks don't need pre-training, and the error rates drop significantly.

2012 - Artificial pattern-recognition algorithms achieved human-level performance on certain tasks.

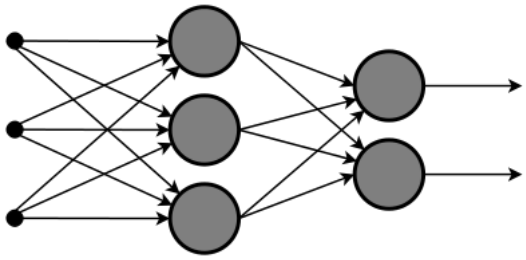
2016 - Google DeepMind's algorithm AlphaGo mastered the art of the complex board game Go and beats the professional Go player Lee Sedol.

4.2 Learning Algorithms

Machine learning algorithms are broadly categorized as *unsupervised* or *supervised* according to the way their training is realized [42]. Although, there are more subcategories of machine learning algorithms out there. To name a few, there are supervised, unsupervised, semi-supervised, multi-instance and reinforcement learning algorithms.

But first things first, the holy grail of neural networks learning process – backward propagation, is preliminary for understanding those algorithms. It is responsible for correct calculation of desired modifications on synaptic weights – the learned parameters of neural network. Learning algorithms differ mainly in the way they obtain measure of error, which is further used to update parameters during backpropagation.

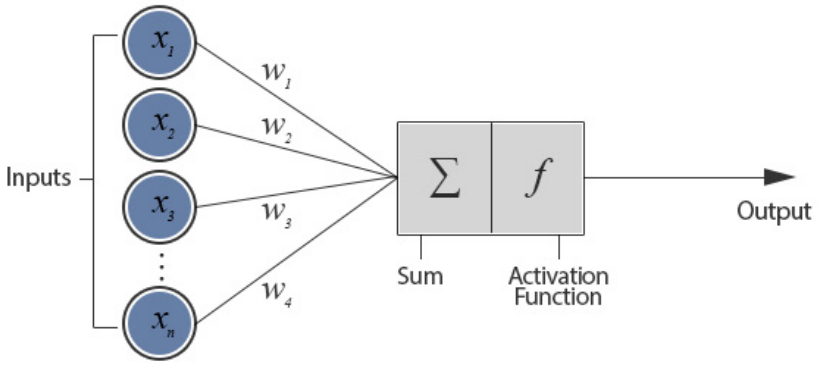
Conventional feedforward ANN consisting of multiple layers of neurons, each having synaptic connections to all neurons in the next layer, is called Multi-Layer Perceptron (MLP). Simple MLP model with 3 input neurons, 3 hidden neurons and 2 output neurons is shown in Figure 11. We further describe fundamentals of forward and backward propagation in terms of calculation steps.



Obr. 11: *Feedforward Neural Network; retrieved from [3].*

Forward propagation:

- Input neurons observe features of a data point x, which are in turn propagated forward to the next layer of neurons, weighted over synaptic weights w.
- Each neuron applies its activation function f to weighted sum of its inputs. Result of such calculation is called activation or output of the unit.
- Activations are used as inputs to next layer and the procedure is repeated until the output layer is reached.



Obr. 12: *Diagram of single neural processing unit activation; retrieved from [6].*

Backward propagation:

- When activations on output layer of neurons are calculated, they are used and measure of error is provided by the learning algorithm.

- Error is calculated according to specific *optimization* algorithm and *regularization* factors. It is typically further factorized by *learning rate*. These belong to set of *hyper-parameters*.
- Each synaptic weight is then updated[1] by its contribution to the error, which is calculated by reverse-mode differentiation[2] using chain rule for composite functions [1].

Similarly to forward propagation, backward propagation is performed layer-wise, since architecture of the network in terms of intra-layer connections defines mathematical dependencies between its parameters.

An important property to mention is the time complexity of backward propagation. It is at most a constant factor slower than the forward computation of the output.

4.2.1 Supervised Learning

Supervised learning is based on data containing input observations together with their corresponding output labels. One iteration of supervised learning cycle in ANN consists of whole forward-backward propagation.

Input observations are fed to the network, expected output labels are provided to the algorithm, error measure is calculated and propagated back through the network, updating weights (parameters) accordingly.

Supervised learning is used to train models for classification or regression tasks. But it is often accompanied with a problem called *overfitting*. Overfitting occurs when model is optimized too much on training data, and it is signalized by great performance on training data, but poor on new data examples. It basically means, that model learned too much about the training data chunk. Instead of just general properties of given type of data, it learned specifics of examples from the training set. An overfitted model fails to *generalize* on training data, thus instead of learning the rules, it memorizes properties of training examples.

In order to improve the learning process and prevent model from overfitting, it is important to use and search for optimal setting of *regularization* techniques. These are further explained in 4.3 along with importance of hyperparameters, such as *weights initialization*, *loss* function, *optimization* strategy, *learning rate decay* and others.

[1] Update is naturally performed in the direction, which would potentially lower the error contribution.
[2] Note that composite function must be *continuous*, to be *differentiable*, for this rule to be applicable.

4.2.2 Unsupervised Learning

Unsupervised learning is family of algorithms that learn from a dataset of examples with many features but with no use of labels. They learn to extract and interpret the properties of the data, which are most useful in determining data points identity. Therefore, they are sometimes denoted as *feature learning* or *representation learning* algorithms.

Some examples of unsupervised learning algorithms besides those neural-based would be K-means Clustering algorithm used to find groups of similar data examples, or Principal Component Analysis (PCA) used to extract most relevant features from data and thereby reduce the size of feature sets.

In the context of deep learning, we usually want to learn the entire probability distribution that generated a dataset, whether explicitly as in density estimation or implicitly for tasks like synthesis or denoising [42].

Since unsupervised and supervised learning are not completely distinct concepts, there are many machine learning models, which perform both tasks [42].

4.2.3 Reinforcement Learning

Reinforcement learning is very specific type of learning. It does not experience data examples from a fixed dataset, but rather through interactions with an environment instead. Network is learning based on its experience, gaining feedback from the environment to each interaction it has, thus, being able to evaluate utilization of each interaction.

It has been major contribution to the success of above mentioned AlpaGo algorithm designed by Google DeepMind. It can also be viewed as a way to represent non-continuous cost functions such as sets of rules as continuous thanks to reward function. This way, any non-differentiable function can be learned with reinforcement backpropagation, as authors demonstrate with music theory rules in [43].

4.3 Training techniques and strategies

The performance of neural networks proved to be very sensible to various settings and configurations of the model architecture, and even training algorithm. Therefore, instead of simple usual parameters, they have rather been called *hyperparameters*.

We first write down some common regularization techniques, which are used to prevent

the training from overfitting. They also add up to the set of hyperparameters since some of them have to be tuned too.

4.3.1 Regularization techniques

- Momentum − smoothes the gradient vector by number of previous values. Used to improve convergence of optimizer to the global minimum on the error surface.
- Early termination − stops training when error measured on validation data starts growing. Used to prevent from overfitting.
- L2-norm − penalizes large weights by adding normalized weight term to the loss function. Leads to better convergence.
- Dropout − randomly turns off some neurons. Makes the model create different redundant representations of the data, thus learn context, and prevents it from memorizing, what means overfitting.

All hyperparameters have generally influence on different aspects of model performance. Sometimes, previous work might indicate which settings are optimal for certain problems, but mostly they need to be fine-tuned by hand based on evaluation results, since optimal setup differs between various datasets and problems.

4.3.2 Hyperparameters

- Depth of the model − number of hidden layers of the model.
- Width of the model − number of neurons[1] in hidden layers. Too few causes underfitting, too much, instead overfitting.
- Set of activation functions − typically, activation functions are defined per layer of neurons. These choices are important in terms of mathematical modelling capacity of the network.
- Weight initialization − generally random distribution with zero mean and small, equal variance along whole distribution. Size of the variance depends.

[1] Neurons as a computational units of neural network are sometimes denoted as *units*.

- Optimization algorithm − optimizer, which chooses how to feed the model, measure error and apply it to minimize loss function. Choice is generally model-dependent and problem-dependent.
- Learning rate decay − velocity of adjustments made to the weights, representing how fast the model learns. Helps to find higher global optimum[1].
- Dropout probability − probability of discarding neuron's activation. Equally for all neurons where dropout is used.

Tuning of hyperparameters is a manual process driven by evaluation of model's performance on validation (development) dataset, based on human intuition and expert knowledge. It is fundamentally different from optimization of model *parameters* performed by optimizer in scope of training session, which is actual training of the network. Although this tuning is still an optimization in a sense, it is by one abstraction layer higher.

4.4 Architectures

One fundamental key consideration for design of neural networks is determining the architecture. The word architecture refers to the overall structure of the network: how many units it should have and how these units should be connected to each other [42].

There are several different architectures and models being proved and used for specific applications. Some architectures also contain well known architectural patterns, recognized for their particular function. We describe some of the well known network architectural patterns and modules.

4.4.1 Feedforward Models

The architecture of conventional deep feedforward network, often denoted as MLP, is pretty straightforward. It is based on the model of perceptron, introduced by Rosenblatt in [73]. A layer alone has no connections, while two adjacent layers are fully connected. Such layer of neurons, where each unit has connections to each unit of previous layer, is called *fully connected* layer.

[1] Similarly to principle of system cooling in simulated annealing algorithm.

The major advantage of MLP consists in utilization of its deep structure. It learns feautre representations of input data on multiple layers, while recognizing the relevant structures in the data and modelling relationship between input and output.

In practice, this architecture alone showed to have limited use, but is often combined with different architectures to form new, more complex ones.

4.4.2 Representation Learning Models

The goal of representation learning within neural networks is to find different *code* or *feature descriptor* for the data observation, to extract the most relevant features for given task.

In principle, the learning is performed by encoding (in terms of compression) input vector to a vector with possibly different density, and then forcing the reconstruction of the input from this representation. The distance between reconstructed data and original data is used for learning.

4.4.2.1 Restricted Boltzmann Machines

Boltzmann Machines (BM), inspired by Hopfield Networks (HN), have been preliminary to representation learning models, and introduced in [38]. Each neuron is connected to each other neuron, resulting in fully connected network. Training is initialized with random weights and performed through repetitive back and forth propagation until network reaches state of equilibrium. Activations are controlled by *global temperature* level.

Restricted Boltzmann Machines (RBM), introduced in are special variation of BMs. Main difference is, that they are restricted in sense of disabling connections between neurons within the same layer, so that only pairs of neurons, each from different layer, can have direct synaptic connection.

Special kind of deep architecture is Deep Belief Network (DBN) introduced in [18]. It combines unsupervised learning with supervised learning approach. It builds on layers of RBMs stacked one on another, pre-trains these layers in a greedy layer-wise unsupervised manner, and then attaches the classifier with fully connected layer on the top of the model. It fine-tunes the model by supervised training with backpropagation. Thus, model is using self-learned feature representations and supervised classifier.

4.4.2.2 Autoencoders

Autoencoders (AE) are roughly identical to MLP in terms of architecture, while main difference is in how they are trained and used. Their purpose is to automatically find parameters of an encoding decoding function, while in contrast to RBMs, parameters are not shared between those functions.

The idea is to compress the information into hidden layers which are smaller than input and output layer. They can be trained by backward propagating error as a distance of output from input activations.

Since they have been introduced [19], they inspired formation of different model variations, such as Sparse Autoencoders, Variational Autoencoders or Denoising Autoencoders, used mostly in different ways for various purposes [7].

4.4.3 Convolutional Models

Convolutional Neural Networks (CNNs) also called *ConvNets* are very special kind of neural networks. They are designed (but not restricted) to learn features from graphical content. They have been successful in tasks such as object recognition and image classification. In combination with other networks they were also able to generate image descriptions.

But digital representation of 2-dimensionally spaced image by 1-dimensional vector is not really suitable for a network, to understand the graphical content, since for most tasks, image locations of features to be recognized can be spaced differently amongst training data. Therefore, the new way of looking on the input was employed by introducing convolutional layers [53].

A convolutional layer, consisting of so called *filters*, is generated using convolution kernels, where values of the kernel are the trained parameters. By striding this kernel over input image, the filter values of next convolutional layer are calculated pixel-wise. These convolutions thus share their synaptic weights across the space.

Their major advantage from other models is, that their convolutional layers of neurons are immune to so called *translation invariance* of their inputs. In other words, they learn to recognize patterns no matter where they occur on the input.

This property borrows them new incredible abilities. By combining variously sized convolution filters, different modules have been created, that can learn to model graphical content on different levels of abstraction.

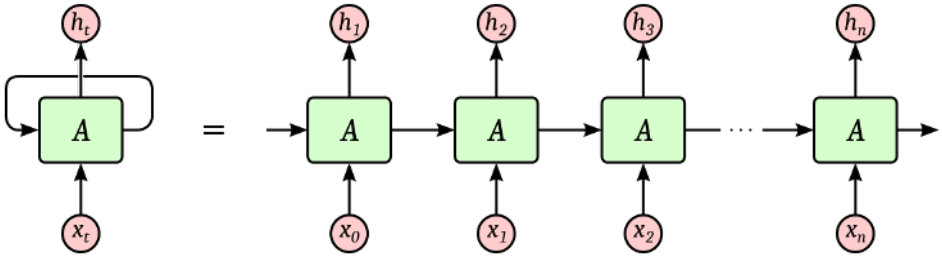
One such example are *inception modules*. They combine multiple layers of convolutions together to create one semantically rich layer capturing much more information than would be possible with single convolutional layer.

Thanks to their translational invariance, modern ConvNets are also vastly used for tasks besides computer vision, where this property is desirable.

4.4.4 Recurrent Models

Recurrent Neural Networks (RNNs) introduced in [29] are different from conventional ANNs in a way similar to CNNs. They are also deep models but instead of space, they are deep in time. Designed to deal with semantic and latent relationships between members of sequences in time series.

Their analogy to CNNs is actually pretty straightforward. RNNs also have shared their parameters, but only instead of space, they share them across time. The concept of sharing weights across time only means, to re-use them while processing each observation in a sequence.



Obr. 13: *Diagram of RNN cell unrolled in time; reprinted from [12].*

RNNs are designed to model sequences of inputs in time domain, so they contain a recurrent connection(s) from the past states to current. Although, while sharing weights over time, the error backpropagation also needs to get through time.

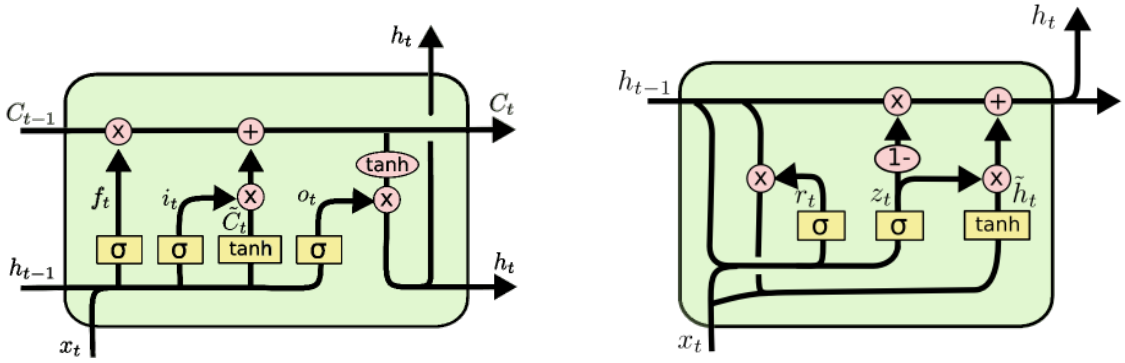
This means lots of little correlated updates to the same set of weights all at once. It causes problems with gradient, since due to strong correlations between these updates, either gradient explodes (goes to infinity) or vanishes (goes to zero).

To solve the problem of exploding gradient, simple hack called *gradient clipping* was employed. It only lets gradient grow to a certain defined maximum value.

However, vanishing gradient is a bit harder problem to solve. In sequence modelling it actually causes memory loss. To solve the problem of vanishing gradient, Long Short-Term Memory (LSTM) cell was introduced [39] as an alternative kernel of RNN.

It encapsulates the memory of recurrent unit with "protection"gates for read, write and forget operations. These gates filter the input signals, behaving as a single-unit layers of neural nets, having sets of weights and continuous activation functions, which makes them differentiable and trainable together with other parameters of the model through backpropagation.

One recent variation of LSTM cell is called Gated Recurrent Unit (GRU) cell. Internally it uses one update gate instead of separate input, output and forget gate. Therefore, GRU is slightly faster to train than LSTM, but also slightly less expressive. It has been shown, that in some extra cases, when expressiveness is not required, GRUs can outperform LSTMs [7, 25].



(a) LSTM cell; reprinted from [11].

(b) GRU cell; reprinted from [11].

Obr. 14: *Diagrams of most used recurrent unit architectures.*

Mathematical details if these internals, as depicted in Figures 14a and 14b are also explained by Chris Olah in his blog post [11].

4.4.5 General Considerations

4.4.5.1 Time Complexity

The computational complexity of neural network is an important characteristic. It depends strongly on composition of architecture and should be considered when design decisions are made.

Generally, time complexity of model training is proportional to

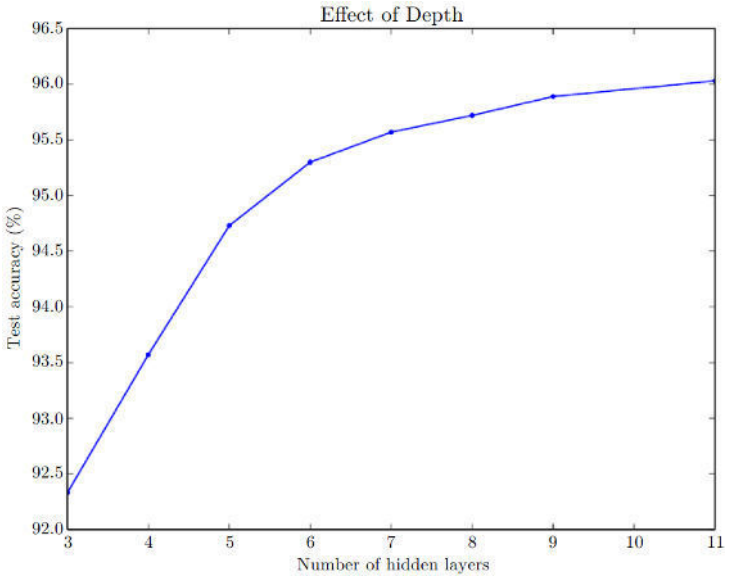
$$O = E \times T \times Q, \tag{1}$$

where E is number of training epochs, T is number of observations in training data and Q is time complexity of single forward-backward pass through the network, specific for each model [62].

4.4.5.2 Space Complexity

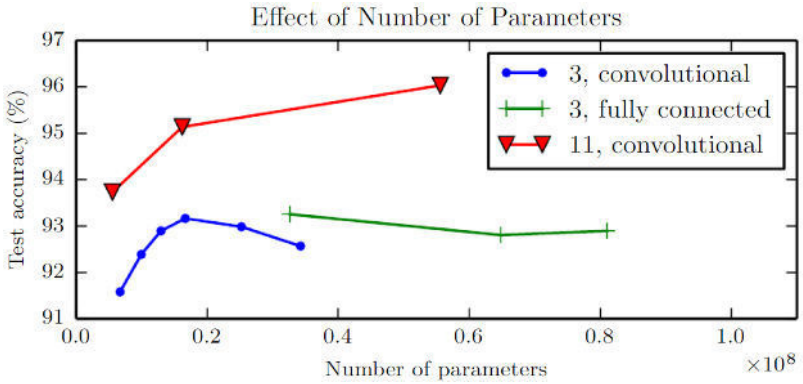
Since the structure of the network must be stored somehow, there is always some overhead due to this. Therefore, space complexity of a model is only roughly equal to its number of parameters.

Empirical results showed, that deeper networks generalize better when used to transcribe multi-digit numbers from photographs of addresses. On the figure below, it is shown that the test set accuracy is consistently increased with increasing depth [42].



Obr. 15: *Empirical results on performance as a function of model depth; retrieved from [42].*

Moreover, another experiment shows, that increasing the number of parameters in layers of convolutional networks without increasing their depth is not nearly as effective at increasing test set performance [42].



Obr. 16: *Empirical results on performance as a function of number of parameters; retrieved from [42].*

5 Task Definition

In this work we approach the problem of automatic music transcription from audio, with some specific requirements in terms of information retrieval. Our goal is to transcribe polyphonic music with best possible accuracy, and then find mapping of played notes to their sound sources (possibly musical instruments). Ideally, preserving information about dynamics of the sound such as velocity and volume over time. Assuming, that number of played instruments is unknown, additional challenge is to correctly identify different sound sources contained in an audio piece.

For the sake of accuracy, music transcription should be done by processing small consecutive time fragments. In addition, if sufficiently low time complexity is achieved, construction of real-time processing system would be possible as well. This would be restricted in case of need to pass through the whole audio performance in multiple iterations, calculating different features and based on known time context.

This is a challenging task, because for every tone present in a polyphony, we need to recognize its specific timbre[1]. Timbre is a characteristic of sound, which makes sounds with the same pitch and loudness sound different from one another. It has been called, "...the psychoacoustician's multidimensional waste-basket category for everything that cannot be labeled pitch or loudness."by McAdams and Bregman, in their work [20].

Attempts have been made to decompose timbre into set of fundamental attributes. Schouten describes them as "determined by at least five major acoustic parameters-[76] and states them as follows:

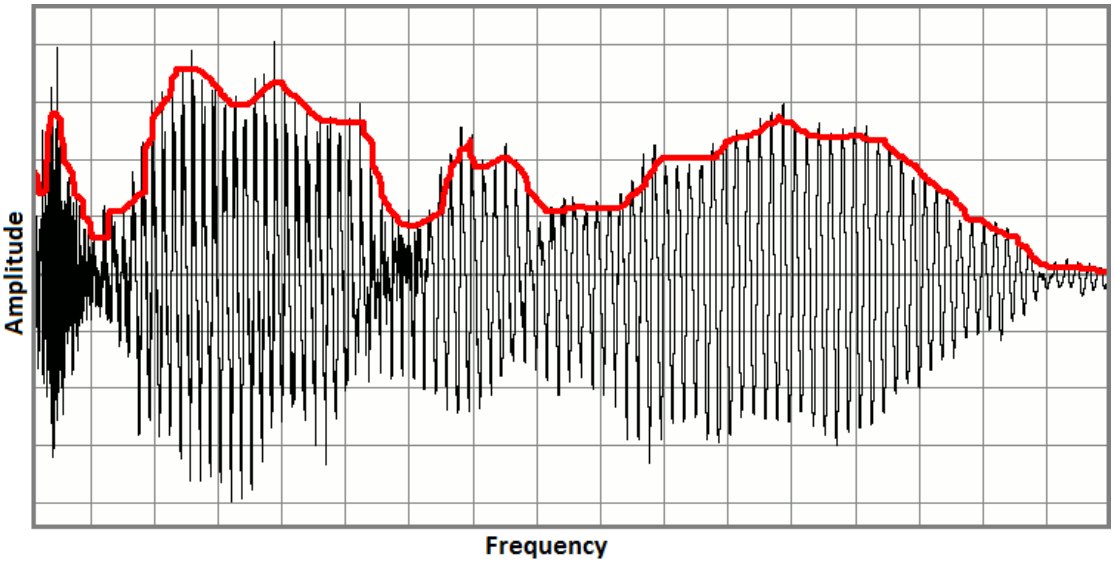
1. The range between tonal and noiselike character.
2. The spectral envelope.
3. The time envelope.
4. The changes of spectral envelope and fundamental frequency.
5. The prefix (onset) of a sound, quite dissimilar to the ensuing lasting vibration.

It is also important to note, that the tone color is known to be mostly determined by pattern of its harmonic components and their relative strength in time. From above stated attributes, the 2nd and 4th are the ones indicating this property.

[1] In music terminology, timbre is a quality of musical note, known also as *tone color*.

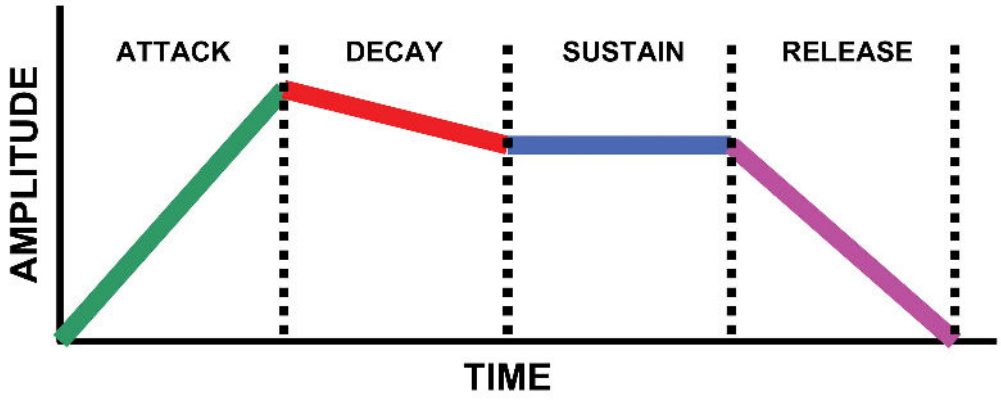
It seems, that some physical characteristics of sound which significantly contribute to the human perception of timbre include spectral envelope and time envelope.

Spectral envelope describes an envelope of the spectrum. It wraps tightly around the magnitude spectrum, linking the peaks [9]. It describes one time frame of audio signal, and carries information about the distribution of the signal's energy over frequency in that time frame. Example is show on figure below.



Obr. 17: *Spectral envelope of a signal; retrieved from [4].*

Time envelope, also known as Attack Decay Sustain Release (ADSR) envelope, as defined in [10] has its contour specified by attack time, decay time, sustain level and release time. Every musical instrument has its characteristic attack, sustain, and decay pattern. Therefore, time envelope is an important element of timbre.



Obr. 18: *Time envelope contour; retrieved from [5].*

Additionally, while transcribing music directly from audio, we would like to preserve the expressive means of musical tones. For stringed instruments, those would be vibrato, tremolo or pizzicato. Erickson states, that these means are related to specific physical attributes of timbre [31]. For example, vibrato seems to be mainly related to frequency modulation, while tremolo is physically an amplitude modulation.

Then, we also need a way to describe these expressive means for particular musical tones. For example, correctly played vibrato has its own tempo parameter.

Since there seems to be quite a bit of challenge in recognition required by the task described above, we might need to do some trade-offs while designing our model. Therefore, in terms of performance measured by transcription accuracy, we establish priorities for our subtasks from the highest to lowest as follows:

1. Detect tones in polyphony.
2. Assign tones to sound sources.
3. Detect and describe possible expressive means of musical tones.

6 Our approach

With previously mentioned specifics of acoustics and musical signals in mind, we consider multiple alternatives to approach the problem of polyphonic music transcription from the perspective of neural computing.

These alternatives differ mainly upon the way we represent musical data to the neural network, which also determines the scope of problem to be solved by the model and drives further design decisions about its architecture. We describe our ideas and proposed methods in this section.

6.1 Modelling Features in Frequency Domain

The initial step in almost each Digital Signal Processing (DSP) pipe-line for music content analysis is transformation of signal from time to frequency domain, thus construction of spectrogram. This is very intuitive and straightforward, according to the relationship between spectral content of the signal and harmonic content of corresponding music. Therefore, we first examine learning capacity of some neural network models trained on spectral feature descriptors.

6.1.1 Method Description

The overall process in steps necessary to apply this method is following:

1. Preprocess input audio data, generate spectrograms.
2. Preprocess reference labels, generate piano rolls.
3. Save inputs (spectrograms) with labels (piano rolls) correctly aligned in time.
4. Divide data into train, validation and test sets while attempting to maintain equal variety of musical content across these subsets.
5. Train the model on training set with periodical evaluation on validation data. Adjust hyperparameters across training sessions to maximize the validation accuracy.
6. Test the model by inference on test data. Treat output layer activations as estimated piano roll for quantified and empirical evaluations.

6.1.2 Preprocessing Phase

Initially, audio with reference notation is preprocessed one element at a time. In case of MIR data sets, one data element is mostly a pair of sound file and a notation file, containing single musical piece. Depending on the character of the data set, musical piece can be e.g. song, chord or a scale.

6.1.2.1 Re-sampling Method

To re-sample loaded audio signal, we use method described in [80]. It is based on interpolation by the Kaiser windowed $sinc$ function as defined by (2) normalized to 16 zero-crossings. Kaiser window is calculated according to formula (3)

$$sinc(x) = \frac{sin(x)}{x} \tag{2}$$

$$w[n] = \left\{ \begin{array}{cc} \frac{I_0\left(\pi\beta\sqrt{1-\left(\frac{2n}{N-1}-1\right)^2}\right)}{I_0(\pi\beta)}, & 0 \leq n \leq N-1 \\ 0 & otherwise \end{array} \right\} \tag{3}$$

where N is number of samples of the windowed function, I_0 is zeroth-order modified Bessel function of the first kind and parameter β is set to 8.5 to satisfy standard setup used for fast re-sampling.

6.1.2.2 Spectrogram Calculation

The CQT calculation is done by recursive sub-sampling method, described in [75]. The advantage of this method is that it allows to choose frequency resolution in terms of number of bands per note, lowest frequency to be examined and number of bins spaced geometrically among spectrum, satisfying Constant-Q factor condition. This results in having ability to choose range of musical notes to capture, in terms of fundamental frequencies.

Possible disadvantage of spectral bins being linearly spaced w.r.t fundamental frequencies of notes, is just the implication, that they end up spaced exponentially across the frequency range. Therefore, resolution at higher frequencies is lower, which may result in timbral features being captured with insufficient accuracy for instrument identification. This is not an issue, while we work with solo instrumental music, but as timbre relevance grows with

different subtasks, replacing CQT by standard STFT spectrogram calculation method is a reasonable subject to consideration and possibly experimental comparison. Also, combination of multiple spectral feature descriptors could be used as an input to the network.

Since this method uses recursive sub-sampling to calculate DFTs for octaves from highest to lowest, it places restrictions to sampling rate of input signal. Therefore, in case loaded audio has different sampling rate than required, initial step of this method is re-sampling.

Next, signal is processed by sliding window of size determined by sampling frequency and number of octaves being analyzed. Temporal resolution of resulting representation is given by the size of this window and length of stride in number of samples, which is another parameter to the algorithm.

Time resolution as the number of spectral feature descriptors per second of signal is given as a ratio of sample rate to stride size. For each time sample, Hann window is applied to extracted sequence and zero padding is performed up to the length of nearest power of 2.

Each frequency band in octave has its own filter length, which remains constant during the calculation, where logarithmic frequency spacing is utilized by sub-sampling between octaves and discarding already extracted information (higher frequencies).

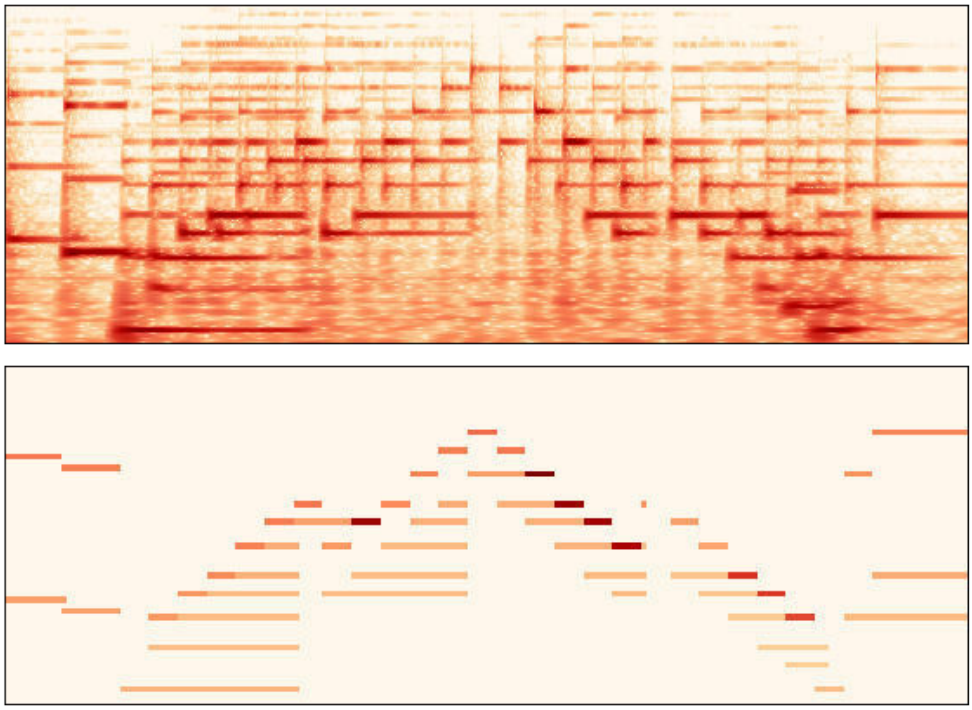
6.1.2.3 Piano Roll Alignment

Along with spectrogram calculation, piano roll is calculated from reference notation file. It is a format for visual presentation of musical MIDI events - specifically notes duration in time - created and used within instances of Digital Audio Workstations (DAW) software.

Internally we represent it as a matrix of values indicating which notes (rows) are being played in which time frames (columns). We use all 128 MIDI notes by default, although this is a bit redundant for piano music where the 88 notes of piano keyboard are sufficient.

The temporal resolutions of spectrogram and piano roll are chosen to be equal, so that they can be precomputed and stored together with correct time alignment, for purposes of easy and quick retrieval at the time of training.

The visualization of spectrogram aligned to piano roll is shown on Figure 19, where color intensity represents spectral amplitude power or note velocity power, respectively.



Obr. 19: *Example of input data aligned to labels.*

6.1.2.4 Further Steps before Training

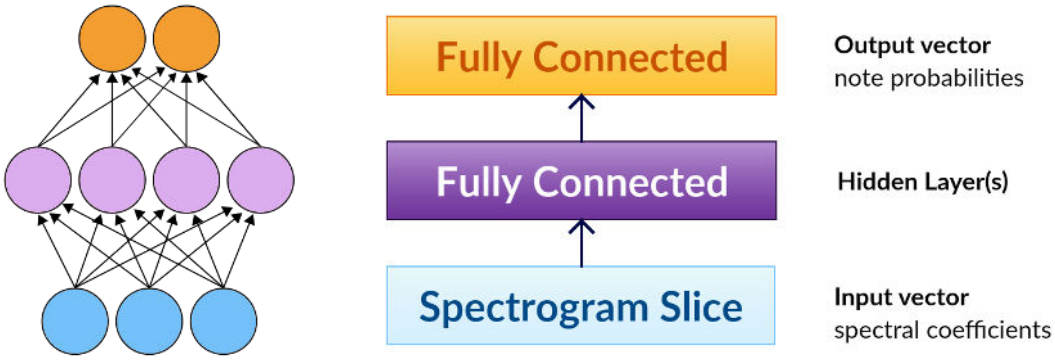
As the values of computed amplitudes may vary widely among the data, they are normalized to interval $[-\frac{1}{2}, \frac{1}{2}]$ according to peak amplitude contained in the whole audio piece. The choice of this mapping is also motivated by general observation, that neural networks yield better performance on data with zero mean and small, equal variance [54].

Another variation of preprocessing is to discard the dynamics information, which is what we do to reduce the task by estimation of dynamics to simple detection of notes presence, otherwise the task is to estimate both at the same time. It is done by simply truncating the velocity values in piano roll matrix to $\{0, 1\}$. This makes the piano roll a binary matrix, indicating just presence of notes in time.

Lastly, the data are divided into training, validation and testing sets in some reasonable ratios. This is, however, done in such a manner, that preserves the most possible musical variety within all constructed subsets.

6.1.3 Architecture Design and Training Phase

To be able to iteratively build optimal architecture, we initiate our efforts with following configuration. We build simple MLP model and feed it with spectral coefficients on the input, yielding discrete probabilities of presence per note from the output. Therefore, we denote this model as *MLP-Spec*.



(a) Multi-Layer Perceptron (b) Default setup of MLP model

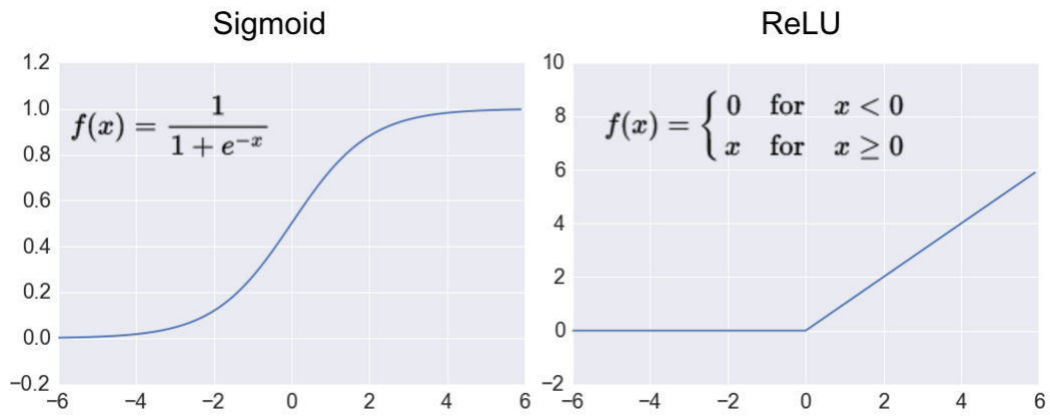
Obr. 20: *Outline of MLP-Spec architecture.*

The size of input layer depends on spectral resolution chosen during the preprocessing as a combination of parameters to CQT spectrogram calculation. Size of output layer is given by range of estimated notes defined through provided piano roll labels. Number and size of hidden layers and their activation functions belong to set of model hyperparameters.

Initial setup shown on Figure 20 consists of single hidden layer and an output layer. To compute activations on hidden layer we use Rectified Linear Unit (ReLU) activation for computational efficiency and sufficient modelling capacity.

We use sigmoid activation on the output layer, since estimating presence of notes in polyphony is a multi-class classification task. This function activates the output layer preactivations to a set of values from interval $(0, 1)$ representing discrete probability estimations per note.

On Figure 21, we plot these default activation functions. Additionaly, we use Adam [50] as the default algorithm for model optimization. However, this is just a default setup and experimental search is to be done in order to find optimal set of hyperparameters.



Obr. 21: *Graph plots of used activation functions; reprinted from [8].*

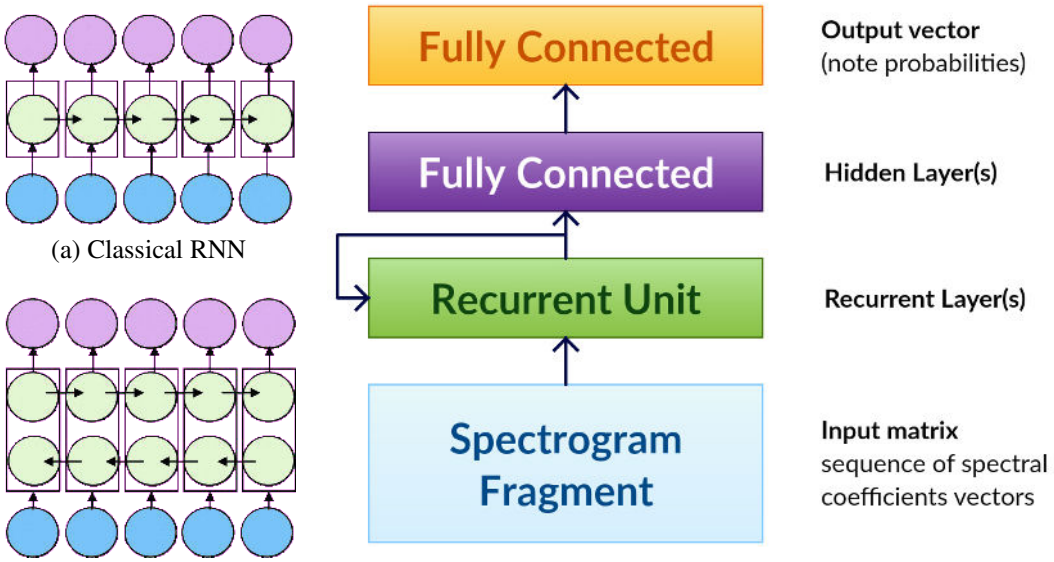
In next iteration of architecture design based on initial results, we propose slight modification to MLP-Spec. In order to provide the model with some context of the estimated time frame, we add a recurrent layer on the input and feed it with a *sequence* of spectral descriptors of consecutive time frames, instead of single one as opposed to MLP-Spec feeding method.

First, we apply classical *uni-directional* recurrent layer to process sequence of consecutive spectral descriptors. At prediction time of each frame, hidden state of recurrent unit encodes information gained by sequence of past time frames.

Although predictions are made for each time frame, only last one of the sequence is predicted with utilization of whole sequence. Therefore, at training time, it is considerable whether to backpropagate error from whole sequence of predictions, or only from the last one. At the inference time, we use only the last prediction of the sequence at the cost of higher inference time, since we need to feed spectrogram fragments with huge overlap of `n-1` where `n` is length of the sequence.

Additionally, in order to enable utilization of time context from *both* sides of estimated frame, we examine the use of *bi-directional* recurrent layer. In order to provide equal time context from both sides, we only use sequences of odd lengths and train to predict the frame at the center of sequence, instead of the last one.

Based on type of recurrent layer used within particular model variation, as outlined on Figure 22, we denote the model with classical one as *RNN-Spec* and the model with bi-directional one as *BiRNN-Spec*.



(a) Classical RNN

(c) Bi-directional RNN

(b) Default setup of recurrent models

Obr. 22: *Outline of recurrent architectures.*

Though length of the sequence is not restricted by RNN, as we currently have no means of determining the desired length of context, we choose a constant value for each training session, ending up with additional hyperparameter to tune.

There are of course many possible adjustments to these models we could examine, e.g. placement of recurrent unit on the top of MLP classifier, between the last hidden layer and fully connected output layer. Multiple recurrent units can be also used on different places and their outputs combined using residual connections.

It should be made clear, that generating batch of data samples during training differs among these model, since each model takes different data unit as input observation and according to this, output labels are also chosen specifically.

During training of MLP-Spec model, frames are sampled from training set according to pseudo-randomly generated distribution of indices to array or queue of training examples, based on size of data set and thus batching method. In similar fashion, sequences of frames are pseudo-randomly sampled during training of recurrent models.

At time of inference for testing, sequential batching is a straightforward method to use, although recurrent models get zero-padded spectrograms in order to compensate the time missing context at start (or end) and get equal shape of estimated piano roll to ground truth.

6.1.4 Postprocessing

In scope of simple note detection task where information about dynamics of individual notes is discarded, the last processing step we do is *thresholding*, which simply means rounding estimated probabilities to logical values. This is performed according to specified threshold q, which is a *parameter of evaluation*, given

$$R_{n,t} = \begin{Bmatrix} 1, & \text{if } P_{n,t} \geq q \\ 0, & \text{otherwise} \end{Bmatrix} \tag{4}$$

where note n and time frame t are indexes to arrays P for estimated probabilities and R for resulting piano roll.

With this representation, standard metrics for evaluation of information retrieval tasks can be computed. We further use this representation for empirical evaluation. Visual plots of piano rolls can be created and MIDI representation can be reconstructed, which can be played using different soundfonts and libraries of various musical instrument audio samples.

6.2 Modelling Features in Time Domain

Although spectral analysis is very important step in most signal processing tasks, it comes at the cost of losing some (howsoever small) amount of information about the signal. By windowing the audio snippets during spectral analysis, we disrupt the signal continuity information and yet still some artifacts are generated in the spectrum.

Alternatively, excluding any feature extraction from preprocessing phase, one could treat the musical data in its raw form, as a sequence of samples in time. However, this way, the frequency content analysis is an additional task left for the neural network to tackle.

The idea is inspired by success of various deep models applied to signal processing tasks, which could learn multiple feature maps on different layers of representation.

The fact, that raw audio signal contains all the necessary information for human brain to understand and perceive the musical content of the signal is tempting for attempts to design a deep neural network architecture with sufficient capacity to be trained for this task and mimic this process. After all, recent advances in various image processing tasks made by deep neural networks have demonstrated relevance of this approach. This suggests, that deep neural networks could be able to learn feature representations, which will be more robust against acoustic variations in audio signal, and therefore also better *tailored* for the task at

hand. Also with raw signal on the input, network has enough information to gain ability to perform multiple transcription tasks simultaneously, such as dynamics estimation or sound source recognition.

Besides the recent success of deep CNNs at processing visual data, which is a raw signal too, recent advancements also demonstrated, how deep networks can effectively learn to model structure of signal from raw sequences of audio samples [84]. They proved this by generating highly comprehensible quality utterances of speech and musical signals with *WaveNet: A Generative Model for Raw Audio* [2].

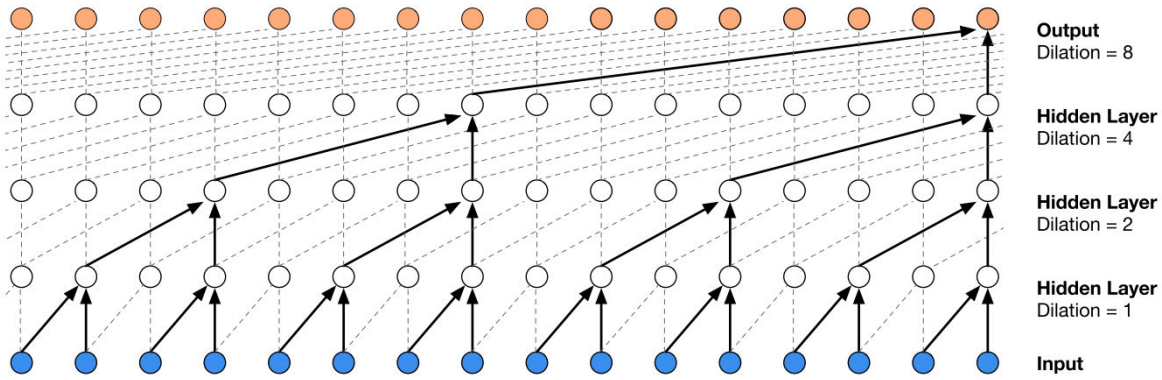
We initiate our work in this direction by examination of possibilities to re-use the fundamental ideas of WaveNet architecture. Since this model has been proven effective on task of audio synthesis, our goal is to examine its capacity on task of multiple F_0 estimation.

6.2.1 WaveNet for Transcription

In this section, we describe the WaveNet architecture as introduced in DeepMind paper [84]. We further describe how we adjust and train this model to perform frame-level transcription of polyphonic music.

6.2.1.1 The WaveNet Architecture

The architecture of WaveNet is built out of 1D convolutions with some special properties. Figure 23 depicts an exemplary stack of *dilated causal convolutions* with dilation factors 1,2,4, and 8.



Obr. 23: *Stack of* dilated *causal convolutions; reprinted from [84].*

The important thing about causal convolutions is that they do not use any of the future timesteps when calculating prediction for timestep *t*, in order to preserve any possible *causal* relationship between subsequent values in given series. This enables to model conditional probability distribution for value of next sample, conditioned over sequence of preceding samples [84].

The issue with regular causal convolutions applied to `16 kHz` audio is the lack of means to effectively increase receptive field[1] of the network, since the formula is

$$receptive\ field = |layers| + filter\ width - 1 \tag{5}$$

and therefore to achieve any reasonably large receptive field would require too many layers or very large filters.

That is why *dilation factor* is very useful. It enables to increase the receptive field by orders of magnitude, without greatly increasing computational cost, through enabling to apply convolution filter over an area larger than its length by skipping input values with a certain step [84]. Provided that all layers use same filter width and one simple causal convolution is applied first, followed by stack of dilated causal convolutions, the formula is

$$receptive\ field = (filter\ width - 1) \times \left(\sum_{d}^{dilations} (d) + 1 \right) + 1 \tag{6}$$

where $dilations$ is a set of dilation factors describing by the stack of dilated convolutions.

By increasing dilation of successive layers by factor of 2, receptive field of the network grows exponentially with its depth. Another means of increasing the receptive field is to further stack multiple such blocks, one on another. This helps to increase the receptive field size as well as model capacity. For example, authors state their experimental configuration as follows:

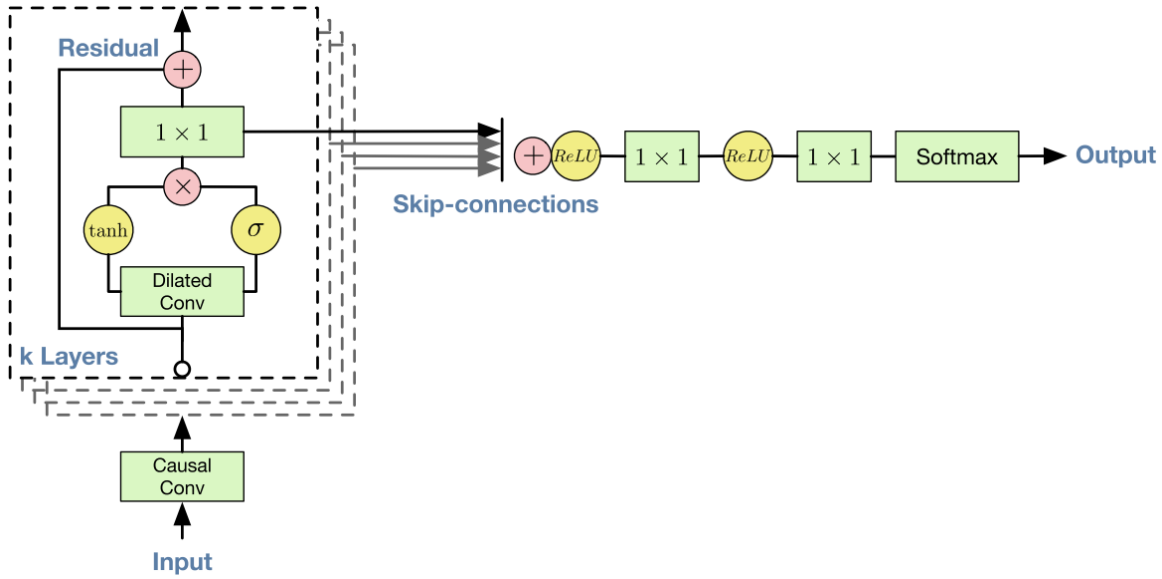
$$1, 2, 4, ..., 512, 1, 2, 4, ..., 512, 1, 2, 4, ..., 512.$$

Additionally, in order to make the prediction of samples more tractable, authors train WaveNet on samples quantized to 256 possible values, to predict the value of next sample using *softmax* distribution over these 256 possible values.

Another key feature of WaveNet is the engagement of *residual* and *skip* connections across the whole stack of dilated convolution layers. By introducing addition operation to

[1] Receptive field is the number of samples WaveNet can use to compute single prediction.

network computation graph, they help to speed up convergence and enable training of much deeper models. This again starts to make sense once we realize how addition benefits the propagation of gradient through the network back to the initial layers, in terms of reverse-mode differentiation [1]. Their use within a single residual block along with entire WaveNet architecture is depicted on Figure 24.



Obr. 24: *Overview of WaveNet architecture and its residual block; reprinted from [84].*

Also,the residual and skip connections are parameterised, which means each connection has its set of weights being optimized to let the exactly required amount of information flow through their channels.

Further features of WaveNet architecture include global and local conditioning on the input. For TTS model it makes sense to be conditioned globally e.g. on a speaker embedding, whilst locally on some linguistic features such as sequence of phonemes to generate.

6.2.1.2 Proposed Adjustments

Inspired by previous success of WaveNet in audio modelling domain, we build our version based on its open source implementation[1] and adjust it for the task of frame-level transcription.

Since generation and transcription are fundamentally different tasks, the data preparation for training and inference differs as well. However, the advantage of WaveNet convolution

[1] `https://github.com/ibab/tensorflow-wavenet`

mechanics which allows the parallel processing of sequences significantly longer than the receptive field to be processed in single inference step, producing sequence of predictions of length $(input\ length - receptive\ field + 1)$, becomes quite obsolete. This is because WaveNet yields output sequence at temporal resolution equal to input sample rate, which is at least `16 kHz` in music transcription scenario, while temporal resolution of `100 fps` is established as absolutely sufficient for frame-level transcription, according to measured limitations in human perception of rhythm.

In order to utilize the convolution mechanics of WaveNet, we train and evaluate on large sequences. However, in real world use-case when we want to visualize or MIDI-reconstruct the output, we further sub-sample it by average pooling to get reasonable temporal resolution. Alternatively, we could append another layer or pooling operation to the network output for sub-sampling and work with appropriate resolution of output all the time.

In order to provide the network with time context from both sides, similarly to BiRNN-Spec, we force the alignment of estimated piano roll frame to the center of input sequence. This is done by padding input sequence by $\left\lceil \frac{receptive\ field-1}{2} \right\rceil$ zeros from left side and by $\left\lfloor \frac{receptive\ field-1}{2} \right\rfloor$ zeros from right side.

As opposed to generative WaveNet, we omit the non-linear quantization of input samples and feed raw data instead, since quantized representation has proved redundant by showing little to none influence on transcription performance.

We replace *softmax* activation function with *sigmoid* on the output to get multi-class classifier network, instead of single-class classifier. The number of output channels changes from 256 originally quantization channels to 128 now MIDI note numbers.

We don't have a way to employ model conditioning at the moment, since audio signal is the single relevant data source for given task. However, in Section 8 we mention some of our ideas to utilize local and global conditioning in several various AMT scenarios.

For further reference and evaluation purposes, we denote this adjusted WaveNet model by acronym *WN4T* (WaveNetForTranscription) within the scope of this work.

7 Evaluation

To evaluate our methods, we combine resulting estimations with ground truth to quantify the results. For empirical evaluation, we also visualize this combination and additionally, we generate MIDI from estimations and compare audio representations of estimated notation to ground truth.

7.1 Methodology

It is common that training sessions of deep networks on large data sets take not just hours, but rather days. Therefore, performance of trained models is evaluated in regular intervals, and results serve to track the progress of training.

For this intermediate evaluation, validation set of data is reserved. Based on performance on this data set, different model configurations are examined across experiments[1]. Finally, when training and tuning is finished, model is evaluated on test set, which is another set of data reserved for this purpose.

In sake of correctness, training, validation and testing data should always consist of disjoint sets of data samples. Unless explicitly stated otherwise, we stick to standard ratios of distribution between these sets, as stated in Table 1.

Tabul'ka 1: *Standard ratios of data set distribution in machine learning.*

Training	**Validation**	**Testing**
80 %	10 %	10 %

From estimations and ground truth labels on evaluation data, we compute standard information retrieval metrics. Namely $precision$, as ratio of correctly estimated notes against all estimated notes, and $recall$, as ratio of correctly estimated notes against all ground truth notes. We mainly track their harmonic mean - F_1 score, as a single and most relevant indicator of performance, given by (7).

$$F_1 = 2 \cdot \frac{1}{\frac{1}{precision} + \frac{1}{recall}} = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{7}$$

[1] In deep learning jargon, this is also called *tuning the hyperparameters*.

Though we track measure of $accuracy$ as number of correctly estimated values against all values, we don't report this metric since it is much less informative than F_1 score. This is because musical data representation by piano roll yields a very sparse matrix[1] in which any model is always able to gain first $\approx 95\%$ of accuracy just by correctly estimating silence.

However, in order to enable comparison to previous work, we also measure *frame-level accuracy* (8) as proposed by Dixon in [28]

$$Acc = \frac{TP}{(FP + FN + TP)} \tag{8}$$

where TP, FP and FN denote total counts of true positives, false positives and false negatives. Evaluation metrics are computed from estimations and labels. To examine confusion empirically, we further create visualizations of estimations or use piano roll to construct playable MIDI file.

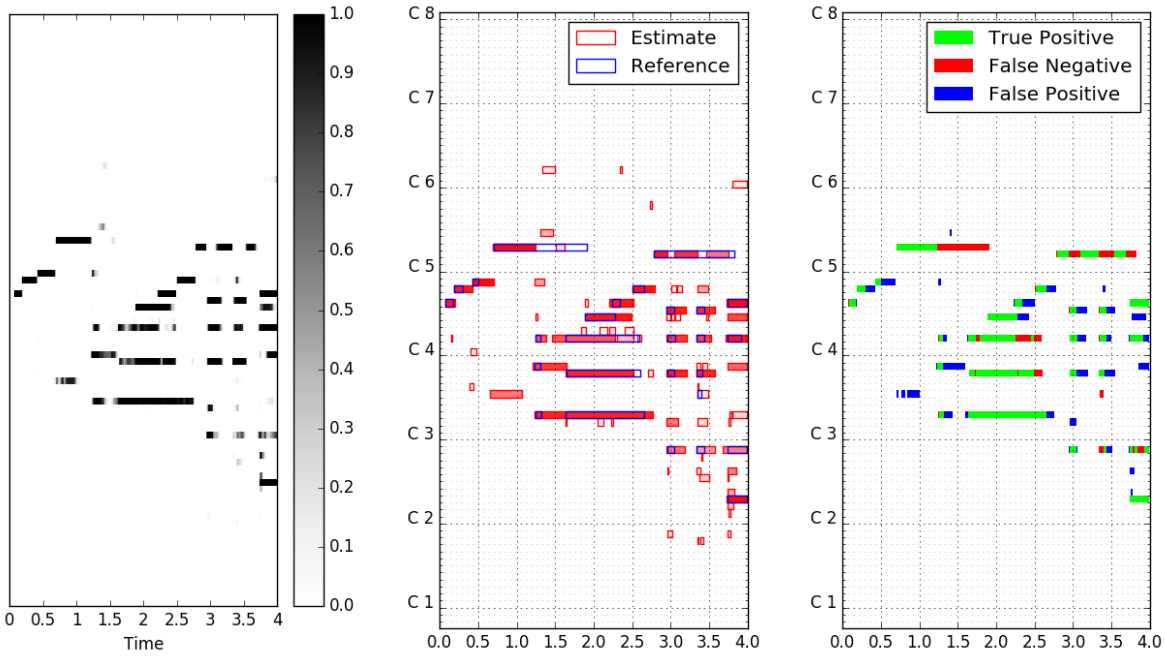
Since we tackle a multi-class classification problem, construction of confusion matrix is not an option. Therefore, to understand the performance issues of our models, we examine cherry-picked transcription fragments at least manually using colorful visualizations of true positive, false negative and false positive estimations.

Additionally, we use early stopping technique on training sessions, in terms of regularly validating model performance and saving checkpoint of model parameters, in case performance is improved against current maximum. When there is no improvement for specified number of checks, training is stopped, best model checkpoint is restored and evaluated on test data.

7.1.1 Visualizations

We demonstrate the use of our evaluation plots to examine the transcription fragments as depicted below on Figure 25. In all plots, horizontal axis represents time in seconds, while vertical axis represents notes ordered chromatically according to piano keyboard - from lowest (bottom) to highest (top).

[1] Most of time in music, only tiny fraction of all playable notes are being played.



(a) Estimations (b) Certainties (c) Evaluations

Obr. 25: *Evaluation plots.*

On Subfigure 25a we see raw matrix of predicted probabilities with colorbar denoting probability values.

Next, Subfigure 25b shows certainty of predicted note events after piano roll is converted to set of MIDI events. Conversion is performed by linear scan of each note across time axis. Note event is generated from each coherent sequence of predictions with value above specified threshold `0.007`, since sigmoid activation function will never return zeros, only values close to zero. Certainty of prediction is calculated as mean of all prediction values in given sequence and it is expressed through alpha channel of plotted note bar, thus by color intensity. Onset and offset times are set to first and last predictions of the sequence.

Finally, on Subfigure 25c we see the retrieved values after thresholding in contrast with ground truth. White space represents True Negatives, while legend explains the remaining colors. This allows us to instantly observe the fail modes of our models.

To generate these visualizations, we utilize `mir_eval` [71], an open-source python library of evaluation functions for MIR and audio signal processing algorithms.

7.2 Data Sets

In this section we state details of data sets used in our experiments. Parameters of preprocessing which were applied equally to all data with no variations amongst experiments are stated as well.

7.2.1 LabROSA

The Laboratory for the Recognition and Organization of Speech and Audio[1] (LabROSA) provides a data set of recorded live piano performances[2] to public. This is a part of larger data set which has been used by Poliner & Ellis in their work [67].

It is meant to extend the larger subset of data generated from MIDI notation, to enrich the training and testing set by timbral variations of live music performance.

Recorded part of data set consists of `.wav` audio files and `.mid` reference files. Audio is sampled at sample rate `8000 Hz` and depth of dynamic range `16 bits`.

This, according to Nyquists sampling theorem, is theoretically sufficient for piano music, as denoted in [67], because highest note playable by piano is $B7$ with fundamental frequency of `3951.1 Hz`. However, some timbral features of higher tones are definitely lost, since they are located far above the Nyquist frequency of `4000 Hz`.

In following paragraphs, we describe three variations of this dataset, which have been used across our experiments.

7.2.1.1 LabREC - Recorded Tunes

The set of recorded performances provided by LabROSA contains first 60 seconds of 29 different compositions.

From each of these labeled recordings, first `40` seconds were divided into `32` training, `4` validation and `4` testing seconds of data. With resolution of `100` frames per second this amounts to `92800` training, `11600` validation and `11600` testing samples.

We start our initial experiments with this subset, as its size allows for fast prototyping and quick experimentation with simple models. Since we further use different variations of LabROSA dataset, we denote this variation as LabREC.

[1] `http://labrosa.ee.columbia.edu/`
[2] `http://labrosa.ee.columbia.edu/projects/piano/`

7.2.1.2 LabSYNTH - Synthesized Tunes

For next set of experiments, we use MIDI files of tunes from LabREC and synthesize the audio using high quality piano soundfonts[1], resulting in another LabROSA variation we call LabSYNTH.

For audio synthesis, we use 3 different piano soundfonts: Nice-Keys-Extreme-V1.5[2], Arachno[3] and Yamaha Disklavier[4]. During the training, batches are generated by random sampling from randomly chosen tunes with randomly chosen version of synthesis. The goal is to prevent the models from overfitting to timbral characteristics of single piano instrument.

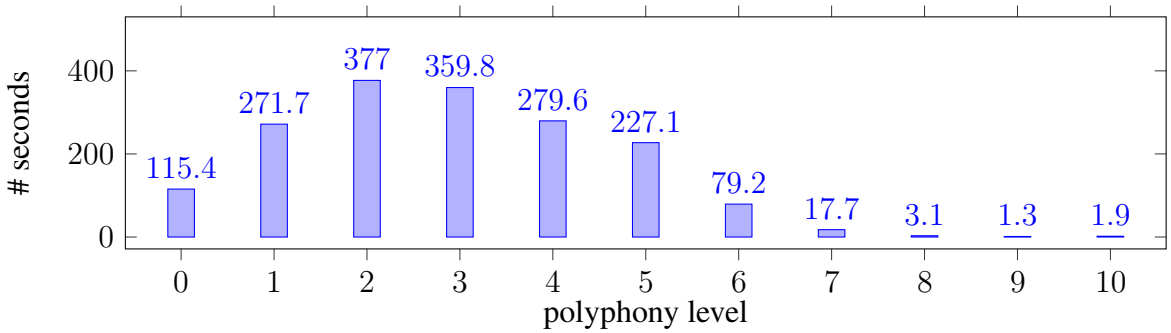
This set is divided into subsets in exactly same way as *LabROSA-rec*, resulting in `89600` training, `11200` validation and `11200` testing samples

Since LabREC and LabSYNTH contain the identical collection of MIDI files, they can both be described equally in terms of musical content.

Tabul'ka 2: *Statistics for musical pieces and note events of* LabSYNTH.

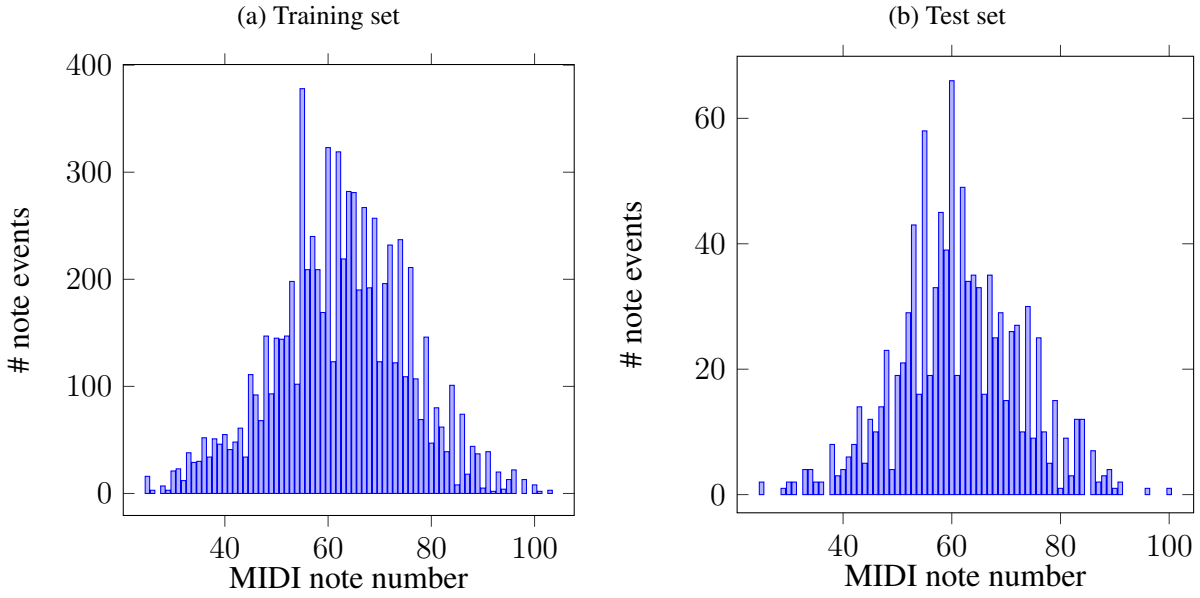
Duration of pieces				**Note events**	
Total	**Average**	**Max**	**Min**	**Average duration**	**Total count**
29 m	60 s	60 s	59 s	0.22 s	15066

Table 2 summarizes data set statistics on level of musical pieces and note events. Figure 26 shows distribution of frames with various polyphony levels. On Figure 27 note occurrence histograms in training and test subsets are shown side-by-side for easy visual comparison.



Obr. 26: *Distribution of polyphony in number of samples across* LabSYNTH.

[1] `http://www.synthfont.com/sfspec24.pdf`
[2] `https://sites.google.com/site/soundfonts4u/`
[3] `http://www.arachnosoft.com/main/soundfont.php`
[4] `http://zenvoid.org/audio/`



Obr. 27: *Note occurrence histograms in training and test subsets of* LabSYNTH.

7.2.1.3 LabCOMP - Complete Dataset for Reference Evaluation

To enable comparison of our method to reference approach, we try to obtain the most possible similar reconstruction of dataset used in the paper [67]. Although the paper contains comprehensive description of used data set in a table, its content contradicts with detailed description of used data in Section 2.1 Audio Data, which states that data contains "95 training, 25 testing and 13 validation pieces", while table lists only 87 training and 24 testing pieces.

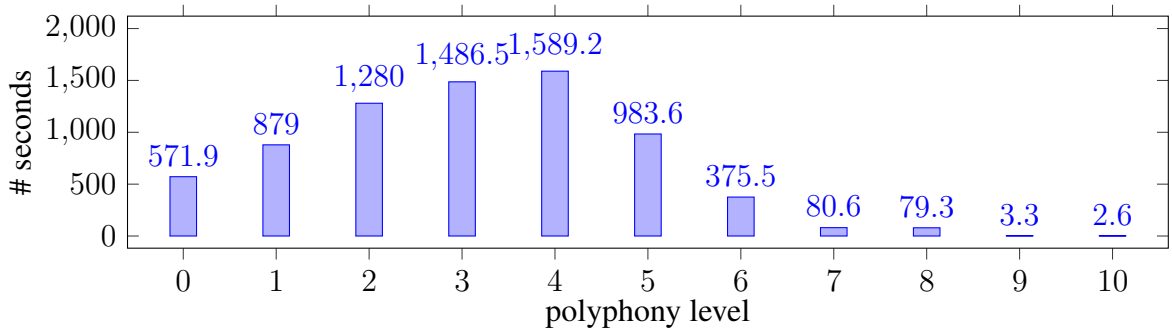
Nevertheless, we retrieved the published recordings from publication web site and collected MIDI files of remaining tunes from referenced site[1] according to table of MIDI compositions attached with the paper [67]. Additionally, we also used the same soundfont for MIDI synthesis - Yamaha Disklavier grand piano. This variation is denoted as LabCOMP.

We describe the exactly same statistics as previously to enable comparison of data sets.

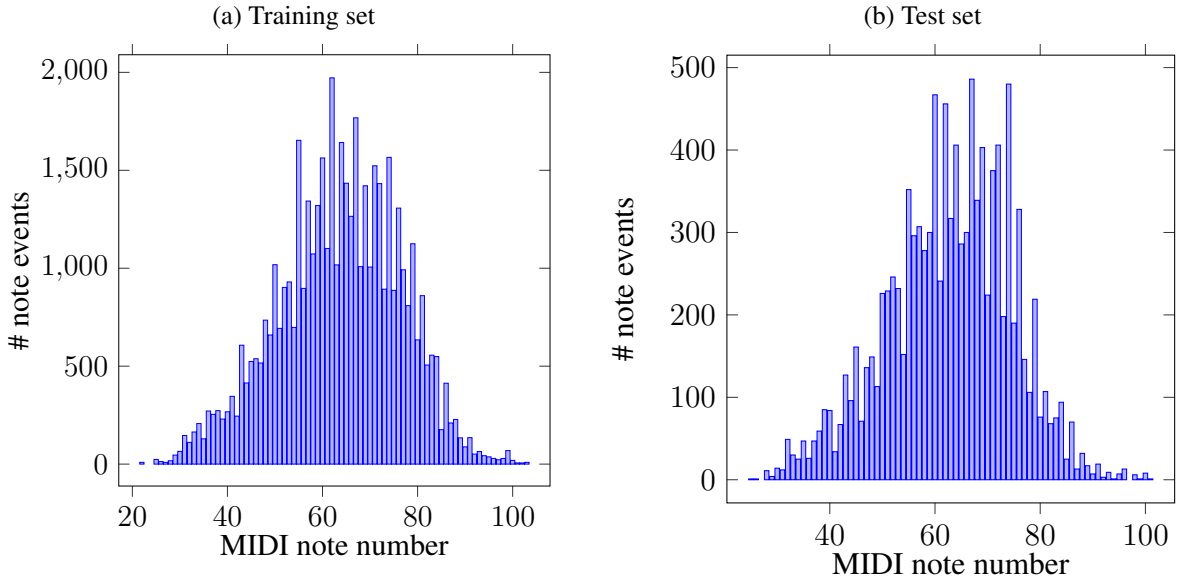
Tabul'ka 3: *Statistics for musical pieces and note events of* LabCOMP.

Duration of pieces				**Note events**	
Total	**Average**	**Max**	**Min**	**Average duration**	**Total count**
2.04 h	59 s	60 s	23 s	0.23 s	65510

[1] `http://piano-midi.de/`



Obr. 28: *Distribution of polyphony in number of samples across* LabCOMP.



Obr. 29: *Note occurrence histograms in training and test subsets of* LabCOMP.

7.2.2 PIMIDE

This set represents the largest collection of classical piano tunes we collected. After duplicates removal, it consists of 337 MIDI tunes all retrieved from Classical Piano MIDI Page website[1] including `format0` MIDI files[2].

Purpose of this dataset is to enable training of large-scale deep models with bigger size

[1] `http://piano-midi.de/`
[2] `http://piano-midi.de/midis/format0/`

and capacity, although due to time constraints we only performed two large-scale experiments with this data set.

Yet before we finding that PIMIDE is over one order of magnitude larger than previously used datasets, in preparation for even larger models, we extend this dataset with 12-fold transposition[1], in order to get each tune into each of 12 different keys. We denote this model as PIMIDE_EXT and provide statistics of both sets in tables below.

Tabul'ka 4: *Statistics for musical pieces and note events of* PIMIDE.

Duration of pieces				**Note events**	
Total	**Average**	**Max**	**Min**	**Average duration**	**Total count**
22.97 h	4.09 m	33.52 m	23.34 s	0.21 s	784262

Tabul'ka 5: *Statistics for musical pieces and note events of* PIMIDE_EXT.

Duration of pieces				**Note events**	
Total	**Average**	**Max**	**Min**	**Average duration**	**Total count**
264.71 h	3.66 m	33.52 m	3.95 s	0.21 s	8853029

7.2.3 MAPS

MIDI Aligned Piano Sounds[2] (MAPS), is a piano sound database dedicated to research on multi-F0 estimation and automatic music transcription [30].

It is distributed under Creative Commons License and we obtained access to a somewhat limited (10GB) version of this data set upon e-mail request. Construction of contents of this data set is described in detail in [14]. This data set provides following categories of sounds in a denoted subsets:

- ISOL - isolated notes and monophonic sounds.
- RAND - random chords.

[1]Our algorithm for 12-fold transposition attempts to transpose each tune into all different keys. If there is free piano range, direction is chosen such that center of *transposition octave* approaches center of the keyboard. Its implementation is located at `/prep/transp.py` on electronic medium attached to this work. For more details refer to Appendix A.

[2]`http://www.tsi.telecom-paristech.fr/aao/en/2010/07/08/maps-database-a-piano-database-for-multipitch-estimation-and-automatic-transcription-of-music/`

- UCHO - usual chords.
- MUS - pieces of music.

Dataset consists of `.wav` audio format sampled at sample rate `44100 Hz` and depth of dynamic range `16 bits`. Ground truth is provided in MIDI and also text format.

7.2.4 Common Preprocessing Parameters

To perform DSP routines necessary for audio preprocessing, we used `librosa` - python library for audio and music analysis [60]. To generate piano rolls from MIDI files and vice-versa, we utilized another python library `pretty-midi` [70] and implemented custom routine for MIDI reconstruction from piano-roll. Though this representation does not carry most of the notation metadata, in some situations, simply note onsets and offsets are sufficient for data examination.

In order to achieve sufficient time and frequency resolution of CQT spectral analysis, we used following parameters to the algorithm.

- `sample rate = 25600 Hz` for initial re-sampling.
- `hop length = 256 samples` for temporal resolution[1].
- `minimum frequency = 27.5 Hz` corresponding to note A0.
- `number of spectral bins = 88 * 3` having 3 bands per note.
- `bins per octave = 12 * 3` since octave has 12 semi-tones.

For this configuration, temporal resolution of piano roll was set to `100 fps`, in order to align properly to spectrograms.

Note, that this is the baseline configuration for preprocessing phase. Wherever we used data which were preprocessed with different parameters, the difference is explicitly stated.

7.3 Experiments

In this section we present several performed experiments, which exposed some interesting results. At all of our experiments, we used the value of threshold $q = 0.5$ unexceptionally.

[1] This configuration results in temporal resolution of `100 fps` (frames per second).

7.3.1 Initial Attempts

First set of experiments was conducted on LabREC - a subset of recorded piano performances.

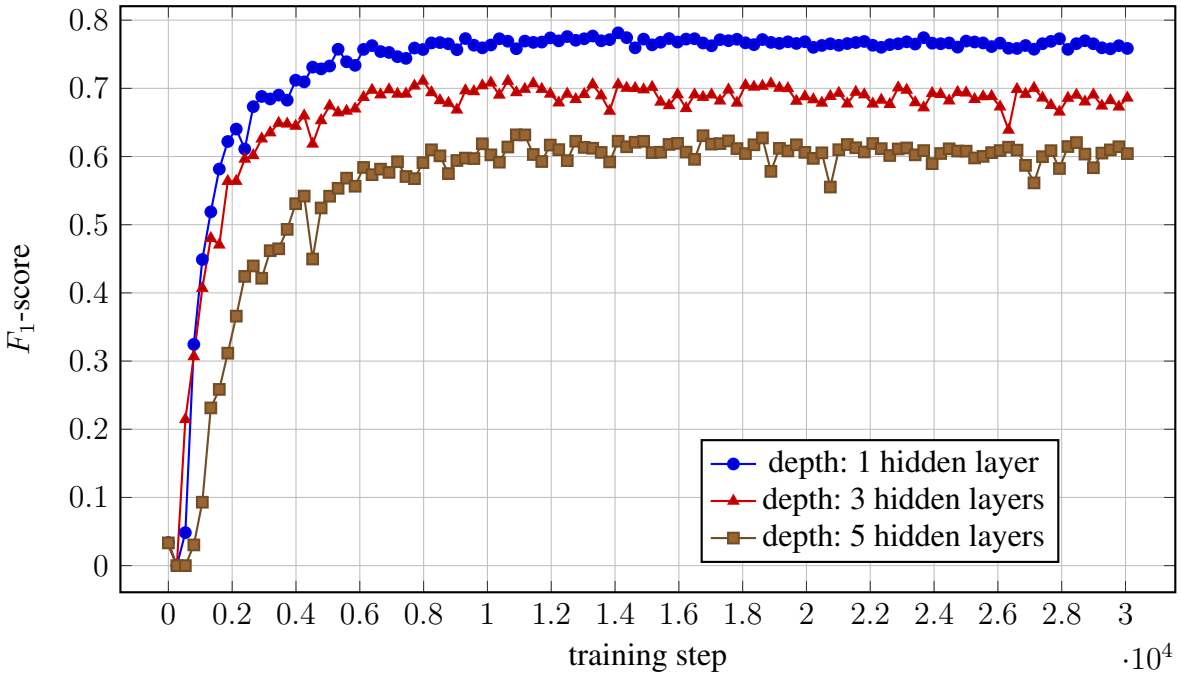
7.3.1.1 MLP-Spec: Model Depth

We examined influence of model width on validation accuracy during training, by visualizing F1 score convergence curve.

Tabuľka 6: *Experimental setup: model depth*

Model	**Batch size**	**Width**
MLP-Spec	300 frames	200 units

Number of hidden layers (depth) of the model was variable parameter, while others remained constant between the runs. Hidden activation functions (ReLU) and optimizer (Adam) remained from default configuration.



Obr. 30: *MLP-Spec performance at different model depths.*

Results on Figure 30 show us trend of decreasing validation performance with increasing model depth.

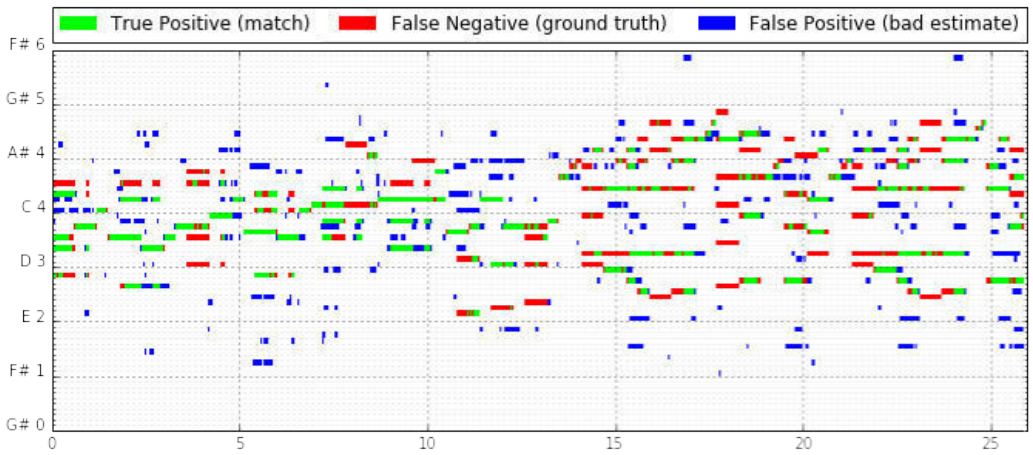
Apparently, this is related to so called *skinny jeans problem*, which basically means to find the right model size for the size of training data. With increasing model size but same data corpus size, problem of *overfitting* occurs. On the contrary, if model size and thus capacity is too small for the data size, *underfitting* occurs.

Enlarging the data set is one possible approach to fix this issue, although even if we can easily get more data, this is not a reliable approach, since we want our model to be ready for absorbing new data, just in case, and not rely on exact data size determined experimentally.

One common solution to this problem is to have model with larger capacity than it needs for given data corpus, and introduce regularization techniques during the training, in order to prevent the model from overfitting.

7.3.1.2 RNN-Spec: Context Sequence Length

Results of MLP-Spec model showed us, as visualized on Figure 31, that the predictor is somehow missing the time context of predictions it generates. This is obvious from many estimations of notes with very short duration. Therefore, we designed RNN-Spec, in order to introduce the concept of note duration.



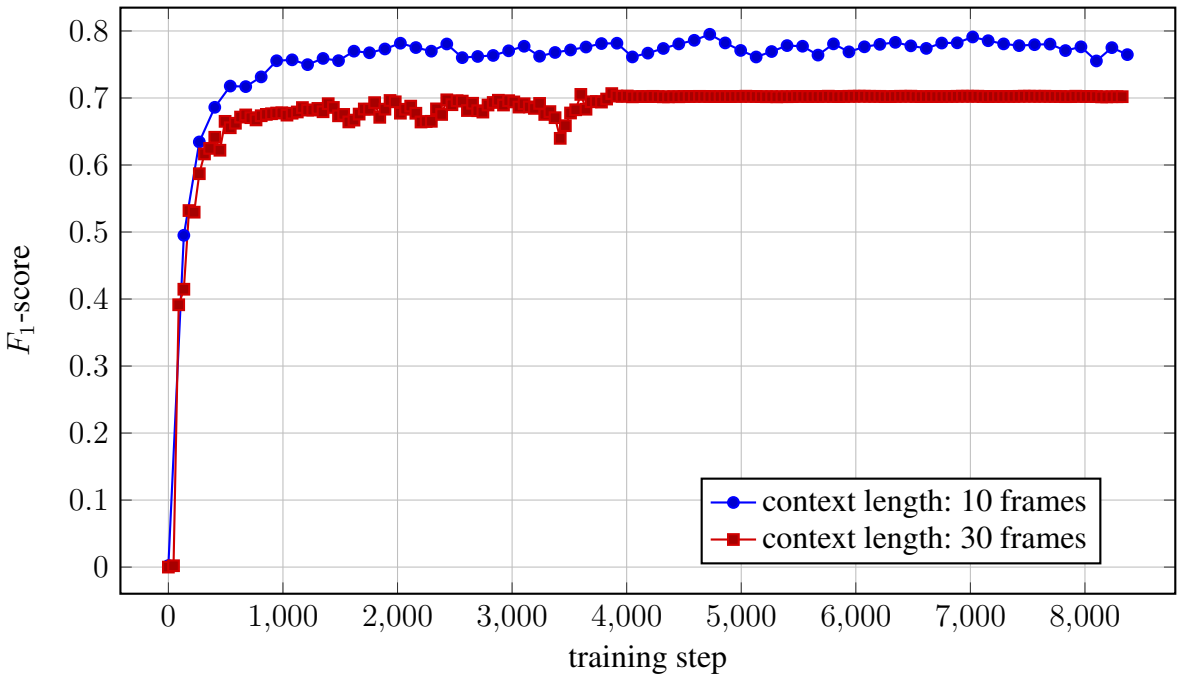
Obr. 31: *Example visualization of estimations against true labels.*

We examine how length of context provided to recurrent cell will influence the accuracy.

Tabul'ka 7: *Experimental setup: context length.*

Model	Batch size	Width	Depth	Cell
RNN-Spec	64 sequences	1000 units	1 layer	GRU

In this setup, parameter *width* as number of units is reused in both hidden and recurrent layer. We also lower the batch size against previous experiments, but since each input is now a sequence of frames, actual number of frames contained within one batch is larger.



Obr. 32: *RNN-Spec performance at different input sequence lengths.*

On the plot of validation performance, we can see how extended context from `100 ms` to `300 ms` providing the model with higher modelling capacity actually made the model overfit the training data very quickly, so it became stuck in some local optimum after ≈ 4000 training steps.

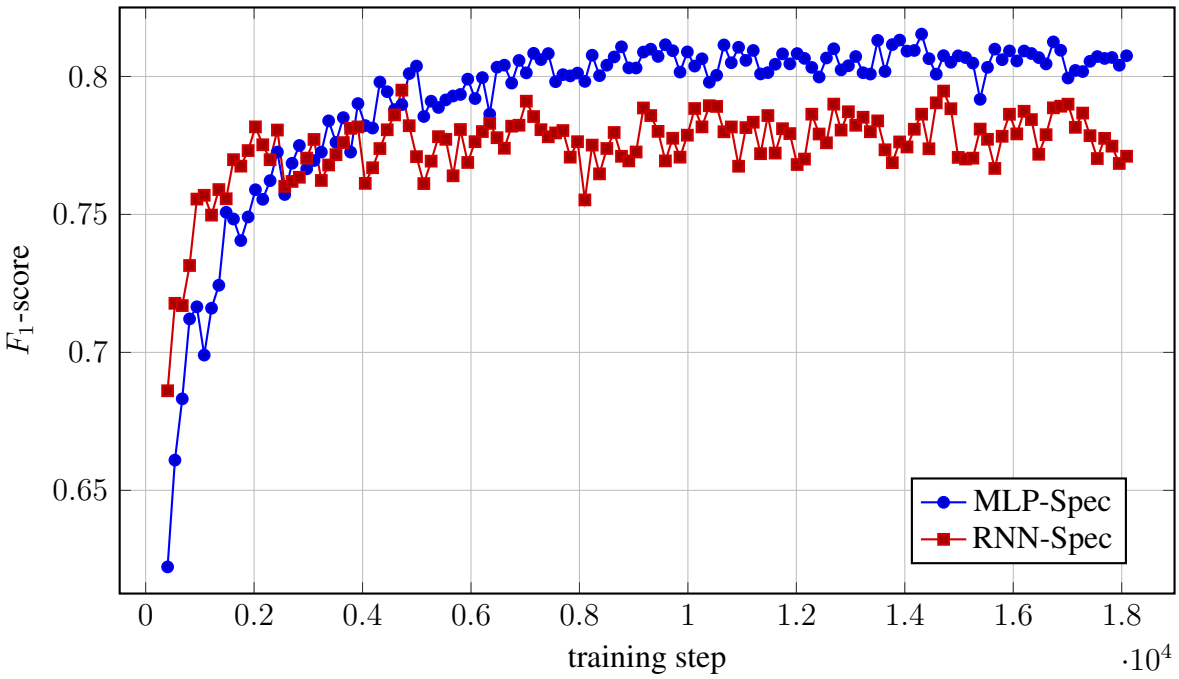
7.3.1.3 Model Comparison: MLP-Spec vs. RNN-Spec

When taken two best performing setups of both models, the winning RNN-Spec setup is the one with context length of `10` frames and GRU recurrent unit. Remaining parameters are

given by Table 12.

Tabul'ka 8: *Experimental setup: MLP-Spec vs. RNN-Spec on LabREC corpus.*

Model	**Batch size**	**Width**	**Depth**
MLP-Spec	64 frames	1000 units	1 layer
RNN-Spec	64 sequences	1000 units	1 layer



Obr. 33: *Performance of MLP-Spec vs. RNN-Spec on classical music.*

Clearly, RNN-Spec converges into less optimal solution than MLP-Spec, despite its faster acceleration. This is apparently caused by the same problem as in the experiment with different depths. By introducing additional recurrent layer on the input, size and capacity of the model has grown, while training corpus stayed the same size.

We suppose, that RNN-Spec was able to recognize specific structures in spectral changes over time, thanks to provided context, and overfit to their occurrences in training set. Our further work with larger corpus supports this theory.

7.3.2 Usual Chords

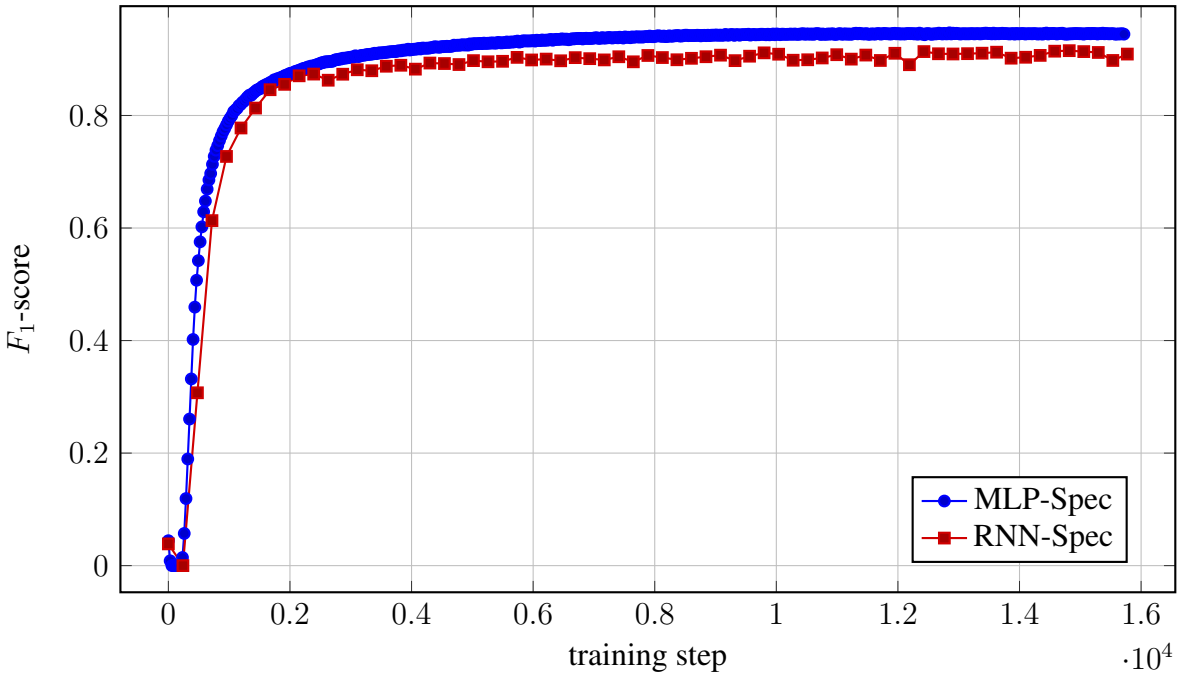
From the MAPS dataset, the subset of usual chords (UCHO) was selected for additional experiment. We reserved 1 chord for validation and 1 for test set from each chord category. This resulted in distribution of chord examples described by Table 9.

Tabul'ka 9: *Distribution of examples from subset MAPS_UCHO.*

MAPS_UCHO	**Train**	**Test**	**Valid**
Chord examples	1374	198	198
Minutes of audio	40	5	5

7.3.2.1 Model Comparison: MLP-Spec vs. RNN-Spec

We again evaluate our former models, although now on different data. Now with relatively balanced sets of hyperparameters as depicted in Table 10.



Obr. 34: *Performance of MLP-Spec vs. RNN-Spec on usual chords.*

Tabul'ka 10: *Experimental setup: MLP-Spec vs. RNN-Spec on MAPS_UCHO.*

Model	Threshold	Batch size	Width	Depth	Cell	Context
MLP-Spec	0.5	8000 fragments	100 units	1 layer	-	-
RNN-Spec	0.5	100 sequences	128 units	1 layer	GRU	10 fragments

Although relative rates of performance between concurrent models remains practically the same, significant improvement of maximum precision is exposed by both models, against classical music data.

The actual test set performance of validation top-performing model checkpoints was remarkable, in contrast to previous results, as stated in table 11.

Tabul'ka 11: *Test performance summary on F1 metric.*

	MLP-Spec	RNN-Spec
LabREC	75.32%	70.64%
MAPS_UCHO	93.52%	91.64%

This suggests the importance of data set and its structure. Since usual chords are repeated over dataset, just in different inversions and transpositions, the structural pattern of harmonic components is repeated as well and is then more easily learned. Unlike classical music, full of unique harmonic patterns and combinations.

7.3.3 Large-scale Training

In this chapter we mention some experiments performed on PIMIDE data corpus.

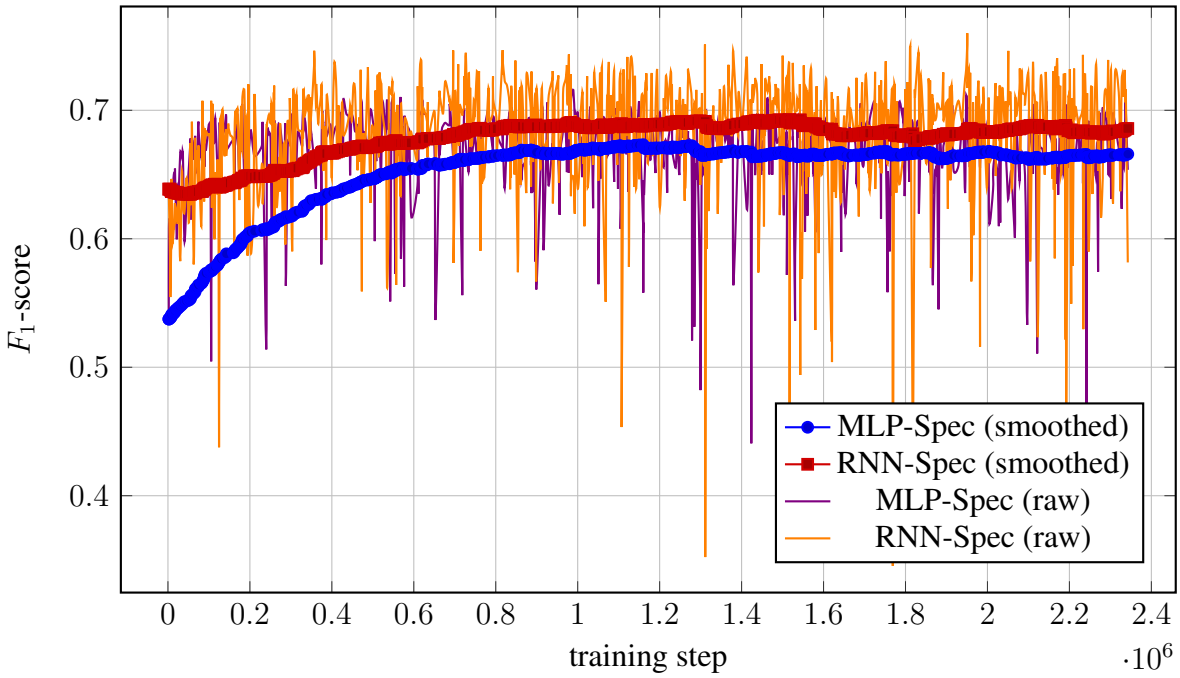
7.3.3.1 Model Comparison: MLP-Spec vs. RNN-Spec

For first experiment with large data corpus we take the two best performing setups of MLP-Spec and RNN-Spec and examine their performance in similar fasion as in 7.3.1.3.

Tabul'ka 12: *Experimental setup: MLP-Spec vs. RNN-Spec on PIMIDE corpus.*

Model	Batch size	Width	Depth	Context	Cell
MLP-Spec	100 frames	1000 units	1 layer	-	-
RNN-Spec	100 sequences	1000 units	1 layer	10 frames	GRU

After $\approx$ 5 days of training together and $\approx$ 2.1 millions of training steps per training session, we terminate the experiment, since validation performance curves make it clear to observe what was expected.



Obr. 35: *Performance of MLP-Spec vs. RNN-Spec on classical music.*

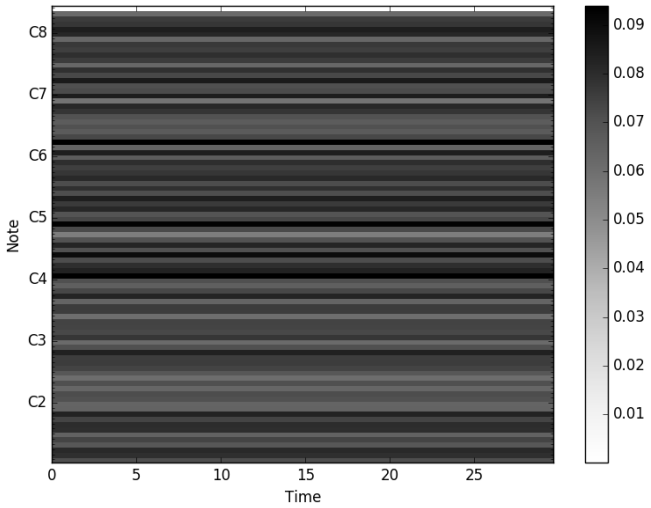
On Figure 35 we see huge oscillations of raw validation performance curves. This is clearly an opposite case than the comparison in 7.3.1.3, where due to lack of data, both models were overfit. On the contrary, in this experiment, we have too large data corpus for both models, therefore this is clearly an *underfitting* issue. Oscillating performance indicate attempts to absorb small amounts of knowledge from huge data corpus. This might be caused by proportions in sizes of models and data. Lowering learning rate or enlarging batch size could also help, however, one adds computational costs, while the other adds memory costs.

The slightly better performance of RNN-Spec also confirms our previous assumption that it has been overfit on the small training corpus due to higher modelling capacity. Now in case of underfitting, the capacity added by recurrent layer and provided context finally provides advantage to RNN-Spec over MLP-Spec.

7.3.3.2 Initial Training of WN4T

Upon construction of large-scale data corpus and adjustment of WN4T training job, during the development of evaluation module, we executed a 17 days long training session. Unfortunately, only 30 epochs were passed during this period, since the data corpus is large, and WN4T processing mechanics in high resolution over high-dimensional data, are also time-intensive.

After evaluation was implemented, we found that this corpus was also way too large even for the WN4T model. After all this time, model was stuck at predicting constant probability distributions across time, which means its predictions were not conditioned on input values changes, so it did not really ever start to converge.



Obr. 36: *Example predictions of non-converging WN4T.*

7.3.4 Additional Experiments

In this section we describe tests performed on the LabSYNTH dataset with all proposed architectures and finally summarize and compare all variants.

7.3.4.1 Bi-directional Recurrency

After implementation of BiRNN-Spec model, we trained this model along with MLP-Spec and RNN-Spec on LabSYNTH corpus. Main parameters[1] of training configuration are listed

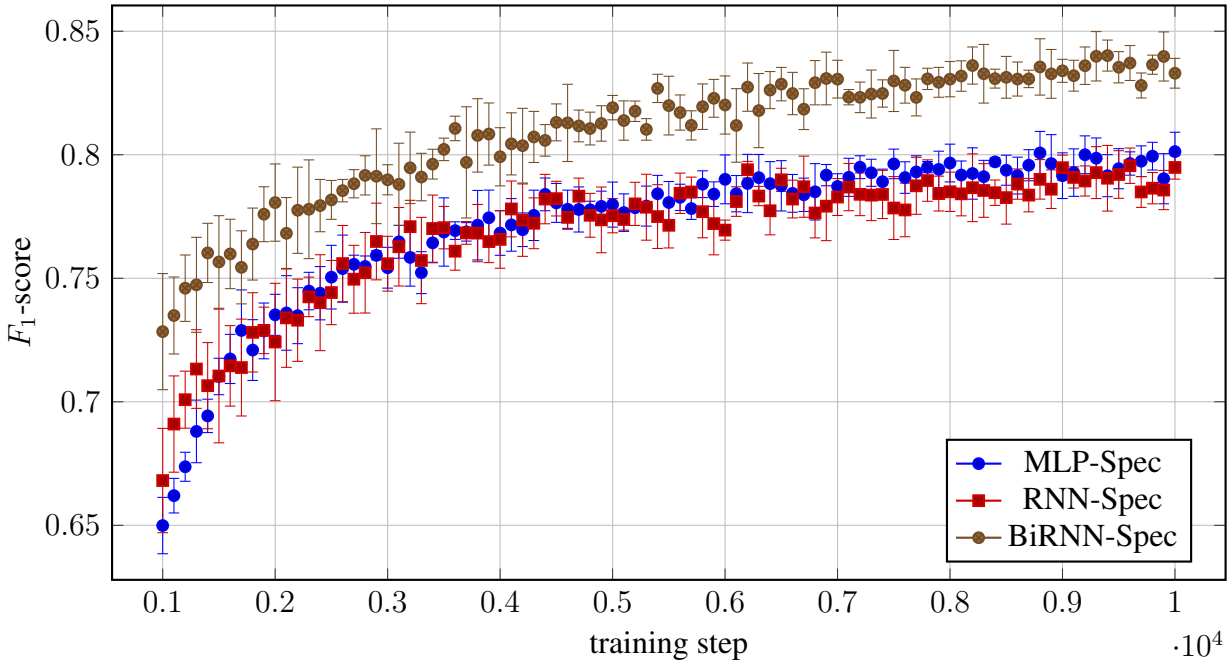
[1] Each element in square brackets describes single layer.

in Table 13 below.

Tabul'ka 13: *Parameters of* *-Spec *models trained on LabSYNTH.*

	MLP-Spec	**RNN-Spec**	**BiRNN-Spec**
batch size	500	100	100
sequence length	-	20	21
recurrent layers	-	[1000]	[1000]
recurrent activations	-	[tanh]	[tanh]
fully connected layers	[1000]	[1000]	[1000]
fully connected activations	[relu]	[relu]	[relu]

In this experiment, each model was trained in 5 independent sessions with randomly initialized weights. Validation performance curves of our models in Figure 37 are plotted from mean values of these sessions, while standard deviations are included in the plot.



Obr. 37: *Validation performance on synthesized data.*

Initial results make BiRNN-Spec appear as best improvement in our architecture design so far. However, in this case it is clear, that additional modelling capacity comes at cost

of additional computational complexity. Regardless, additional tests of this model will be necessary to unveil its properties in robust evaluation and comparison.

7.3.4.2 Gradual Training of WN4T

Finally, we found a method to train our WN4T model and evaluate it on the same data as other models, for comparison. However, we found it non-trivial to train this model for polyphonic texture recognition from scratch. Especially randomly initialized model. Therefore, we train this model gradually as follows.

1. First, pretrain WN4T on a corpus of monophonic piano notes with durations of 0.1, 0.5, 1, 3 and 7 seconds which gives 5×88 training files in total.
2. After convergence on this data, generate all different pairs of piano notes played as harmonic intervals with duration of 1 second.
3. This gives 3828 training files which are added to the existing corpus and the training continues until convergence on training data.
4. Start adding polyphonic data samples into training corpus, but maintain equal proportion to monophonic data samples.

After few training epochs on monophonic and interval data, and upon convergence on this data when model reaches reasonable training set error, we add LabSYNTH data to the training set.

Model was trained on `16000` Hz audio. The stack of dilation layers of our model is depicted in Table 14. It comprises of 33 layers and results in receptive field of 6143 audio samples.

Tabul'ka 14: WN4T *model layers setup in comparison experiment.*

Dilation factors of successive layers										
1	2	4	8	16	32	64	128	256	512	1024
1	2	4	8	16	32	64	128	256	512	1024
1	2	4	8	16	32	64	128	256	512	1024

During the training, sample size was set to 10000 consecutive samples. This is the number of time frame estimations generated within single training step, so it can be interpreted

as parallelization of the training. We left batch size, which defines actual number of such sequences in single training step, equal to 1, due to memory constraints of GPU we used for training.

Input layer is processing mono audio, therefore number of input channels is 1. Number of output and skip channels is 128 according to notes in MIDI standard. Between layers throughout the model, we used 256 residual and dilation channels.

Finally, we report and compare the test performance of our models trained on LabSYNTH corpus, which however, due to various sizes of train samples and batch sizes and constrained training times, resulted in different numbers of training epochs. Numbers in parentheses indicate the number of training epochs passed by given model in this comparison.

Tabul'ka 15: *Comparison of our models trained on LabSYNTH corpus.*

LabSYNTH	**MLP-Spec** (56)	**RNN-Spec** (12)	**BiRNN-Spec** (12)	**WN4T**[*] (13)
precision	82.26%	81.94%	**84.21%**	73.37%
recall	75.85%	74.70%	**78.98%**	37.84%
F1	78.93%	78.15%	**81.51%**	49.93%
Acc	65.19%	64.14%	**68.80%**	33.27%

Naturally, we used the best performing checkpoint for this evaluation, regarding our set of MLP based models, since we had 5 different checkpoints of each model, according to multiple training sessions described above.

7.3.5 Comparison to Previous Work

Additionally we compare our results to the reference approach [67] due to availability and detailed description of data set used in their experiments, which we collect and denote as LabCOMP.

After ≈ 3 training epochs we report frame level accuracy of our models . We acquired the published recordings from publication web site[2] and MIDI scores of remaining tunes from referenced site[3] according to table of MIDI compositions attached with the paper [67], which

[*]Although this model is included in the comparison, it is very different in nature and size from the other models and therefore it has been trained differently, as was described above.

[2]`https://labrosa.ee.columbia.edu/projects/piano/`

[3]`http://piano-midi.de/`

comprehensively describes used data set. We also used the same soundfont for synthesis as stated in the paper, namely Yamaha Disklavier grand piano.

Tabuľka 16: *Comparison to reference approach.*

Approach	Recorded (10)	Synthesized (25)	Combined (35)
Poliner and Ellis [67]	**56.5%**	**72.1%**	67.7%
Nam et al [63]	-	-	**72.5%**
Ryynanen and Klapuri [74]	41.2%	48.3%	46.3%
Marolt [57]	38.4%	40.0%	39.6%
MLP-Spec	56.2%	61.2%	59.2%
RNN-Spec	46.3%	56.1%	52.1%
BiRNN-Spec	54.5%	63.5%	59.9%

Due to time insufficiency, our models are not fully trained, neither fine-tuned on validation data of LabCOMP corpus, as opposed to concurrent approaches, so these results now serve just as a proof of concept.

In this work, we propose several novel approaches and dedicate our focus to their realization and iterative validation of subsequent design ideas. The search for optimal hyperparameters, such as model size and regularization, training parameters will be subject to future research, along with fine-tuning of these models on validation subset in order to prepare for proper evaluation and comparison.

Also, the notable difference in performance of our models between evaluations on LabSYNTH and LabCOMP datasets seems to follow the structural difference between those datasets. While LabCOMP is divided into training, validation and test subsets by grouping whole tunes into these 3, LabSYNTH cuts 3 pieces out of each tune and places those into subsets. Since repetition of musical themes and phrases is typical for classical compositions, LabSYNTH has higher probability to generate overlap of polyphonic textures between those subsets, simply by design.

7.4 Human-Level Evaluation

In order to gain deeper insight into performance of our methods, we often visualize and evaluate individual transcriptions on real-world data. Although this naturally belongs to the development and debugging process, we share some takeaways from this process in this chapter.

Additionally, we test the robustness of our method by comparing transcription results of digitally created audio snippet versus the same audio snippet reproduced and re-recorded with a microphone embedded in smartphone.

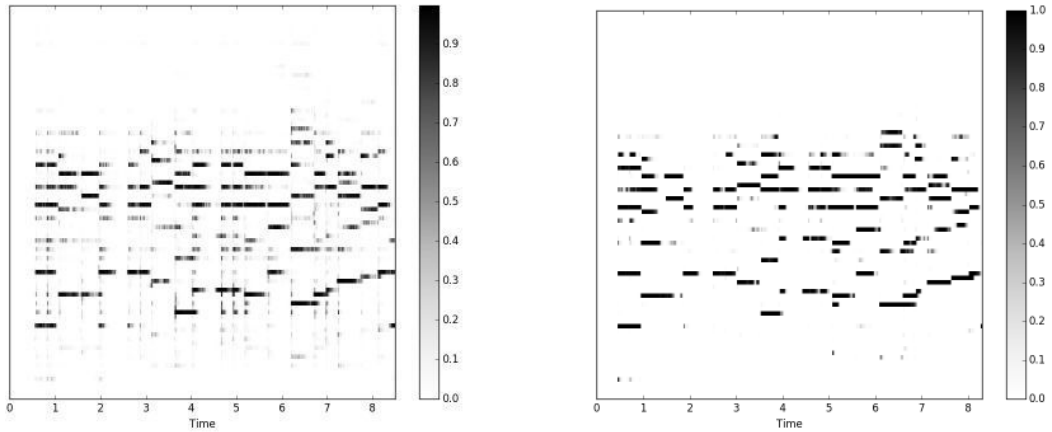
7.4.1 Role of Temporal Context

After initial experiments with simple MLP-Spec model, we noticed plenty of *noise* in its predictions. These are all the extra-short note predictions occurring with very little to none support from neighboring time frames and indicate that system is missing capacity for temporal modelling.

Some similar approaches previously used to employ Hidden Markov Model for temporal smoothing of classifier outputs such as [67, 63]. However, we want to rely purely on the neural network modelling capacity and thus approach this issue from the viewpoint of architecture design.

By introducing time context to each prediction sample, the support for note presence from neighboring time frames is encoded in the state of recurrent unit after the sequence leading to particular prediction is processed. In case of bi-directional RNN, this state encodes the sequence of time frames from both directions in time, which results in improved consistency in predicted note durations, as should be clear from Figure 38.

For this examination, the used piece of music is intro to *Freedom at Midnight* by *David Benoit*.

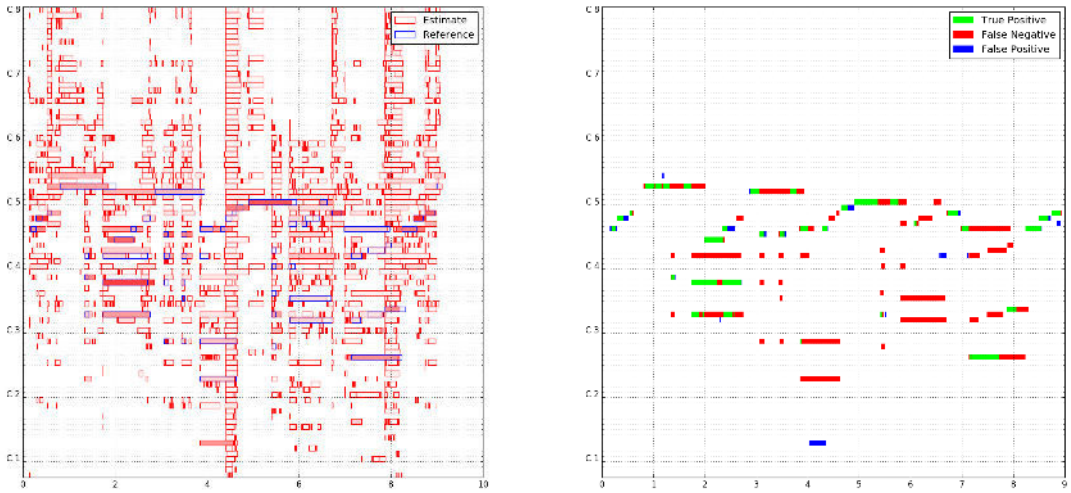


(a) MLP-Spec (b) BiRNN-Spec

Obr. 38: *Temporal smoothing by recurrent layer and input time context.*

7.4.2 Examining Gradually Trained WN4T

After multiple sessions of gradual training of our WN4T model, as described in 7.3.4.2, we examined its transcription abilities and fail modes on some audio snippets from different music excerpts from those included in training data.



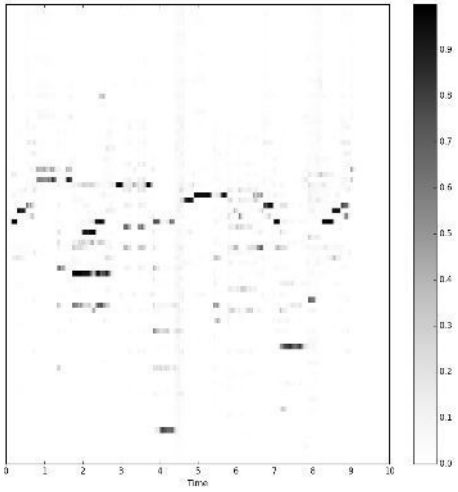
(a) Note certainties (b) Evaluation matrix

Obr. 39: *Evaluation and certainties of WN4T predictions*

On subfigure 39a at $\approx$ 4.5 seconds, there are 3 very short consecutive notes played chromatically, which results in audio frames containing multiple overlapping parts of ADSR envelopes of different notes. Response shows some very low certainty but at all different notes during this time period. This and also other "vertical peeksïn prediction matrix suggests, that this kind of confusion happens when multiple familiar waves get spotted in single sample as previously unseen example of wave snippet.

Additionally, the ability of WN4T to transcribe monophonic texture is obvious from subfigure 39b. While most of polyphony ends up getting only the strongest note transcribed, if any, certainty plot shows networks' familiarity with some harmonic intervals, possibly learned from the small data set of synthesized tunes from LabROSA.

We also plot raw predictions on this piece for reference of how noisy prediction certainties change over time.



Obr. 40: *Raw predictions of WN4T.*

For this examination, we used piece of intro to well-known jazz standard *Autumn Leaves*.

7.4.3 Testing Robustness of Proposed Method

We decided to also make an extra test for robustness of our method, based on real-world use case, while also checking for different parameters of our input data. We evaluate our currently most accurate transcription method, which is based on BiRNN-Spec model.

Following test is intended to give some indication about the robustness of our method against 3 parameters of audio piece being transcribed.

1. Musical genre of the tune.
2. Tempo of the tune.
3. Quality of audio recording.

We check for 1 by choosing non-classical compositions for the test of model trained on purely classical music corpus. Next, we check for 2 by choosing two compositions with opposite extremes in tempo parameter. Finally, we check for 3 by introducing two variants of each piece excerpt.

First variation is classically synthesized audio (columns *synthesis* in the table), while second variation is created by reproduction of synthesized audio through speakers[1] and

[1] We used speakers model `Genius SP-HF1800A` for audio reproduction.

subsequent re-recording of reproduced sound with smartphone[1] built-in microphone (columns *recording* in the table).

For both pieces we use performances by Japanese jazz virtuoso and composer *Hiromi Uehara*: a slow tempo tune *Place to Be*[2], and a fast tempo tune *The Tom and Jerry Show*[3]. We trim first 100 seconds of each MIDI file, synthesize the resulting piece with previously described soundfont `Nice-Keys-Extreme` and replay-record resulting audio files.

We further time-align the recordings to labels and test our BiRNN-Spec model checkpoint, which was trained according to description in 7.3.4.1, on this data configuration[4].

Tabuľka 17: *Testing performance of BiRNN-Spec model on different data variations.*

Tune	**Place to Be (slow)**		**Tom and Jerry (fast)**	
Variant	**synthesis**	**recording**	**synthesis**	**recording**
precision	89.40%	**91.00%**	**72.36%**	61.92%
recall	**61.25%**	37.98%	**47.92%**	24.31%
F1	**72.69%**	53.60%	**57.65%**	34.92%
Acc	**57.10%**	36.61%	**40.50%**	21.15%

According to parameters of this test and results presented in Table 17, we observe the outcomes and make following notes.

1. The size of test sample is too small to enable us make any conclusions about robustness against genre variations, however, average performance drop against test results on classical music data is $\approx 15\%$ which is significant.
2. Results indicate, that harmonic content together with tempo of the tune, in terms of average frequency and duration of notes, are both important parameters of dataset variety. They should be considered when approaching model robustness issues.
3. Quality of audio recording seems to be another important parameter of data variety. Apparently, recall of many notes is based on their timber, since it drops significantly with lowering audio quality, meaning that many new False Negatives are introduced.

[1] We used smartphone model `Samsung Galaxy Ace 4` to create the recording.

[2] Acquired from `https://musescore.com/user/6785046/scores/1718291`.

[3] Acquired from `http://www.mediafire.com/file/4kc3lzniptem1c1/Hiromi+Uehara+-+The+Tom+and+Jerry+Show.mid`.

[4] This data configuration is also stored on path `/data/hiromi/` on electronic medium attached to this work. Results of this test including metrics and medial content are also stored on path `/tests/robustness/hiromi/`.

Since sound quality is one of factors with significant influence to note color, the recognition system must become more robust to their variations. Although we tried to accomplish this by training on multiple-soundfont based synthesized data, we admit, that according to the size of data corpus we trained the evaluated model on, it might still be overfit.

There are also numerous other means to introduce variations to timbral content of training data which we might take into focus during our future work. Some examples might include application of sound effects, filters and equalization with intention to mimic imperfections of real-world recording setups. This might lead to creation of dataset with much higher representativeness of expected production data.

8 Conclusion

Although we have initially designated quite ambitious goal of approaching multiple music transcription subtasks simultaneously, as stated in Section 5, due to problem complexity and amount of resources available, we kept our initially stated priorities and stayed at the number one task of *detecting notes in polyphony*, for the scope of this work.

We proposed several methods to approach the problem of Multiple Fundamental Frequency Estimation using artificial neural networks and deep learning methods. With focus on network architecture design, we realized all of our proposals, evaluated and compared them to concurrent approaches and amongst each other.

Currently the best performing amongst our approaches appears to be simple Multi-Layer Perceptron model augmented by a single bi-directional recurrent layer. This is also our latest approach and thus it still requires more exhaustive evaluation.

When compared to state-of-the-art approaches, our models have quite some limitations in terms of evaluation performance. However, their full potential may have not been unveiled yet. At first, our models could gain capacity just by increasing size and introducing regularization techniques, which is pretty straightforward approach that actually has not been realized yet.

First branch of this work is based on iterative and incremental design of neural network architecture tailored to learn the know-how related to music transcription process from spectral domain. This branch has started from scratch and appears to be the more successful one at the moment. Some possible future directions here include involvement of convolution layers on the input and application of recurrency rather to higher level features. Additionally, our spectrogram architecture might grow deeper by employing residual or skip connections across its main building blocks.

The second main branch builds on sophisticated deep architecture with capacity to learn new feature space directly from raw audio. Since this is a very dense data representation, it is challenging for processing, in terms of time costs. However, there are countless options to extend our current state of work with WaveNet architecture.

Initially, we could use adaptive threshold instead of static one. Implementation options include for example use of binary activations on the output layer, based on threshold values learned by biases for each output unit individually.

Next, it would be very interesting to visualize activations of hidden layers of WaveNet, since the model is actually forced to invent its own method for extraction of spectral features,

during the training. By feeding noise to the trained network and visualizing hidden layers activations, similarly to famous DeepDream[1], it might be very useful to actually see what is going on within deep representations of this large, complex network architecture.

Interesting option is also to further iteratively build on current WaveNet architecture by introducing slight changes with goal to optimize it for task of transcription. Also, significance of context size and receptive field size for transcription task has not been examined yet.

We also consider our idea to use local conditioning of WaveNet as a placeholder to supply the network with spectral magnitude data and provide it along with raw audio, which could fundamentally unburden the network from the cognitive load of spectral analysis, or at least provide significant help with the task, very interesting. This extra capacity could improve the network performance or help it to learn perform multiple transcription subtasks simultaneously.

Additionally, global conditioning could be used to define e.g. sound source which the network should focus on during the transcription. Actually, one interesting initiative already gave rise to a comprehensive data set of all possible sound sources and sound colors used in modern music production, with a goal to train the networks as much as it goes about the timber, and possibly even provide a tool for live performance based on neural audio synthesis[2].

Fortunately, deep learning will obviously continue its evolution in a rapid tempo and will find various applications in possibly every domain, leaving no excuse for audio and music.

[1] `https://github.com/google/deepdream`
[2] `https://magenta.tensorflow.org/nsynth`

Literatúra

[1] Backpropagation algorithm. http://colah.github.io/posts/2015-08-Backprop/. [Visited 10.05.2016].

[2] Deepmind's blog post on wavenet. https://deepmind.com/blog/wavenet-generative-model-raw-audio/. [Visited 10.11.2016].

[3] Image example for feedforward neural network. https://en.wikibooks.org/wiki/Artificial_Neural_Networks/Feed-Forward_Networks. [Visited 10.05.2016].

[4] Image example for spectral envelope of a signal. https://www.researchgate.net/figure/247773842_fig7_Figure-17-Spectral-envelope-of-a-signal. [Visited 01.05.2016].

[5] Image example for time envelope of a signal. http://www.dragonflyalley.com/constructionJHtrapezoidVCA.htm. [Visited 01.05.2016].

[6] Introduction to artificial neural networks. http://www.theprojectspot.com/tutorial-post/introduction-to-artificial-neural-networks-part-1/7. [Visited 10.11.2016].

[7] Neural network zoo. http://www.asimovinstitute.org/neural-network-zoo/. [Visited 10.11.2016].

[8] A practical introduction to deep learning with caffe and python. http://adilmoujahid.com/posts/2016/06/introduction-deep-learning-python-caffe/. [Visited 10.11.2016].

[9] Spectral envelopes in sound analysis and synthesis. http://recherche.ircam.fr/anasyn/schwarz/da/specenv/3_3Spectral_Envelopes.html. [Visited 01.05.2016].

[10] Time envelope in musical sound.
http://www.britannica.com/science/envelope-sound.
[Visited 01.05.2016].

[11] Understanding lstms.
http://colah.github.io/posts/2015-08-Understanding-LSTMs/.
[Visited 10.11.2016].

[12] Visualizations of rnn units.
http://kvitajakub.github.io/2016/04/14/rnn-diagrams/.
[Visited 10.11.2016].

[13] Vipul Arora and Laxmidhar Behera. Musical source clustering and identification in polyphonic audio. *IEEE Transactions on Audio, Speech and Language Processing*, 22(6):1003–1012, 2014.

[14] Roland Badeau. MAPS - A piano database for multipitch estimation and automatic transcription of music MAPS - Base de données de sons de piano pour l ' estimation de fréquences fondamentales multiples et la transcription automatique de la musique Département Traitement d. 2010.

[15] P. Balazs, M. Doerfler, F. Jaillet, N. Holighaus, and G. Velasco. Theory, implementation and applications of nonstationary Gabor frames. *Journal of Computational and Applied Mathematics*, 236(6):1481–1496, 2011.

[16] Juan P. Bello, Giuliano Monti, and Mark Sandler. Techniques for automatic music transcription. *International Symposium on Music*, pages 1–8, 2000.

[17] E Benetos, S Dixon, D Giannoulis, H Kirchhoff, and A Klapuri. Automatic music transcription: challenges and future directions. *Journal of Intelligent Information Systems*, 41(3):407–434, 2013.

[18] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy Layer-Wise Training of Deep Networks. *Advances in Neural Information Processing Systems*, 19(1):153, 2007.

[19] H Bourlard and Y Kamp. Auto-Association by Multilayer Perceptrons and Singular Value Decomposition. *Biol. Cybern*, 59:291–294, 1988.

[20] Albert Bregman and Stephen Mcadams. Musical Streams. *Computer Music Journal*, 3(4):26–43, 2014.

[21] Alfonso Perez Carrillo and Marcelo M. Wanderley. Learning and Extraction of Violin Instrumental Controls from Audio Signal. *Proceedings of the 2nd International ACM Workshop on Music Information Retrieval with User-Centered and Multimodal Strategies (MIRUM)*, pages 25–30, 2012.

[22] A. T. Cemgil, H. J. Kappen, and D. Barber. A generative model for music transcription. *IEEE Transactions on Audio, Speech, and Language Processing*, 14(2):679–694, March 2006.

[23] C Chafe, D Jaffe, K Kashima andB. Mont-Reynaud, and J Smith. Techniques for Note Identification in Polyphonic Music, 1985.

[24] Chris Chafe and David Jaffe. Source Separation and Note Identification in Polyphonic Music, 1986.

[25] Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. *arXiv*, pages 1–9, 2014.

[26] W. T. Cochran, J. W. Cooley, D. L. Favin, H. D. Helms, R. A. Kaenel, W. W. Lang, G. C. Maling, D. E. Nelson, C. M. Rader, and P. D. Welch. What is the fast fourier transform? *Proceedings of the IEEE*, 55(10):1664–1674, Oct 1967.

[27] D. Deutsch. *The Psychology of Music*. Academic Press Series. Academic Press, 1999.

[28] Simon Dixon. On the computer recognition of solo piano music. *Proceedings of Australasian Computer Music Conference*, pages 31–37, 2000.

[29] Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990.

[30] Valentin Emiya, Roland Badeau, and Bertrand David. Multipitch estimation of piano sounds using a new probabilistic spectral smoothness principle. *Trans. Audio, Speech and Lang. Proc.*, 18(6):1643–1654, August 2010.

[31] R. Erickson. *Sound Structure in Music*. University of California Press, 1975.

[32] Daniel R Franklin and Joe E Chicharo. Paganini - a music analysis and recognition program. pages 22–25, 1999.

[33] T. Gagnon, S. Larouche, and R. Lefebvre. A neural network approach for preclassification in musical chords recognition. *The Thrity-Seventh Asilomar Conference on Signals, Systems & Computers, 2003*, 2:2106–2109, 2003.

[34] ZOUBIN GHAHRAMANI. An Introduction To Hidden Markov Models And Bayesian Networks, 2001.

[35] Simon Godsill and Manuel Davy. Bayesian harmonic models for musical pitch estimation and analysis. *IEEE International Conference on Acoustics Speech and Signal Processing*, pages II–1769–II–1772, 2002.

[36] M. Goto. A predominant-f0 estimation method for cd recordings: Map estimation using em algorithm for adaptive tone models. In *Acoustics, Speech, and Signal Processing, 2001. Proceedings. (ICASSP '01). 2001 IEEE International Conference on*, volume 5, pages 3365–3368 vol.5, 2001.

[37] Philippe Hamel and Douglas Eck. Learning Features from Music Audio with Deep Belief Networks. *International Society for Music Information Retrieval Conference (ISMIR)*, (Ismir):339–344, 2010.

[38] Ge Hinton, Ge Hinton, Tj Sejnowski, and Tj Sejnowski. Learning and relearning in Boltzmann machines. *MIT Press, Cambridge, Mass., 1986*, 1(January):283–317, 1986.

[39] Sepp Hochreiter, S Hochreiter, Jürgen Schmidhuber, and J Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–80, 1997.

[40] E J Humphrey and J P Bello. Rethinking Automatic Chord Recognition with Convolutional Neural Networks. *Machine Learning and Applications (ICMLA), 2012 11th International Conference on*, 2:357–362, 2012.

[41] E. J. Humphrey, T. Cho, and J. P. Bello. Learning a robust tonnetz-space transform for automatic chord recognition. In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 453–456, March 2012.

[42] Yoshua Bengio Ian Goodfellow and Aaron Courville. Deep learning. Book in preparation for MIT Press, 2016.

[43] Natasha Jaques, Shixiang Gu, Richard E. Turner, and Douglas Eck. Tuning Recurrent Neural Networks with Reinforcement Learning. *Thesis*, pages 410–420, 2016.

[44] Li Lei Jiayin Sun, Haifeng Li. KEY DETECTION THROUGH PITCH CLASS DISTRIBUTION MODEL AND ANN Jiayin Sun, Haifeng Li, Li Lei School of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001, China. (2), 2009.

[45] Judith C. Brown. Calculation of a Constant–Q spectral transform, 1990.

[46] Kyle L. Kashima and Bernard Mont-Reynaud. The bounded-Q frequency transform. pages 1–10, 1985.

[47] K. Kashino and S. J. Godsill. Bayesian estimation of simultaneous musical notes based on frequency domain modelling. In *2004 IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 4, pages iv–305–iv–308 vol.4, May 2004.

[48] Kunio Kashino and Norihiro Hagita. A Music Scene Analysis System with the MRF-Based Information Integration Scheme. (Figure 1):725–729, 1996.

[49] Youngmoo Kim, Erik Schmidt, and Lloyd Emelle. Moodswings: A collaborative game for music mood label collection. In *Proceedings of the 9th International Conference on Music Information Retrieval*, pages 231–236, Philadelphia, USA, September 14-18 2008. `http://ismir2008.ismir.net/papers/ISMIR2008_257.pdf`.

[50] Diederik P Kingma and Jimmy Lei Ba. Adam : A method for stochastic optimization. *ICLR*, pages 1–15, 2015.

[51] A. P. Klapuri. A perceptually motivated multiple-f0 estimation method. In *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics, 2005.*, pages 291–294, Oct 2005.

[52] Edith Law and Luis Von Ahn. Input-Agreement : A New Mechanism for Collecting Data Using Human Computation Games. *Proc. SIGCHI Conference on Human Factors in Computing Systems*, pages 1–10, 2009.

[53] Yann Lecun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324, 1998.

[54] Yann A. LeCun, Leon Bottou, Genevieve B. Orr, and Klaus Robert Muller. Efficient backprop. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7700 LECTU:9–48, 2012.

[55] Honglak Lee, Pt Pham, Y Largman, and Ay Ng. Unsupervised feature learning for audio classification using convolutional deep belief networks. *Nips*, pages 1–9, 2009.

[56] Megumi Maekawa, Kazuhiko Takahashi, and Masafumi Hashimoto. Remarks on computational emotion classification from physiological signal - Evaluation of how jazz music chord progression influences emotion. *International Conference on Intelligent Systems Design and Applications, ISDA*, (C):967–972, 2012.

[57] M. Marolt. A connectionist approach to automatic transcription of polyphonic piano music. *IEEE Transactions on Multimedia*, 6(3):439–449, June 2004.

[58] Matija Marolt and Student Member. Transcription of polyphonic piano music with neural networks. 11:512–515, 2000.

[59] K D Martin. A Blackboard System for Automatic Transcription of Simple Polyphonic Music. (385), 1996.

[60] Brian McFee, Matt McVicar, Colin Raffel, Dawen Liang, Oriol Nieto, Eric Battenberg, Josh Moore, Dan Ellis, Ryuichi YAMAMOTO, Rachel Bittner, and et al. librosa: 0.4.1, Oct 2015.

[61] Matt McVicar, Raúl Santos-Rodríguez, Yizhao Ni, and Tijl De Bie. Automatic chord estimation from audio: A review of the state of the art. *IEEE Transactions on Audio, Speech and Language Processing*, 22(2):556–575, 2014.

[62] Tomas Mikolov, Greg Corrado, Kai Chen, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. *Proceedings of the International Conference on Learning Representations (ICLR 2013)*, pages 1–12, 2013.

[63] J Nam, J Ngiam, Honglak Lee, and Malcolm Slaney. A Classification-Based Polyphonic Piano Transcription Approach Using Learned Feature Representations. *Ismir*, (Ismir):175–180, 2011.

[64] Juhan Nam, Jorge Herrera, Malcolm Slaney, and eus Smith. Learning Sparse Feature Representations for Music Annotation and Retrieval. *International Conference on Music Information Retrieval*, (Ismir):565–570, 2012.

[65] Julien Osmalskyj, Marc Van Droogenbroeck, and Jean-jacques Embrechts. Neural networks for musical chords recognition. *Actes des Journées d'Informatique Musicale JIM 2012*, (May):39–46, 2012.

[66] Graham E Poliner and Daniel P W Ellis. IMPROVING GENERALIZATION FOR POLYPHONIC PIANO TRANSCRIPTION LabROSA , Dept . of Electrical Engineering Columbia University , New York NY 10027 USA. pages 86–89, 2007.

[67] Graham E Poliner and Daniel P W Ellis Labrosa. A Discriminative Model for Polyphonic Piano Transcription. pages 1–16, 2006.

[68] Rohit Prabhavalkar, Raziel Alvarez, Carolina Parada, Preetum Nakkiran, and Tara N Sainath. AUTOMATIC GAIN CONTROL AND MULTI-STYLE TRAINING FOR ROBUST SMALL-FOOTPRINT KEYWORD SPOTTING WITH DEEP NEURAL NETWORKS Google Inc ., Mountain View , USA ; 2 University of California , Berkeley , Department of EECS , USA. *Proceedings of International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE*, (i):2–6, 2015.

[69] Lawrence R Rabiner and Ronald W Schafer. *Introduction to Digital Speech Processing*, volume 2. 2007.

[70] Colin Raffel and Daniel P W Ellis. INTUITIVE ANALYSIS , CREATION AND MANIPULATION OF MIDI DATA WITH pretty _ midi. 2014.

[71] Colin Raffel, Brian Mcfee, Eric J. Humphrey, Justin Salamon, Oriol Nieto, Dawen Liang, and Daniel P. W. Ellis. mir_eval: A Transparent Implementation of Common MIR Metrics. *Proc. of the 15th International Society for Music Information Retrieval Conference*, pages 367–372, 2014.

[72] Christopher Raphael. Automatic transcription of piano music. *CEUR Workshop Proceedings*, 379, 2008.

[73] F Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386–408, 1958.

[74] M. P. Ryynanen and A. Klapuri. Polyphonic music transcription using note event modeling. In *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics, 2005.*, pages 319–322, Oct 2005.

[75] Christian Schörkhuber and Anssi Klapuri. Constant-Q transform toolbox for music processing. *7th Sound and Music Computing Conference*, (JANUARY):3–64, 2010.

[76] Jan Frederik Schouten. The perception of timbre. In *Reports of the 6th International Congress on Acoustics*, volume 76, page 10, 1968.

[77] Christian Schörkhuber, Anssi Klapuri, and Alois Sontacchi. Audio pitch shifting using the constant-q transform. *J. Audio Eng. Soc*, 61(7/8):562–572, 2013.

[78] Alexander Sheh and D.P.W. Ellis. Chord segmentation and recognition using EM-trained hidden Markov models. *Proc. ISMIR*, pages 185–191, 2003.

[79] A Shmilovici. Data Mining and Knowledge Discovery Handbook. *Data Mining and Knowledge Discovery*, pages 257–267, 2005.

[80] Julius O. Smith. Digital Audio Resampling Home Page. *Center for Computer Research in Music and Acoustics (CCRMA), Stanford University*, pages 1–20, 2002.

[81] Julius O. Smith. Digital Audio Resampling Home Page. *Center for Computer Research in Music and Acoustics (CCRMA), Stanford University*, pages 1–20, 2002.

[82] Daylin Troxel. Music transcription with a convolutional neural network. MIREX, 2016.

[83] George Tzanetakis. *Signal Processing Methods for Music Transcription (review)*, volume 32. 2004.

[84] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. WaveNet: A Generative Model for Raw Audio. pages 1–15, 2016.

[85] Matthias Varewyck, Johan Pauwels, and Jean-Pierre Martens. A novel chroma representation of polyphonic music based on multiple pitch tracking techniques. *Proceeding of the 16th ACM international conference on Multimedia - MM '08*, page 667, 2008.

[86] Francisco Fernandez De Vega. A Novel Approach to Automatic Music Transcription Using Electronic Synthesis and Genetic Algorithms. *Most*, pages 2915–2922, 2007.

[87] A. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13(2):260–269, April 1967.

[88] Xin Xu, Haibo He, Dongbin Zhao, Shiliang Sun, Lucian Busoniu, and Simon X Yang. Machine Learning with Applications to Autonomous Systems. 2015:2–4, 2015.

A Technical Documentation

In this section, we present details of technical realization of our method with focus on selected aspects of our solution.

A.1 Project Structure

This project is organized into modules of python scripts. It also provides some Jupyter Notebooks for purposes of functionality demonstration or fast prototyping in development use cases.

Root directory of this project is located in directory `/source` on electronic medium attached to this work. It is structured into logical units by directories as described below.

`data/` - data sets and subsets organized into folders and subfolders.

`sf2/` - directory containing SF2 format audio sample libraries used for synthesis.

`logs/` - logs from TensorFlow training sessions and saved trained model checkpoints.

`prep/` - module of utility scripts for data preparation and preprocessing. Also some obsolete code for manual preprocessing into pickle files is included in files:

`pickle_routines.py` - set of functions for audio and MIDI data preprocessing and storage.

`prepare_data.ipynb` - development notebook for preprocessing methods.

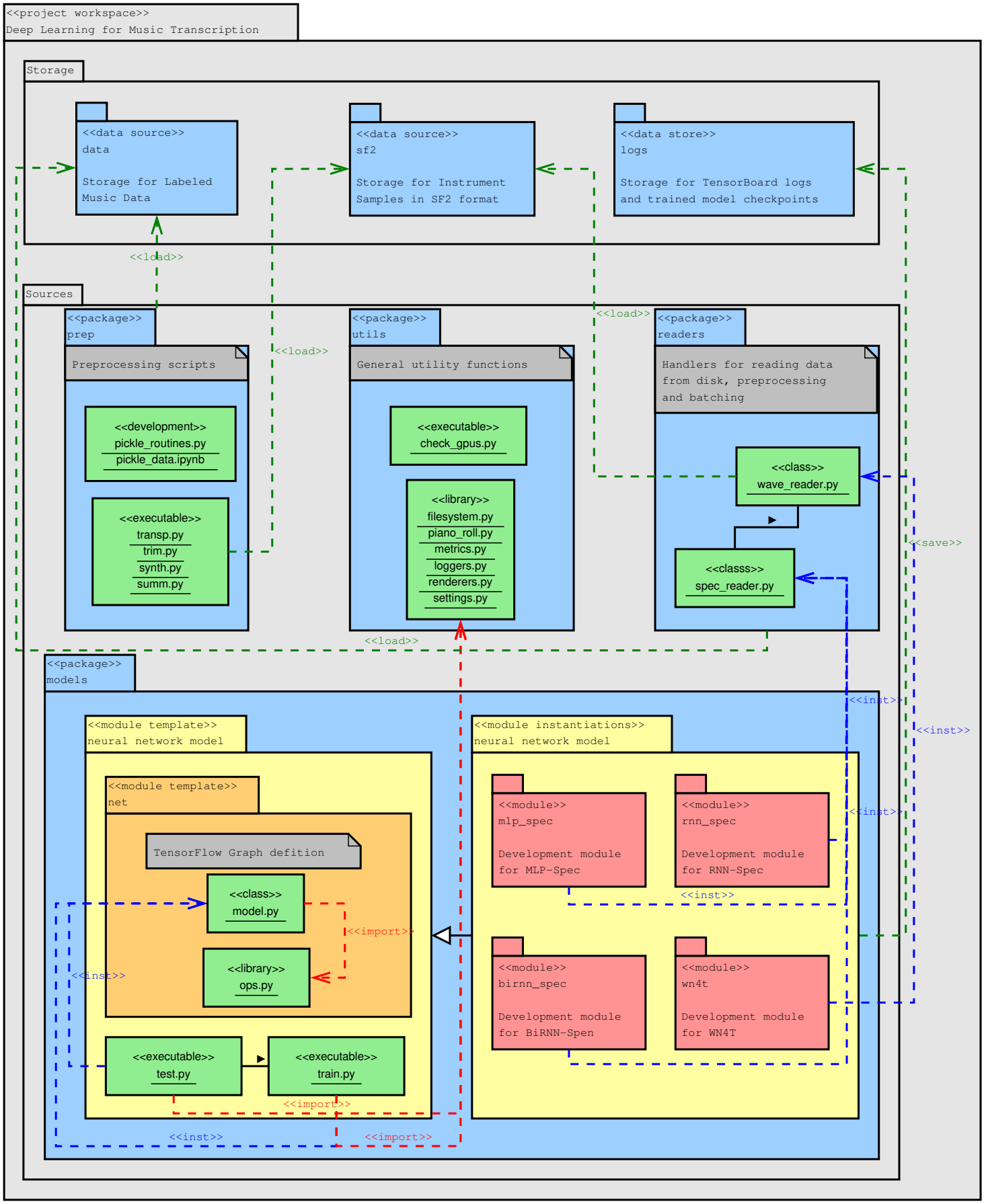
`pickle_data.ipynb` - demonstration notebook with examples of pickling, saving, loading and visualizing prepared data.

`readers/` - module containing readers for multi-threaded data loading and preprocessing, used by executable training and testing scripts of different models.

`utils/` - module that aggregates source files of various utility functions imported by other modules and used across the project.

`models/` - training and usage implementation of neural network models in TensorFlow. Also, obsolete code for training on data stored in pickle format is included in notebooks `mlp_spec.ipynb` and `rnn_spec.ipynb`.

On Figure 41 is outlined structure of our implementation by means of UML package and class diagram components, although with extensions in terms of custom stereotypes.



Obr. 41: *Diagram illustrating project structure and component dependencies.*

A.2 Packages and Modules

Since we use custom notation for several aspects of our solution organization, we further provide some context to the above diagram. We divide the project sources into packages and modules according to their purpose. We implement several executable python programs, which follow standard CLI interaction behavior. We also try to follow established code style conventions[1].

A.2.1 Preprocessing

This package, located in directory `prep/`, contains 4 executable python scripts serving for dataset manipulation and analysis.

`transp.py` - our 12-fold transposition algorithm, already mentioned in 7.2.2.

`trim.py` - trimming of given time interval from MIDI file collection and saving new collection with same file names into arbitrary location.

`synth.py` - synthesis of WAV audio provided MIDI data collection and valid paths to SF2 file indexed through file `soundfonts.json` also located in this folder.

`summ.py` - summarization of dataset statistics by parsing events from MIDI files and counting corresponding WAV files. Requires each MIDI file to have at least 1 respective WAV file in same directory with same file name and optional postfix. This dataset file naming structure is implicitly created by `synth.py` and is also required by data readers to work properly during training.

Additionally, Jupyter Notebook `usage.ipynb` is provided for usage demonstration of these data processing scripts.

Package also contains open-source implementation for computation of inverse CQT `icqt.py` for development purposes, and demonstration notebook `sample_check.ipynb` for examination of preprocessed data.

A.2.2 Utility Functions

In directory `prep/`, the executable script `check_gpus.py` serves simply for availability of GPU devices and proper configuration of tensorflow-gpu installation.

All other scripts in this package are organized into python module and their parts are importable by external scripts. Main purpose of this library is code re-usability. Source file `filesystem.py` contains routine functions for filesystem manipulation used widely across project. By simply importing piece of functionality from common location, we prevent code redundancy and enhance code sustainability.

[1] `https://www.python.org/dev/peps/pep-0008/`.

Rest of the source files provide functionality respective to their names, thus namely for piano roll data structure manipulation, metrics calculation, tensorboard logging routines, evaluation-level audio and image rendering routines and training session settings definition. These components are mainly imported by training and testing scripts of neural network models implementations.

A.2.3 Data Readers

Data readers have been aggregated into separate package for the sake of future reusability and unification, since their functionalities overlap. They might be refactored for use with some design patterns in the future.

Currently, `spec_reader.py` implements `SpecReader` class which is sufficient data processing worker for all three *-Spec models, provided modest but sufficient parametrization.

For training of WN4T model, `wave_reader.py` provides multiple data acquisition methods. First is traditional loading of audio from disk files. Alternatively, notes are generated with many degrees of randomization and synthesized on the fly, which gives rise to training method with potentially infinite training data variance. That is also why this program loads SF2 data source.

A.2.4 Neural Network Models

Since we implement four different neural network models into individual modules, we used somewhat abusive representation in our diagram. By generalization relationship between *module instantiation* and *module template* neural network model packages, we denote that each individual module instantiation (`{mlp,rnn,birnn}_spec` and `wn4t`) has the same structure, contents and relationships to other components in the diagram, as is represented by *module template* package.

Further, by «`inst`» stereotype, we denote *instantiation* of destination component by source component, respective to the arrow direction.

Each module of specific neural network model contains executable python CLI programs for training and testing, both richly parametrizable through program arguments and external file.

In following diagram, we try to visualize fundamental aspects of model training mechanism, with focus on data processing and interactions between trainer and data reader

thread.

A.3 Online Data Processing with Multi-Threading

Previously, we performed one-time preprocessing and saved prepared data into single file, which was whole loaded into operational memory during the training process. Due to increasing demands on size of training data throughout the work, we had to switch our data processing method to more scalable one.

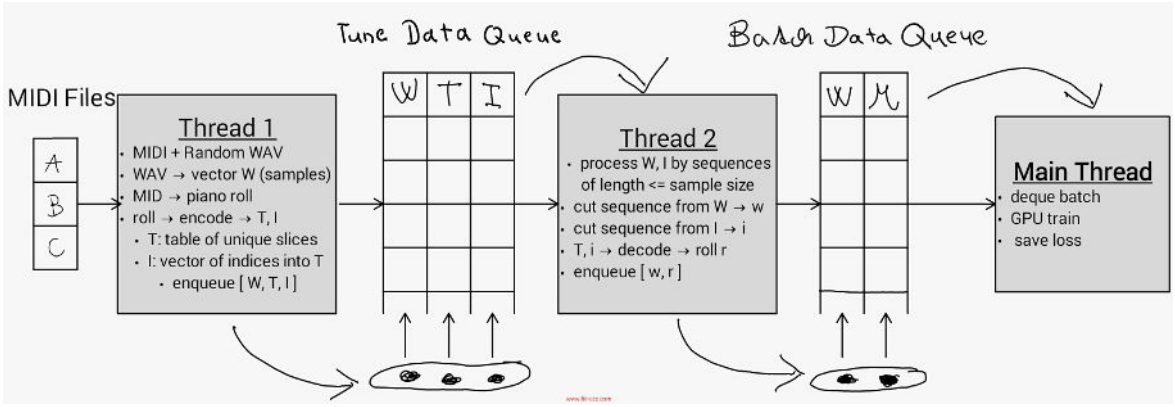
As depicted on Figure 43 while main thread executes training and evaluation on GPU, reader threads effectively use free CPU time to preprocess data and prepare training samples into TensorFlow Queue[1]. Since *enqueue* and *dequeue* are blocking operations, if data preprocessing speed is not in balance with training speed, the faster process waits for the slower one on these Queue operations. More detailed description of specific data reading variants follows.

A.3.1 Waveform Data Reading

Since we train WN4T model on $16kHz$ temporal resolution data, the piano roll labels in this resolution result in increase of memory requirements by factor of 128 for each training example snippet.

Therefore, we get rid of the redundancy of piano roll representation by simply constructing table of unique *polyphony texture vectors* and further keep only vector of indices into this table. This way, we reduce the memory requirements increase down to factor of 1 from factor of previous 128. We further optimize this by subsampling the vector of indices.

[1] `https://www.tensorflow.org/programmers_guide/threading_and_queues`.



Obr. 42: *Diagram illustrating data processing for WN4T training.*

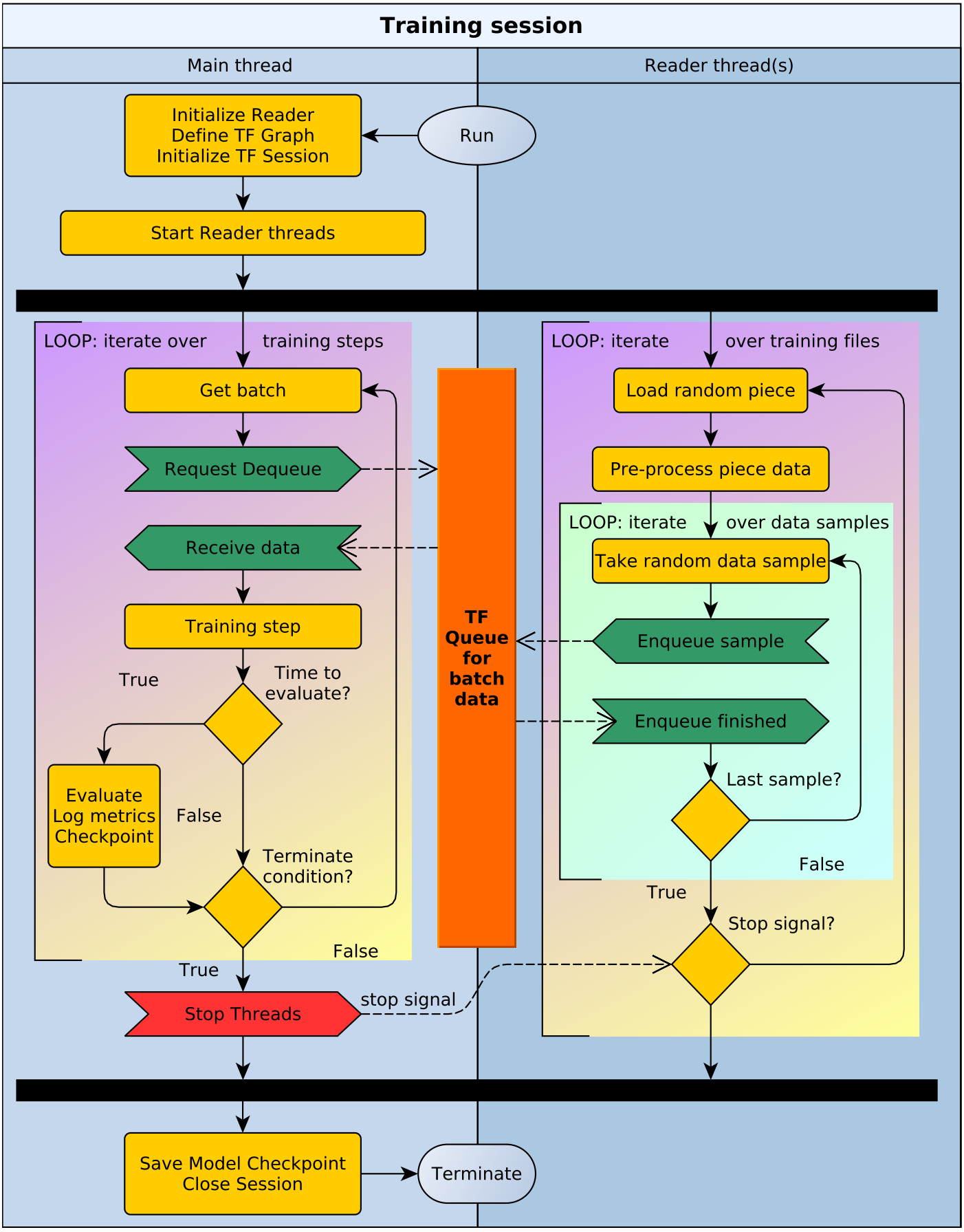
Following tuple $(waveform\ W, table\ T, indicces\ I)$ represents thus encodes the tune data and labels in much memory effective way. Single thread is dedicated to loading tunes from disk, performing this encoding and storing results in first *Tune Data Queue*. Next thread (possibly several threads), operates on tune data retrieved as tuple from this Queue. This thread performs backward up-sampling of indices vector and processes piece data and reference by $(sample\_size)$ fragments, reconstructing full size piano roll only one fragment at a time. This results in pairs $(waveform\ W, pianoroll\ P)$ which are further *enqueued* into *Batch Data Queue* for the training thread. Hihg-level overview of this processing pipeline is depicted on Figure 42.

We use simple `tf.FIFOQueue`[1] for *Tune Data Queue* implementation, since tunes are already being selected randomly by the reader thread. Although for implementation of *Batch Data Queue* we use `tf.PaddingFIFOQueue`[2], since cutting audio pieces into fixed size sequences almost always leaves last sequence of arbitrary size. This is where padding comes in, by *dequeue*-ing multiple samples from this kind of Queue, all sample sequences are automatically zero-padded up to the maximum sequence length within given batch.

A.3.2 Spectral Data Reading

In similar fashion, the `spec_reader.py` implements batching of data from corpus of WAV and MIDI files. Although only single Queue is used this time, since no intermediate encoding is appropriate here. Workflow goes as follows.

[1] `https://www.tensorflow.org/api_docs/python/tf/FIFOQueue`.
[2] `https://www.tensorflow.org/api_docs/python/tf/PaddingFIFOQueue`.



Obr. 43: *Diagram illustrating fundamentals of model training workflow.*

1. Reader thread randomly loads tune from data corpus.
2. Reader thread performs CQT transformation and retrieves piano roll.
3. Reader thread iteratively *enqueues* training samples randomly chosen from piece data until *tune sample rate* ratio given by `tune_sr` argument is satisfied.

For additional randomness in data sampling, `tf.RandomShuffleQueue`[1] is used, with emptiness constraint *min_after_dequeue* is set to 0.5, which specifies minimum number of elements that will remain in the Queue after *dequeue* operation, ensuring a minimum level of mixing elements.

[1] `https://www.tensorflow.org/api_docs/python/tf/RandomShuffleQueue`.

B Usage and Maintenance Guide

B.1 Install Guide

This install guide is provided for users of 64-bit Linux operating system and was tested on Ubuntu 16.04 LTS[1] distribution. In case you are running different operating system, please refer to official instructions online[2].

All installation commands need to be issued with root-level privileges, which is demonstrated by `sudo` prefix in provided examples, but may need different approach under various operating systems.

B.1.1 Install CUDA Toolkit

If your machine has CUDA compatible GPU, in order to utilize its computing capabilities, installation of NVIDIA's CUDA Toolkit, and additionally cuDNN library is required.

1. On site `https://developer.nvidia.com/cuda-downloads` select Linux as your target platform, architecture x86_64, distribution, version of your operating system and finally installer type "runfile (local).
2. Install CUDA Toolkit from within directory where you downloaded the installer using command `sudo sh cuda_8.0.44_linux.run` (edit to the file name you downloaded) and follow the command-line instructions.
3. Install the CUDA Toolkit into default directory.

   If it is not `usr/local/cuda/`, make sure there is symbolic link at this path pointing to the installation directory.

B.1.2 Install cuDNN library

Note, that registration to NVIDIA Developers site will be necessary to access the installation of cuDNN library.

1. Download cuDNN library from NVIDIA[3].

[1] `http://releases.ubuntu.com/16.04/`
[2] `https://www.tensorflow.org/get_started/os_setup`
[3] `https://developer.nvidia.com/cudnn`

2. Assuming that CUDA Toolkit is installed in `usr/local/cuda/`, uncompress and copy files from downloaded cuDNN library into this directory.
3. Run the following commands (edited to reflect the cuDNN version you downloaded).

```
tar xvzf cudnn-8.0-linux-x64-v5.1-ga.tgz
sudo cp -P cuda/include/cudnn.h /usr/local/cuda/include
sudo cp -P cuda/lib64/libcudnn* /usr/local/cuda/lib64
sudo chmod a+r /usr/local/cuda/include/cudnn.h
/usr/local/cuda/lib64/libcudnn*
```

B.1.3 Install pip and Jupyter Notebook

If you wish to run on python3, instead of `pip` use `pip3` in each of following commands.

1. Install `pip` - the package management system for python software - by running following command.

```
sudo apt-get install python-pip python-dev
```

2. Install Jupyter Notebook - interactive development environment for python.

```
sudo pip install -upgrade pip
sudo pip install jupyter
```

B.1.4 Install TensorFlow

1. Install Tensorflow either with GPU support, or without, depending on your hardware and preferences.

```
sudo pip install tensorflow-gpu
sudo pip install tensorflow
```

2. In case this command reports errors, please to this section on official page[1] for specific installation instructions.
3. Trigger TensorFlow installation.

```
sudo pip install -upgrade $TF_BINARY_URL
```

[1] `https://www.tensorflow.org/get_started/os_setup#pip_installation`

B.1.5 Install Auxiliary Python Libraries

Choose some local path, e.g. `/usr/local/lib`, to store installations of required python libraries.

B.1.5.1 Librosa

- To install `librosa`, run following command.

```
sudo pip install librosa
```

B.1.5.2 Pretty MIDI

- To install `pretty_midi` library, fork the project repository[1] into some local directory.
- In command line, within this working directory, run following command.

```
sudo python setup.py install
```

B.1.5.3 FluidSynth

- To install `FluidSynth` library for MIDI sound synthesis in python, first download project sources[2].
- Extract them into some local directory using following command.

```
tar -zxvf pyFluidSynth-1.2.4.tar.gz
```

- In command line, within this working directory, run following command.

```
sudo python setup.py install
```

B.1.5.4 MIR Eval

- To install `mir_eval` library for evaluation routines, fork project repository[3] into some local directory.
- In command line, in this working directory, run following command.

```
sudo python setup.py install
```

[1] `https://github.com/craffel/pretty-midi`
[2] `https://pypi.python.org/pypi/pyFluidSynth`
[3] `https://github.com/craffel/mir_eval`

B.2 User Guide

In this user guide, we describe two fundamental use cases with usage examples: training and usage for transcription / testing.

Additionally, in order to browse the Jupyter Notebooks for demonstration of usage or development and prototyping, follow these steps:

1. Start terminal, set your working directory to the root directory of this project (`source/`) as retrieved from attached medium.
2. Run following command.

   ```
   jupyter notebook
   ```

3. Open your web browser, and visit address `http://localhost:8888/`

You should now be able to see root directory of the project in web interface of `jupyter notebook`.

B.2.1 Configuration Files

To be able to set up the training or evaluation session, it is necessary to properly setup the program parameters. These are mostly inserted as CLI program arguments, or through a `json` file. Since CLI program arguments are properly documented and their list with meaning and usage is accessible through execution of given CLI program with argument -h according to conventions, we only explicitly explain formatting of `json` configuration files.

B.2.1.1 Note Generator

This configuration file, located at `/models/wn4t/note_generator_params.json`, is used to specify how notes are generated by `WaveReader` during training of WN4T and contains following parameters.

`poly` - integer (1, 2, ...) specifying polyphony level to generate examples at.

`combined` - boolean (`true`/`false`) value, whether to generate random polyphony from 1 to specified degree, or only polyphony of specified degree.

`maxlen` - integer (1, 2, ...) specifying upper limit number of seconds to generate for single piece.

B.2.1.2 WN4T Params

This configuration file, located at `/models/wn4t/default_params.json`, defines model parameters of WN4T for given training/testing session. These parameters must stay the same between training and testing of same model checkpoints.

`initial_filter_width` - integer value, size of the filter of *initial non-dilated* causal convolution used as first layer of the network.

`filter_width` - integer value, size of single filter of dilated causal convolution used across the network.

`dilations` - array of integers, specifies dilation factors of successive dilation layers, together with number of dilation layers implicitly by size of this array.

The remaining parameters are self-explanatory.

B.2.1.3 Spec Models Params

This configuration file, located at `/models/*_spec/default_params.json`, defines model parameters of *-Spec model for given training/testing session. These parameters must also stay the same between training and testing of same model checkpoints.

`temporal_resolution_hz` - integer value, specifying resolution required from piano roll and spectrograms for training, in Hz, which is number of time frames per second.

`base_audio_sample_rate` - integer value, base for audio sample rate, must be power of 2.

`bottom_frequency_note` - integer value, the lowest frequency examined by CQT spectral analysis.

`number_of_examined_notes` - integer value, number of notes upward adjacent from the bottom one, whose fundamental frequencies will be examined by CQT spectral analysis.

`bins_per_note` - integer value, number of spectral bins examined per fundamental frequency of each note, specifies spectral resolution of resulting spectrogram.

`outputs_note_range` - array of 2 integer values, first represents the bottom and second top musical note range interval to be examined in scope of network predictions, determines size of output layer.

`fc_sizes` - array of integer values, specifying sizes of fully connected layers within MLP part of given model, size of array implicitly defines number of layers.

`fc_activations` - array of strings, specifying activation functions of fully connected layers within MLP part of given model, cardinality of this set must match with fc_sizes.

`rnn_cell_type` - string, specifies type of RNN cell, possible values are listed in `rnn_cell_factory` defined in file `/models/birnn_spec/net/ops.py` along with possible values for `*_activations`.

`rnn_cell_size` - integer, specifies number of units in RNN cell.

`rnn_cell_activation` - string, specifies activation function for RNN cell(s).

`rnn_number_of_cells` - integer, specifies number of RNN stacked together.

The remaining, non-mentioned parameters are self-explanatory.

B.2.2 Training a Model with TensorBoard Monitoring

In order to start a neural network model training, make sure you have prepared training and validation data in discrete directories. Initially, for both sets, the same set containing single test file is used, namely path `/data/sample/mono`.

Also, make sure you forked this project folder structure on your own writable medium with sufficient disk space (at least 1GB of free space for initial TensorBoard logs).

1. Open terminal.
2. Set your working directory to the module of model to be trained.
3. Type `python train.py -h` in order to see the list of arguments with their descriptions and default values.
4. Type the command again with arguments specifying your data, log directory and other session parameters.

   Optionally add redirection of command output to a file in the log directory (e.g. `train.log`).

Example training execution command might look like following:

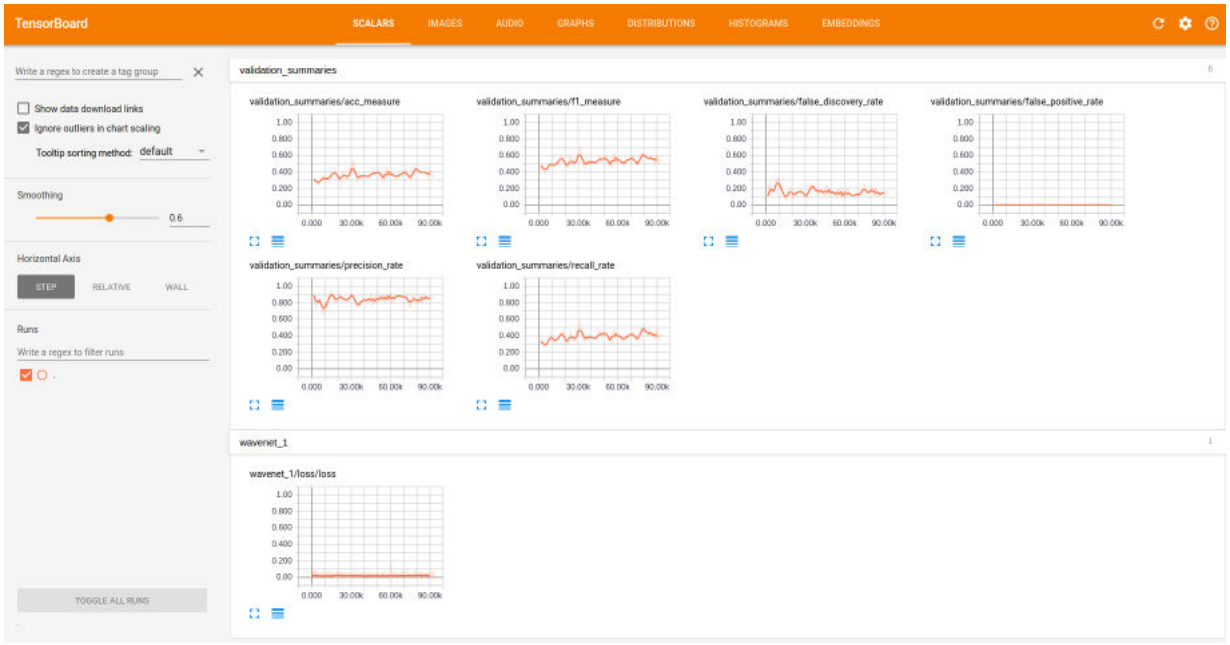
```
python train.py --data_dir_train ../../data/LABROSA/train --data_dir_valid ../../data/LABROSA/valid »
   » --histograms True --num_steps 1001 --evaluate_every 100 --logdir ../../logs/demo &>> ../../logs/»
   » demo/train.log
```

Optionally, in order to monitor and track the progress of training session in terms of logged metrics and intermediate evaluation on validation data, follow these steps for TensorBoard usage:

1. Open command line and go to project rooot directory.
2. Navigate to the logging directory specified for training session you want to monitor.
3. Launch TensorBoard by following command.

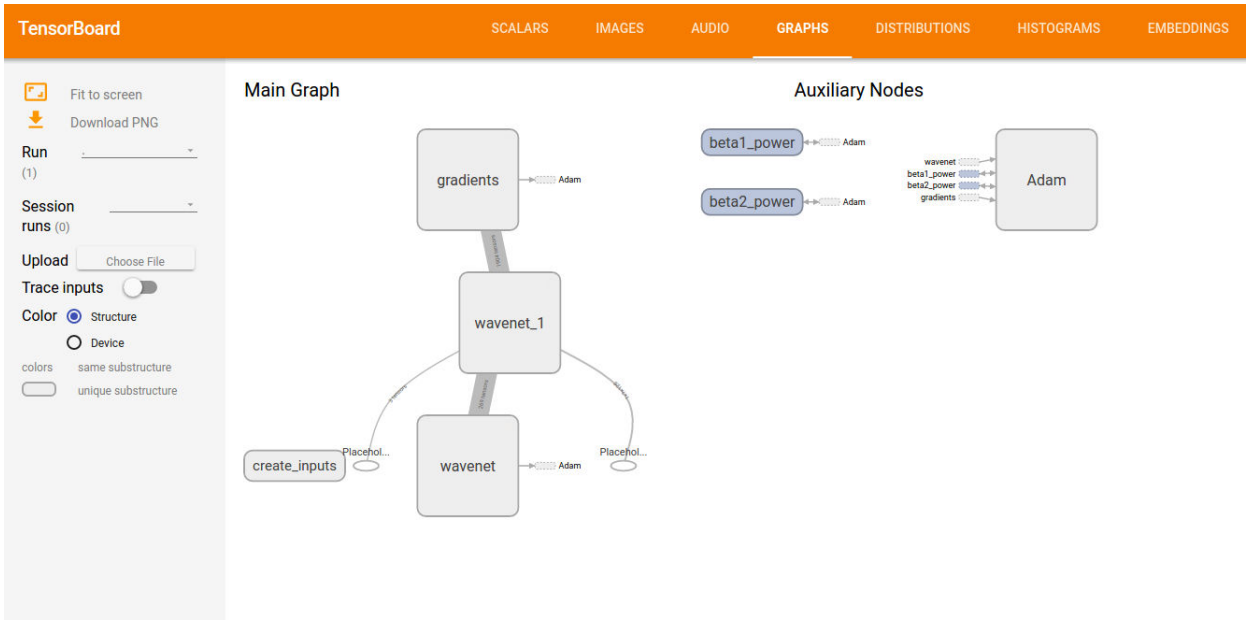
   ```
   tensorboard –logdir=.
   ```

4. Navigate to the URL from the console output (possibly `http://0.0.0.0:6006`).
5. In `SCALARS` dashboard, you will be able to see logged scalar metrics, such as minibatch loss, F1, and accuracy.



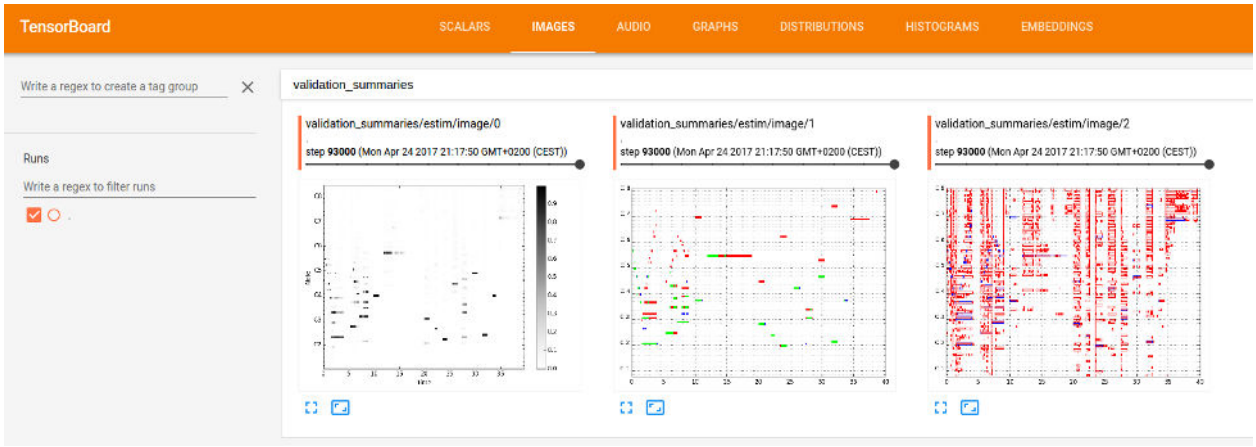
Obr. 44: *Screen shot of SCALARS dashboard.*

6. In `GRAPHS` dashboard, browse the computational graph of neural network model you are training in current session.



Obr. 45: *Illustrative screenshot of GRAPHS dashboard.*

7. During the training, in dashboards `IMAGES` and `AUDIO` you shall find audio and visual representations of validation data estimations.



Obr. 46: *Illustrative screenshot of IMGAES dashboard.*

B.2.3 Using Trained Model for Transcription and Evaluation

In order to test performance of neural network model checkpoint, prepare test data into discrete directory.

1. Open terminal.
2. Set your working directory to the module of model to be trained.
3. Type `python train.py -h` in order to see the list of arguments with their descriptions and default values.
4. Type the command again with arguments specifying your test data directory, log directory where checkpoint of trained model is stored and specify, if audio and image media should be generated, since by default, they will not.

Since by default, best performing models are saved into checkpoint files in subfolder `$LOGDIR/best/`, this is probably the directory where testing script should be instructed to look for model checkpoint to use for testing. Example test execution command might look like following:

```
python test.py --media True --data_dir ../../data/LABROSA/test --logdir ../../logs/demo/best
```

Test results should now be saved in file `$LOGDIR/best/metrics.json` while transcription results in terms of audio and visualizations should be in the same directory.

C Contents of Attached Electronic Media

The electronic media attached to this document has following structure and contents:

/doc

- master's thesis document together with annotations in Slovak and English language

/doc/bibtex

- reference files in BibTeX format

/doc/latex

- documentation files in LaTeX format

/doc/latex/tables/

- source tables of own graphs printed in thesis

/doc/latex/figures/

- pictures and visualizations printed in thesis

/doc/resources

- available resources used

/source

- root directory of project implementation

/source/data

- data sample for used in experiments described in this work

/source/logs

- logs from training sessions and checkpoints of some pretrained models

/source/sf2

- soundfonts used for sound synthesis and cited within this work

/tests

- complex outputs from tests

readme.txt

- description of media content in Slovak and English language

D Project Schedule

D.1 Summer 2016

- Deeper analysis and upcoming selection of appropriate preprocessing methods.
- Dataset selection and preprocessing for easy further access during experiments. Selection of benchmark problem and its benchmark data.
- Deeper analysis of existing models and their applications.
- Hypothesis preparation and design of models for upcoming experiments.
- Experiments with different architectures. Combining modules and testing different models on benchmark data.

D.2 Autumn 2016 (DP2)

- Evaluation of experiments and selection of promising setups.
- Writing design part of thesis report, revision of analysis.
- Implementation and extension of proposed method(s).
- Experiments with implemented method(s) on different data and different problems. Tuning the model.

Most of tasks listed in project schedule for summer and autumn of year 2016 were successfully fulfilled. Some of them only partially, and very few of them postponed, since they were not relevant yet, as we originally expected.

Anyway, the actual time scale of their fulfillment was slightly shifted against the schedule. Mostly, implementation tasks were performed during the summer, while documentation and analysis tasks were shifted to autumn.

Nevertheless, the time required for implementation tasks even during autumn was much larger than initially estimated. This is also why time reserve was and continues to be incorporated within project schedule.

Project schedule for upcoming semester has been updated according to latest progress and results.

D.3 Winter 2016/2017

- Implementation of model for transcription from raw audio.

    - specifically work plans outlined in Section 6.2.

- Large model-specific dataset preparation.
- Large-scale evaluation of proposed method(s).
- Preparation of research article for IIT.SRC 2017.
- Exploration of feature extraction methods and their influence to performance.

D.4 Spring 2017 (DP3)

- Experimental search for optimal setup of promising architectures.
- Final set of experiments and their evaluation.
- Thesis report finalization, including Slovak language extracted version.
- Time reserve for unexpected complications during previous work.

Since the implementation of WN4T model has been more complex task than expected due to significant memory issues, this task has been ongoing for whole winter and significant part of spring. However remaining points did not struggle too much because of this, since we have been able to work on then as well in parallel.

Although we did not get to exploration of feature extraction methods as was originally planned, we focused on experimentation with large data sets and large networks. This, however cause us way too large time delay, which forced us to finish the documentation of this work in an uncomfortably small time span.

E Paper Accepted to IIT.SRC 2017

Modelling Music Structure using Artificial Neural Networks

Lukáš MARTÁK*

Slovak University of Technology in Bratislava
Faculty of Informatics and Information Technologies
Ilkovičova 2, 842 16 Bratislava, Slovakia
martak.lukas@gmail.com

Abstract. As deep learning approaches arise thanks to availability of large datasets and high computing power, they show increasing competence at solving various tasks of growing complexity. Automatic music transcription is one such problem, which has been approached by computer scientists in music information retrieval for decades, remaining unsolved. Recent advances introduced deep architectures with significant audio modelling capacity. Since transcription of complex polyphony requires distinct cognitive capabilities, we believe, that deep learning could successfully tackle this problem. On top of spectral analysis, we propose neural network for frame-level classification, evaluate on standard dataset and conclude competitive and promising results.

1 Introduction

With the advent of new technologies, we have seen some dramatic transformations in music industry together with radical growth of music content. These transformations have influenced our means for musical content production, storage, distribution and consumption. As the richness of music content grows, it is crucial to have new methods to describe this content.

Music Information Retrieval (MIR) is an interdisciplinary science of retrieving various information from music. To list some interesting problems addressed by research in the field, there is Similarity Search, Query by Singing, Key Detection, Chord Estimation, Beat Tracking, Tempo Estimation and notably Multiple Fundamental Frequency Estimation.

All these tasks are motivated by a demand from either academia or industry, to provide software means for music analysis, production, distribution, organization, storage or reproduction.

Manually performed music transcription, also called *musical dictation* in music pedagogy, is a skill of identification music elements solely by hearing, which even talented musicians need to develop by practice (ear training).

In this paper, we address the problem of Automatic Music Transcription (AMT). It is considered one of the Holy Grails in the field, since a reliable transcription yields us symbolic representation of music content, which contains information substantial to many other MIR tasks. It may be also practically used for computational musicology or effective compression of musical data.

Many approaches to multi-pitch estimation have been examined so far. They break down by philosophy into these:

1. Frame-level - separate estimations per time frame.
2. Note-level - tracking notes from onset to offset.
3. Stream-level - tracking pitch streams by sources.

Those methods can be further categorized by domain of operation (time vs. frequency), or core algorithm (e.g. based on rules, signal processing domain knowledge, probabilistic or classification-based).

Since AMT is a complex task, many methods have been tuned to fit specific usage scenario or dataset characteristics. This variety in previous approaches also gave rise to different evaluation methodologies and metrics.

However, the common property of all present methods is the lack of accuracy, represented by several transcription errors per musical piece, which is still deep below human expert performance [1].

We examine a data-driven classification-based approach to frame-level multi-pitch estimation, based on neural networks. We further restrict our method to solo piano music and evaluate on corpus of classical piano tunes.

---

* Master study programme in field: Information Systems
Supervisor: Dr. Márius Šajgalík, Institute of Informatics, Information Systems and Software Engineering, Faculty of Informatics and Information Technologies STU in Bratislava

IIT.SRC 2017, Bratislava, April 27, 2017, pp. 1–8.

2 Related work

To reach the context of our research hypotheses, we focus here mostly on approaches to AMT and music audio modelling, based on machine learning.

Although there are some more recent relevant works, we refer to the project Automatic Piano Transcription by LabROSA laboratory[1] due to availability of used dataset. In this work [4], for each of 88 piano keys, an one-versus-all binary SVM classifier is trained on spectral features. Classification outputs are then processed a posteriori by a Hidden Markov Model for temporal smoothing. Results of this work have been used as a baseline in other similar work [3].

Recently, deep learning with Convolutional Neural Networks (CNNs) has been applied to spectral images in order to detect notes in polyphony [7]. After note onset times were detected, rectangular slices of spectrogram centered at those times were processed by CNN. Resulting 88 note probabilities were filtered by rule-based algorithm to obtain final predictions[2]. Though it has good results at detecting note onsets, this approach lacks ability to track note durations.

Additionally, new architecture of deep neural network was introduced by DeepMind, which demonstrated significant raw audio modelling capacity [8]. WaveNet[3] has been able to generate high quality speech and music audio fragments, one sample at a time. This suggests, that deep hierarchical structures in architecture of neural network could provide capacity for modelling musical structures in AMT and related subtasks, such as timbre recognition or dynamics estimation.

3 Frame-level spectral classification

Our method relies purely on a neural network multiclass classifier. It is trained on spectral amplitude data obtained from dataset of labeled music. Classifier outputs are truncated and used for frame-level evaluation.

1. Preprocess labeled music data.
   - Generate spectrograms from audio.
   - Generate piano rolls from notation.
   - Divide set into train, validation and test subsets with even distribution of data samples.
2. Train the model on training data.
   - Tune hyperparameters on validation data.
3. Evaluate on testing data.
   - Calculate relevant metrics.
   - Examine sample results empirically.

3.1 Preprocessing

Preprocessing is performed element-wise, where single data element is a pair of audio and notation file, describing single musical piece, e.g. tune or chord.

To enable time-frequency transformation, audio is initially re-sampled using method described in [6].

Spectrogram is calculated using Constant-Q Transform (CQT), which has constant ratio of frequency to bandwidth, resulting in equally representative spectral bands across all notes being analyzed. Since human perception of pitch operates in spectrum on a logarithmical scale, this gives CQT advantage over standard STFT, when it comes to musical data. To calculate CQT spectrum, we use recursive subsampling method for efficiency, as described in [5].

Piano roll is constructed from reference notation, as a matrix of numbers denoting absence or presence of notes (rows) in time frames (columns). Values from $\langle 0; 127 \rangle$ denote velocity of played notes.

We further normalize spectral magnitude values into $\langle -0.5; 0.5 \rangle$ to help neural net with convergence. For now, we also truncate piano roll values to $\{0, 1\}$, in order to discard the information about music dynamics and reduce the task to simple note detection.

Finally, data are divided into training, development and testing subsets roughly by ratios $8 : 1 : 1$.

3.2 Building the network architecture

In order to iteratively build optimal network architecture, we initiated our efforts with simple Multi-Layer Perceptron (MLP) model and feed it with spectral coefficients on the input, gaining discrete probabilities of presence per note on the output. Therefore, we denote this model as *MLP-Spec*.

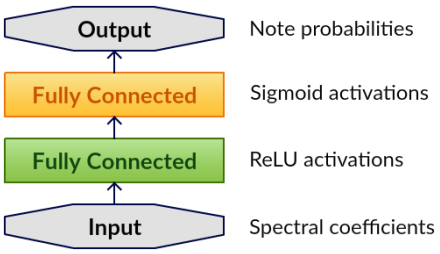


Figure 1. Architecture of MLP-Spec.

Since number, size and activation functions of hidden layers are subject to experimentation, we start with model depicted on Figure 1.

[1] http://labrosa.ee.columbia.edu/projects/piano/
[2] https://www.lunaverus.com/cnn
[3] https://deepmind.com/blog/wavenet-generative-model-raw-audio/

Rectified Linear Units (ReLU) on hidden layer work as computationally efficient non-linearity, while Sigmoid activations on output layer yield probability values from $\langle 0; 1 \rangle$ for presence of all notes, which makes the model a multi-class classifier. We use Adam [2] for model optimization by default.

Based on initial results, we introduced slight modification to *MLP-Spec*. In order to provide the model some time context, we added Gated Recurrent Unit (GRU) between the input and first hidden layer. We denote this model as *RNN-Spec* and feed it with a sequence of consecutive spectral descriptors, finished by the time frame being estimated.

Though length of the sequence is not restricted by RNN, since we do not aim to determine desired length of context dynamically, we experiment with constants.

During the training, we regularly perform inference on validation data, and checkpoint model parameters after each improvement, based on key metric. We also tried various combinations of model hyperparameters between training sessions, in order to maximize performance on validation data. We examined their dependencies based on convergence curves and top validation performance.

3.3 Postprocessing

If we discard velocity values in labels, we also reduce estimated probabilities to binary values according to specified threshold q, which is a *parameter of evaluation*, given

$$R_{n,t} = \begin{cases} 1, & \text{if } P_{n,t} \geq q \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

where note n and time frame t are indices to array P for estimated probabilities and array R for resulting piano roll.

4 Evaluation

4.1 Metrics

As a key metric for performance evaluation we track F_1-Score, which is a harmonic mean of precision $prec$ and recall rec, given by (2).

$$F_1 = 2 \cdot \frac{1}{\frac{1}{prec} + \frac{1}{rec}} = 2 \cdot \frac{prec \cdot rec}{prec + rec} \tag{2}$$

To enable comparison to reference approach, we also measure frame-level accuracy (3)

$$Acc = \frac{TP}{(FP + FN + TP)} \tag{3}$$

where TP, FP and FN denote total counts of true positives, false positives and false negatives.

Evaluation metrics are computed from estimations and labels. To examine confusion empirically, we further create visualizations of estimations or use piano roll to construct playable MIDI file.

4.2 Data

We first carefully selected preprocessing parameters. Time resolution of `100` frames per second turned out to be sufficient, also conforming to reference approach [4]. Bottom frequency was set to `27.5 Hz`, hence fundamental frequency of lowest piano note `A0`. Spectral resolution was initialized to `3` frequency bands per note, while `300` consecutive bands up from the lowest one were analyzed in order to capture the timbral artifacts of the highest piano notes.

LabROSA Most of our experiments with architectures were conducted on set of labeled recordings released by LabROSA [4] containing `28` classical piano pieces performed by a professional concert pianist.

From each tune, first `36` seconds were divided into `30` training, `3` validation and `3` testing seconds of data. With resolution of `100` frames per second this amounts to `87000` training samples and `8700` validation and testing samples.

4.3 Results

At all of our experiments, we stick to threshold value $q = 0.5$ and CQT calculation parameters `sample rate = 25600 Hz` and `hop length = 256` in order to preserve time resolution of `100 Hz`.

4.3.1 MLP-Spec: model depth dependency

With an experimental setup having `200` units wide hidden layers and `300` frames large batch size, we examined the influence of model depth on validation performance.

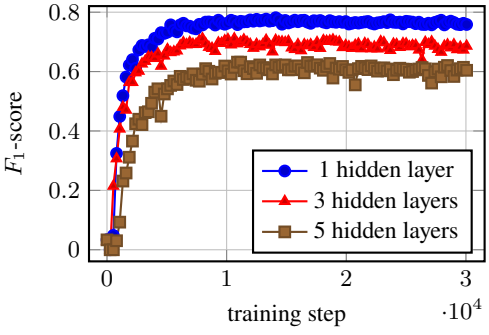


Figure 2. Validation performance of MLP-Spec at different depths.

In this case, results showed trend of decreasing validation performance with increasing model depth. Results on test data also confirmed this dependency.

In similar fashion, we examined the influence of context sequence length to *RNN-Spec* model performance. Results showed, that context did not help to improve performance. In fact, it actually dropped with growing context.

4.3.2 Comparing models performance

With the best performing variants of both examined architectures having single fully connected hidden layer with `1000` neurons, we trained and validated both on same data and compare their convergence curves in Figure 3.

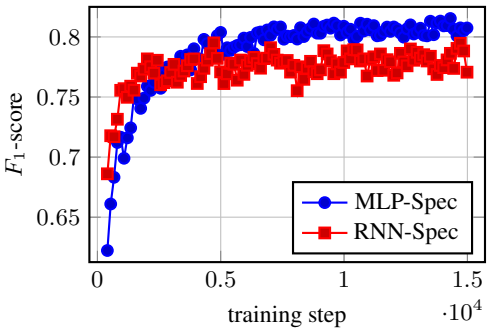


Figure 3. Validation performance comparison.

It is clear, that *RNN-Spec* converged into less optimal solution than *MLP-Spec*. We suppose, that *RNN-Spec* was able to recognize specific structures in spectral changes over time, thanks to provided context, and overfitted to their occurrences in test set. Our further work with larger corpus supports this theory.

Additionally we compare our results to the reference approach [4]. This comparison is only approximate, since the training and testing sets were differing in size between referenced evaluation and ours.

Table 1. Comparison to reference approach.

	MLP-Spec	**RNN-Spec**	**SVM**
Acc	64.3%	56.2%	56.5%

4.4 Further observations

From additional experiments with different datasets, we also observed the general tendency of neural network models to overfit to timbral characteristics of training data.

For example, models trained on synthesized audio achieved test performance with F_1-Score of ≈ 0.7 on synthesized audio data, while on data of recorded audio it was contrasting ≈ 0.3.

Another set of experiments on dataset of chords played in different inversions and transpositions showed the strong influence of dataset structure. Test performance of our models reached F_1-Score of ≈ 0.92 here, in contrast to best results on LabROSA set, which was ≈ 0.75.

5 Discussion

We have shown that even the simplest neural network architectures can be tuned to achieve competitive results on AMT tasks. It is therefore viable to further seek improvements in neural network architectures.

However, our current models seem to have issues identifying inter-frame dependencies, such as note durations. Therefore, one of the upcoming challenges will be finding effective representations and structures for time context modelling.

Some future directions include new feeding of spectrogram fragments with analyzed frame at the center of the sequence. Such fragments could be processed using combination of convolutional layers and recurrent layers.

Additionally, learning from raw audio could give interesting results with architectures employing multiple 1-dimensional convolutions and residual or skip connections.

Acknowledgement: This work was partially supported by the Scientific Grant Agency of Slovak Republic, grant No. VG 1/0646/15.

References

[1] Benetos, E., Dixon, S., Giannoulis, D., Kirchhoff, H., Klapuri, A.: Automatic music transcription: challenges and future directions. *Journal of Intelligent Information Systems*, 2013, vol. 41, no. 3, pp. 407–434.

[2] Kingma, D.P., Ba, J.L.: Adam : A method for stochastic optimization. *ICLR*, 2015, pp. 1–15.

[3] Nam, J., Ngiam, J., Lee, H., Slaney, M.: A Classification-Based Polyphonic Piano Transcription Approach Using Learned Feature Representations. *Ismir*, 2011, no. Ismir, pp. 175–180.

[4] Poliner, G.E., Labrosa, D.P.W.E.: A Discriminative Model for Polyphonic Piano Transcription, 2006, pp. 1–16.

[5] Schörkhuber, C., Klapuri, A.: Constant-Q transform toolbox for music processing. *7th Sound and Music Computing Conference*, 2010, no. JANUARY, pp. 3–64.

[6] Smith, J.O.: Digital Audio Resampling Home Page. *Center for Computer Research in Music and Acoustics (CCRMA), Stanford University*, 2002, pp. 1–20.

[7] Troxel, D.: Music transcription with a convolutional neural network. MIREX, 2016.

[8] van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., Kavukcuoglu, K.: WaveNet: A Generative Model for Raw Audio, 2016, pp. 1–15.

F Resume in Slovak Language

F.1 Úvod

Cieľom tejto práce je navrhnúť, zrealizovať a otestovať metódu automatickej transkripcie hudby s využitím metód hlbokého učenia. Súčasne je prioritou práce inšpirovať sa existujúcimi architektúrami neurónových sietí a aplikovať identifikované vzory pri vlastnom návrhu architektúry.

F.1.1 Motivácia

Music Information Retrieval (MIR) je medzinárodná vedecká disciplína zaoberajúca sa získavaním informácií z hudby. Je to malá výskumná oblasť, avšak s veľkým potenciálom pre aplikáciu vo svete. Zmienime iba niektoré zo zaujímavých problémov adresovaných výskumom v tejto oblasti.

1. Hľadanie podobnosti skladieb.
2. Dotazovanie spievaním či *hmm*-kaním melódie.
3. Automatické odhadovanie akordov znejúcich v polyfónií.
4. Sledovanie rytmu.
5. Odhadovanie tempa.
6. Viacnásobné odhadovanie fundamentálnej frekvencie.
7. Automatická kategorizácia.
8. Automatická transkripcia.

Všetky tieto úlohy sú motivované dopytom buď z akadémie, alebo priemyslu, aby poskytli softvérové prostriedky pre analýzu, produkciu, distribúciu, organizáciu v databázach, či reprodukciu hudby.

Manuálne vykonávaná transkripcia hudby je proces identifikácie hudobného obsahu väčšinou čisto na základe sluchu, čo dokonca aj talentovaní hudobníci potrebujú rozvíjať *cvikom ucha*.

F.2 Analýza problémovej oblasti

V tejto práci adresujeme problém Automatickej Transkripcie Hudby (Automatic Music Transcription - AMT). Táto úloha je považovaná za Svätý Grál v oblasti, keďže dostatočne presná transkripcia poskytuje symbolickú reprezentáciu hudobného obsahu, ktorá už zahŕňa podstatnú časť informácie potrebnú pre riešenie ostatných príbuzných problémov. Taktiež by mohla byť prakticky využitá pri výskume počítačovej muzikológie, či pre efektívnu kompresiu záznamov hudobných nahrávok do formátu symbolickej notácie.

Viacero prístupov k problému odhadovania polyfónie už bolo vyskúšaných. Z hľadiska filozofie ich môžeme rozdeliť do nasledovných kategórií.

1. Na úrovni rámcov - odhadovanie polyfónie v každom časovom rámci zvlášť.
2. Na úrovni nôt - sledovanie nôt od nástupu po odznenie.
3. Na úrovni prúdov - sledovanie prúdov tónov podľa charakteristiky ich zdrojov.

Tieto metódy by sme mohli ďalej kategorizovať aj inak, napríklad podľa domény v ktorej operujú (časová / frekvenčná), či typu jadra použitého algoritmu (napríklad pravidlové systémy, algoritmy založené na expertných znalostiach z oblasti spracovania signálov, pravdepodobnostné modelovanie, či klasifikačné algoritmy).

Keďže AMT je komplexná úloha, veľa metód bolo ladených pre špecifický používateľský prípad, či špecifický charakter hudby. Táto rozmanitosť v predchádzajúcich prístupoch taktiež podnietila vznik rôznych metodológií vyhodnocovania a metrík pre sledovanie úspešnosti týchto metód.

Avšak spoločná vlastnosť všetkých súčasných metód je stále nedostatočná presnosť reprezentovaná štatistickou pravdepodobnosťou chyby meranou v desiatkach percent za predikciu, čo v praxi predstavuje niekoľko chýb transkripcie za čo i len minútu skladby. Toto je stále hlboko pod úrovňou výkonu ľudského experta - človeka s hudobným sluchom a praxou v transkripcií [17].

V tejto práci skúmame prístup budovaný predovšetkým na dátach, klasifikácií na úrovni rámcov a založený na neurónových sieťach. Obmedzujeme sa však v rozsahu tejto práce iba na úlohu transkripcie sólovej klavírnej hudby. Našu metódu budujeme a aj vyhodnocujeme na korpuse klasických klavírnych skladieb.

F.2.1 Existujúce riešenia

V analýze predchádzajúcej práce sa venujeme prevažne prístupom k AMT problému a modelovaniu zvuku hudby založeným na metódach strojového učenia.

Hoci už existujú novšie relevantné práce v tejto oblasti, zvolili sme si ako referenčný prístup projekt s názvom Automatic Piano Transcription od laboratória LabROSA[1], vzhľadom na charakter použitého prístupu a dostupnosť dát ktoré boli využité na učenie modelu. V tomto článku [67] je pre každý z 88 klavírnych tónov samostatný SVM klasifikátor trénovaný na spektrálnych koeficientoch. Výstupy klasifikácie sú ďalej spracované skrytým markovským modelom (HMM) pre časové vyhladenie výstupov. Výsledky tejto práce už boli taktiež využité ako podklad pre porovnanie inými podobnými prácami, ako napríklad práca autora menom Juhan Nam a kolektívu [63].

V tejto práci autori pristupujú k problému polyfonickej transkripcie hudby s použitím niekoľkých metód strojového učenia. Ich prístupom je najskôr aplikovať techniku PCA (Principal Component Analysis) bielenia a následne normalizáciu na spektrogram. V ďalšej fáze sa učením bez učiteľa trénuje dvojvrstvová neurónová sieť DBN (Deep Belief Network), čo sa už v minulosti ukázalo ako úspešné pri úlohách klasifikácie hudby [37, 55]. Tieto boli vrstvy boli trénované tzv. *chamtivým* spôsobom, vrstvou po vrstve. Aktivácie na skrytej vrstve DBN boli ďalej spracované sadou SVM klasifikátorov. Predikcie týchto klasifikátorov boli aktivované funkciou sigmoid na zadné (posterior) pravdepodobnosti, ktoré boli následne spracované dvoj-stavovým HMM pre časové vyhladenie.

Avšak nedávno bolo hlboké učenie konvolučnými neurónovými sieťami (CNNs) aplikované na obrázky spektier za účelom detekcie nôt v polyfónií [82]. Najskôr sa jednoduchým algoritmom pre hľadania vrcholov v spektre našli časy nástupov tónov, následne obdĺžnikové plátky spektrogramu centrované voči časom zdetekovaných tónov boli spracované CNN sieťou. Sieť klasifikovala 88 pravdepodobností znenia tónov, ktoré boli ešte následne filtrované algoritmom založeným na pravidlách získaných heuristikou. za účelom získania finálnych predikcií[2]. Hoci má tento prístup dobré výsledky v detekcií nástupov tónov, chýba mu schopnosť sledovať dĺžky trvania tónov.

Navyše taktiež nedávno DeepMind na scénu uviedol novú architektúru hlbokej neurónovej siete, ktorá demonštrovala znamenitú kapacitu pre modelovanie surového zvukového

[1] `http://labrosa.ee.columbia.edu/projects/piano/`.
[2] `https://www.lunaverus.com/cnn`

signálu [**?**]. WaveNet[1] dokázal vygenerovať vysoko kvalitné úryvky reči a hudby, a to vzorku po vzorke, čisto modelovaním podmienenej pravdepodobnostnej distribúcie nasledujúcej vzorky na základe existujúcej sekvencie vzoriek. To nám napovedá, že hlboké hierarchické štruktúry v architektúrach neurónových sietí by mohli poskytnúť dostatočnú kapacitu pre modelovanie hudobných štruktúr priamo zo zvukového signálu, za účelom AMT a podobných podproblémov, ako rozpoznávanie farby tónu či odhad dynamiky hraného tónu.

F.3 Navrhnuté metódy a opis riešenia

Vzhľadom na špecifickosť akustických a hudobných signálov uvažujeme v tejto práci viacero alternatívnych prístupov k problému polyfonickej transkripcie hudby z hľadiska metód neurónových sietí.

Tieto alternatívy sa líšia najmä v spôsobe, akým reprezentujú hudobný signál na vstupe neurónovej siete, čo taktiež určuje rozsah problému, ktorý musí daná sieť riešiť, čo následne ovplyvňuje rozhodnutia o návrhu architektúry danej siete.

F.3.1 Modelovanie hudby v spektrálnej doméne

Prvým krokom v takmer každom systéme spracovania zvuku za účelom analýzy hudobného obsahu je logicky a už tradične transformácia z časovej domény do frekvenčnej. My preto taktiež najskôr skúšame trénovať neurónové siete na deskriptoroch spektrálnych čŕt.

Celkový proces v krokoch potrebný pre aplikovanie tejto metódy je teda nasledovný.

1. Predspracovanie zvuku, generovanie spektrogramu.
2. Predspracovanie referenčnej notácie do značiek, generovanie *piano roll*ov.
3. Časová synchronizácia predspracovaných dát a značiek.
4. Rozdelenie dát na trénovaciu, validačnú and testovaciu množinu so snahou zachovať podľa možností najvyváženejšiu rôznorodosť hudobného obsahu naprieč týmito množinami.
5. Trénovanie modelu na trénovacej množine s periodickým vyhodnocovaním na validačných dátach. Prispôsobovanie hyperparametrov naprieč trénovaniami za účelom maximalizácie validačnej presnosti (teda presnosti na validačnej množine).

[1] `https://deepmind.com/blog/wavenet-generative-model-raw-audio/`

6. Testovanie modelu na testovacej množine. Vyhodnotenie presnosti transkripcie pomocou vypočítaných metrík vyhľadávania informácií a taktiež manuálnym preskúmaním výstupov.

F.3.1.1 Predspracovanie

Predspracovanie sa urobí po samostatných dátových jednotkách, ktoré reprezentujú samostatná jednotky hudobného obsahu, ako napríklad skladby, stupnice, či akordy, v závislosti od štruktúry datasetu.

Za účelom umožnenia transformácie do frekvenčnej domény je najskôr zvuk pre-vzorkovaný metódou popísanou v [81].

Spektrogram je vypočítaný s použitím CQT (Constant-Q Transform) algoritmu, ktorý má výhodu oproti klasickému STFT (Short-Time Fourier Transform) takú, že má konštantný pomer frekvencie voči frekvenčnému rozsahu, ktorý analyzuje. To má za následok rovnomerné rozmiestnenie reprezentatívnych spektrálnych pásiem naprieč frekvenciami všetkých skúmaných nôt. Keďže ľudské vnímanie výšky tónu operuje v spektre na logaritmickej škále, toto dáva CQT algoritmu výhodu oproti STFT ak sa jedná o analýzu hudobných dát. Na výpočet CQT spektra používame metódu rekurzívneho pod-vzorkovania za účelom výpočtovej efektivity tak, ako je popísaná v [75].

Piano roll je skonštruovaný z referenčnej notácie ako matica čísel značiacich absencie alebo prezencie nôt (riadky) v časových fragmentoch (stĺpce). Hodnoty z intervalu $\langle 0; 127 \rangle$ značia dynamiku (velocity) znejúcich nôt. Táto sa však v čase nemení, udáva iba silu úderu na začiatku tónu a ostáva konštantná až po jeho koniec.

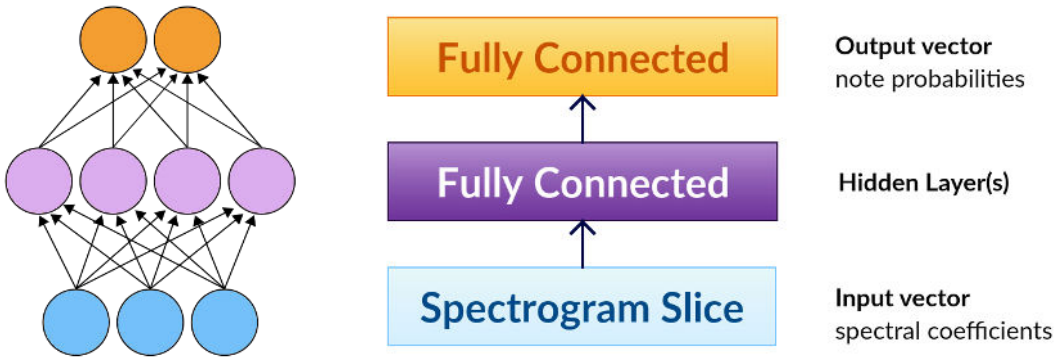
V ďalšom kroku normalizujeme hodnoty spektrálnych magnitúd do $\langle -0.5; 0.5 \rangle$ s cieľom dopomôť neurónovej sieti ku konvergencií. V súčasnej fáze projektu taktiež zarovnávame *piano roll* hodnoty na $\{0, 1\}$, za účelom zbavenia sa informácie o dynamike hudby a súčasného zredukovania úlohy o úlohu detekcie dynamiky na úlohu čistej detekcie prítomnosti nôt.

Na záver rozdelíme dáta do trénovacej, vývojovej a testovacej podmnožiny zhruba v pomere $8 : 1 : 1$.

F.3.1.2 Návrh architektúry a trénovanie

Za účelom postupného budovania optimálnej architektúry začíname s jednoduchým modelom viac-vrstvového perceptrónu (MLP) a trénujeme ho na spektrálnych koeficientoch, dostá-

vajúc diskrétne hodnoty pravdepodobností prítomnosti noty na výstupe. Preto tento model označujeme ako *MLP-Spec*.



(a) Viac-Vrstvový Perceptrón (b) Prvotné nastavenie MLP modelu

Obr. 47: *Náčrt MLP-Spec architektúry.*

Veľkosť vstupnej vrstvy závisí od spektrálneho rozlíšenia zvoleného v čase predspracovania dát formou kombinácie parametrov výpočtu CQT spektrogramu. Veľkosť výstupnej vrstvy je daná rozsahom nôt, ktoré si povieme, že chceme aby sieť predikovala. Počet a veľkosti skrytých vrstiev a ich aktivačné funkcie patria do množiny hyperparametrov modelu.

Prvotné nastavenie obsahuje jedinú skrytú vrstvu a výstupnú vrstvu. Aktivačná funkcia na skrytej vrstve je použitá ReLU (Rectified Linear Unit), za účelom výpočtovej efektivity a postačujúcej nelinearity modelu.

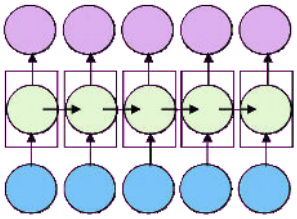
Na výstupe máme sigmoid aktiváciu, keďže chceme z našej siete mať viac-triedovú klasifikáciu, podobne ako majú aj relevantné existujúce prístupy, keďže táto funkcia aktivuje pred-aktivácie výstupnej vrstvy do sady hodnôt z intervalu $(0, 1)$ reprezentujúceho diskrétne pravdepodobnosti estimácií jednotlivých tónov.

V ďalšej iterácií návrhu architektúry na základe prvotných výsledkov pridávame rekurentnú vrstvu na začiatok ako vstupnú vrstvu siete, za účelom poskytnutia časového rámca ktorého polyfóniu predikujeme. Túto rekurentnú vrstvu kŕmime *sekvenciou* spektrálnych deskriptorov po sebe idúcich časových rámcov.

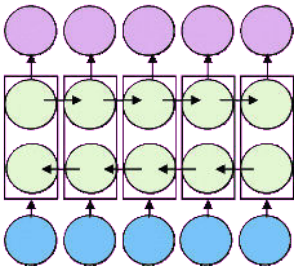
V prvej iterácií používame jednosmernú rekurentnú vrstvu na spracovanie sekvencie. V čase trénovania predikujeme všetky rámce zo sekvencie pre rýchlejšie učenie. V čase testovania berieme do úvahy vždy iba tie predikcie ktoré ukončujú danú sekvenciu ktorá bola na vstupe, za účelom zúžitkovania všetkej kontextuálnej informácie.

Navyše za účelom využitia časového kontextu z oboch strán odhadovaného rámca v čase, teda aj z minulých a aj z budúcich okolitých rámcov, skúmame možnosť využitia obojsmernej rekurentnej vrstvy. Takejto sieti potom dávame predikovať časový rámec uprostred poskytnutej sekvencie, namiesto toho koncového.

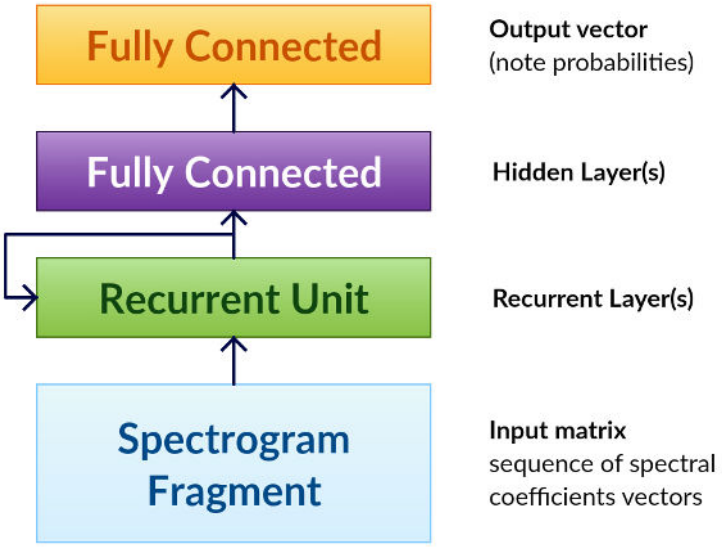
Na základe typu rekurentnej vrstvy použitej vrámci konkrétnej variácie modelu označujeme model s jednostrannou rekurentnou vrstvou ako *RNN-Spec* a model s obojstrannou ako *BiRNN-Spec*.



(a) Klasická RNN



(c) Obojsmerná RNN



(b) Základné nastavenie rekurentných modelov

Obr. 48: *Náčrt rekurentných architektúr založených na MLP.*

Hoci dĺžka sekvencie nie je obmedzená rekurentnou vrstvou, keďže v súčasnosti nemáme metódu určovania žiadanej dĺžky sekvencie inú ako experiment, vyberáme hodnotu konštantne pre každé trénovanie a pracujeme s ňou ako s hyperparametrom.

F.3.2 Postspracovanie

V zmysle úlohy detekcie nôt kde informácia o dynamike je nedostupná, posledným krokom spracovania je *prahovanie*, čo v jednoduchosti znamená zaokrúhľovanie odhadovaných pravdepodobností do logických hodnôt. Toto robíme na základe daného prahu q, ktorý je *parametrom vyhodnocovania*. Ak máme

$$R_{n,t} = \begin{Bmatrix} 1, & \text{if } P_{n,t} \geq q \\ 0, & \text{otherwise} \end{Bmatrix} \tag{9}$$

kde nota n a časový rámec t sú indexy do polí P pre odhadované pravdepodobnosti a R pre originálny *piano roll*.

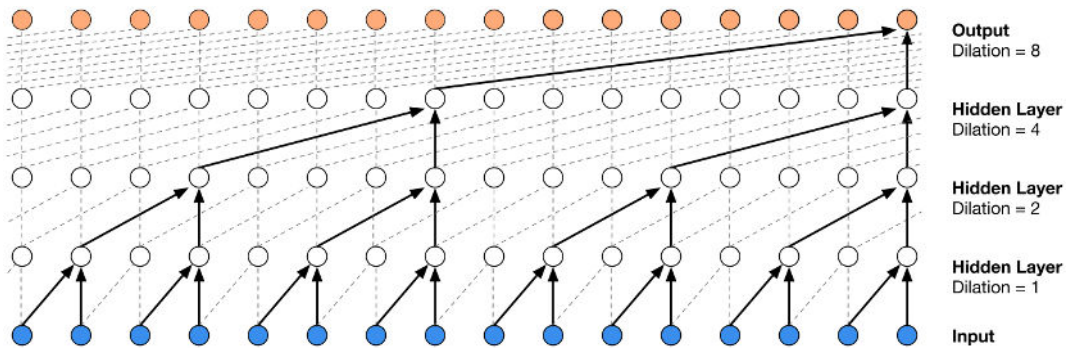
Z tejto reprezentácie výstupu už počítame štandardné metriky pre kvantifikované vyhodnotenie. Taktiež robíme vizualizácie výstupov a manuálnym skúmaním pozorujeme a vyhodnocujeme výkony našich metód.

F.3.3 Modelovanie hudby v časovej doméne

Keďže spektrálna analýza, aj keď je veľmi cenná a dôležitá, predsa len prichádza za cenu narušenia signálu, strácame nejaké množstvo informácie. Pri *windowingu* zvukových úsekov počas spektrálnej analýzy zahadzujeme informáciu o kontinuite signálu a zároveň predsa len vytvárame v spektre nejaké artefakty.

Preto má zmysel skúšať učiť neurónovú sieť priamo zo časovej domény, teda zo surovej reprezentácie vzorkovaného signálu. Začíname experimentovať s architektúrou WaveNet, ktorá už ukázala svoj potenciál pre spracovanie zvuku pri generatívnych úlohách [84].

Architektúra WaveNetu je poskladaná z vrstiev tzv. *dilatovaných kauzálnych konvolúcií* zobrazených nižšie.

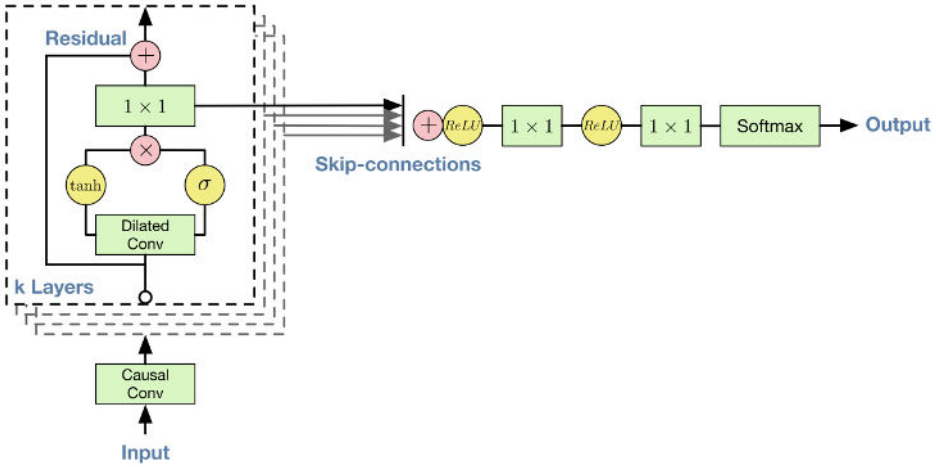


Obr. 49: *Zásobník* dilatovaných *kauzálnych konvolúcií; prekreslený z [84].*

Zvyšovaním dilatácie po sebe idúcich vrstiev o faktor 2-ky zvyšujeme zorné pole[1] siete exponenciálne. Ďalším prostriedkom rozširovania zorného poľa siete je aplikácia viacerých

[1]Zorné pole predstavuje počet vzoriek, ktoré je WaveNet schopný spracovať na vstupe naraz v jednom kroku odvodenia.

zásobníkov po sebe. Toto pomáha zvýšiť nie len zorné pole, ale aj kapacitu siete. Celá architektúra WaveNetu je načrtnutá na obrázku nižšie.



Obr. 50: *Prehľad reziduálneho bloku a celej WaveNet architektúry; prekreslený z [84].*

Na základe inšpirácie predošlým úspechom WaveNet architektúry v doméne modelovania zvuku, zakladáme našu verziu na open-source implementácií[1] a prispôsobujeme ju našej úlohe. Zámenou *softmax* aktivácie za *sigmoid* na výstupe získavame viac-triedový klasifikátor. Veľkosť finálnej vrstvy zužujeme na `128` MIDI tónov z predošlých `255` hodnôt kvantizovaného zvukového signálu.

Za účelom odlíšenia našej upravenej verzie WaveNetu od pôvodnej si ju aj pre potreby ďalšieho značenia a vyhodnocovania tejto práce označíme ako *WN4T* (WaveNetForTranscription).

F.4 Dosiahnuté výsledky

Za účelom priebežného vyhodnocovania sme počas trénovania priebežne vykonávali vyhodnotenie na validačnej množine dát a uložili si stav modelu pri každom zlepšení kľúčovej metriky na historické maximum v rámci daného trénovania. Týmto spôsobom sme boli schopní hľadať vhodné nastavenie hyperparametrov vrámci viacerých trénovaní za účelom maximalizovať presnosť na validačnej množine.

[1] `https://github.com/ibab/tensorflow-wavenet`

F.4.1 Vyhodnocované metriky

Kľúčová metrika ktorú meriame pre vyhodnotenie presnosti je F_1, čo je harmonický priemer presnosti $prec$ and úplnosti rec, danej nasledovne.

$$F_1 = 2 \cdot \frac{1}{\frac{1}{prec} + \frac{1}{rec}} = 2 \cdot \frac{prec \cdot rec}{prec + rec} \tag{10}$$

Pre umožnenie porovnania voči referenčným prístupom taktiež meriame rámcovú presnosť tak, ako bola navrhnutá Dixonom [28]

$$Acc = \frac{TP}{(FP + FN + TP)} \tag{11}$$

kde TP, FP a FN označujú celkové počty správne pozitívnych, nesprávne pozitívnych a nesprávne negatívnych odhadov.

Tieto metriky sú počítané z porovnania estimácií a značiek. Pre empirické skúmanie pomýlenia ďalej vizualizujeme estimácie či používame piano roll pre zostrojenie prehrateľného MIDI zápisu.

F.4.2 Použité dáta

V našich experimentoch sme použili na trénovanie, validáciu aj testovanie niekoľko variácií datasetu LabROSA zverejneného referenčným projektom, ktorý pozostáva z `29` klasických klavírnych skladieb hraných profesionálnym koncertným klaviristom a štúdiovo nahraných. Taktiež sme z týchto MIDI súborov syntetizovali audio s použitím vysoko kvalitných zvukových fontov formátu SF2[1].

Pre syntézu vlastnej verzie datasetu sme z každej skladby vzali prvých `40` sekúnd a rozdelili ich do `32` trénovacích, `4` validačných a `4` testovacích sekúnd dát. S časovým rozlíšením `100` rámcov za sekundu nám toto dáva `92800` trénovacích rámcov, `11600` validačných a `11600` testovacích rámcov spektrálnych magnitúd.

F.4.3 Výsledky experimentov

V rozsahu experimentov ktoré sa nám podarilo vykonať sme zostali pri konštantných hodnotách pre prah $q = 0.5$ a parametre CQT kalkulácie `sample rate = 25600 Hz` a `hop`

[1] `http://www.synthfont.com/sfspec24.pdf`

`length = 256` za účelom zachovania časového rozlíšenia o veľkosti `100 Hz`.

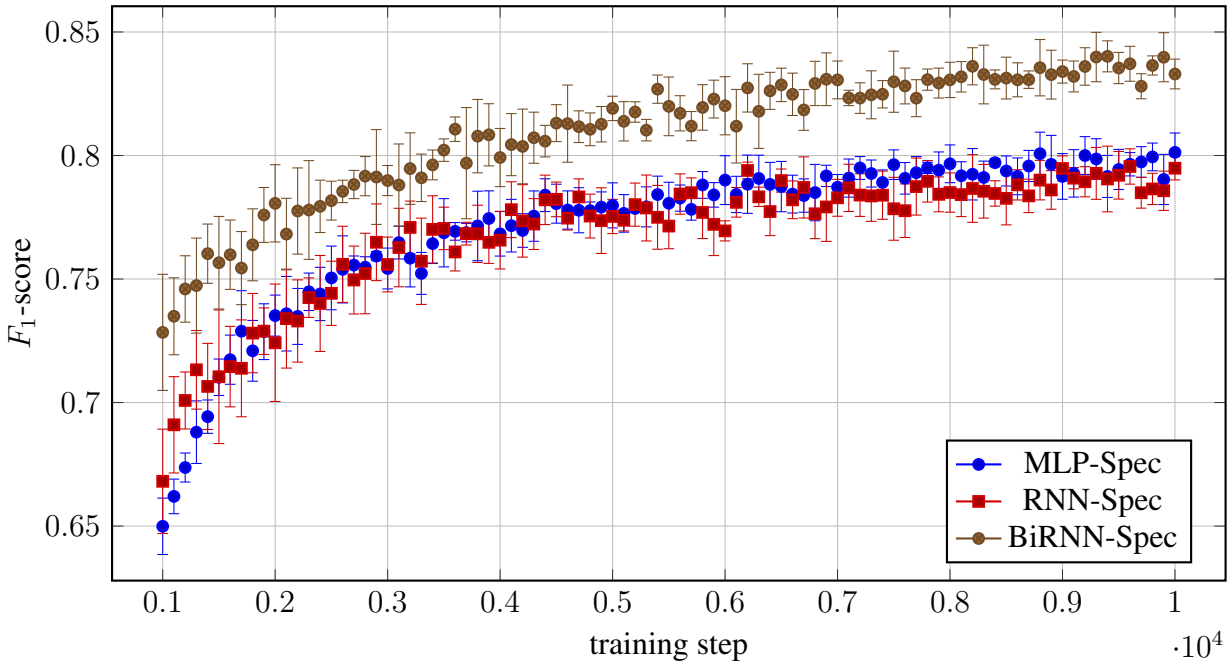
Po implementácií *BiRNN-Spec* modelu sme ich trénovali všetky tri na rovnakom korpuse syntetizovanom z LabROSA datasetu. Podstatné hyperparametre uvádzame v tabuľke nižšie[1].

Tabuľka 18: *Parametre* Spec *modelov trénovaných na syntetizovanom korpuse.*

	MLP-Spec	**RNN-Spec**	**BiRNN-Spec**
batch size	500	100	100
sequence length	-	20	21
recurrent layers	-	[1000]	[1000]
recurrent activations	-	[tanh]	[tanh]
fully connected layers	[1000]	[1000]	[1000]
fully connected activations	[relu]	[relu]	[relu]

V tomto experimente bol každý model trénovaný v 5 nezávislých behoch s náhodne inicializovanými váhami. Priebehy úspešností na validačnej množine sú vykreslené ako priemerné hodnoty z týchto pätíc, spolu s uvedením štandardnej odchýlky pre každú priemernú hodnotu.

[1] Každý element v hranatých zátvorkách popisuje samostatný vrstvu.



Obr. 51: *Validačný výkon na syntetizovanom korpuse.*

Prvotné výsledky naznačujú, že BiRNN-Spec model vyzerá ako naše doteraz najlepšie zlepšenie v návrhu architektúry. Avšak sú potrebné rozsiahlejšie testy.

Po natrénovaní modelu WN4T spôsobom postupného pridávania príkladov do trénovacej množiny od najjednoduchších monofónnych tónov cez dvojtóny až ku zložitejším polyfóniám sme otestovali aj tento model a porovnali ho s ostatnými.

WN4T model bol trénovaný na `16000` Hz zvuku. Zásobník dilatačných vrstiev nami použitej konfigurácie modelu je načrtnutý v tabuľke nižšie.

Tabuľka 19: *Zoznam dilatácií modelu* WN4T.

Dilatačné faktory po sebe idúcich vrstiev										
1	2	4	8	16	32	64	128	256	512	1024
1	2	4	8	16	32	64	128	256	512	1024
1	2	4	8	16	32	64	128	256	512	1024

Zásobník pozostáva z 33 vrstiev a zabezpečuje zorné pole veľkosti 6143 zvukových vzoriek. Veľkosť jednej trénovacej sekvencie je 10000 zvukových vzoriek.

F.4.3.1 Vzájomné porovnanie našich modelov

V tabuľke nižšie je porovnanie testovacieho výkonu našich modelov trénovaných na syntetizovanom zvuku z korpusu LabROSA.

Tabuľka 20: *Porovnanie našich modelov.*

	MLP-Spec	**RNN-Spec**	**BiRNN-Spec**	**WN4T***
precision	0.81	0.8	**0.84**	0.74
recall	0.74	0.75	**0.76**	0.37
F1	0.78	0.77	**0.8**	0.5
Acc	0.63	0.63	**0.66**	0.33

F.4.3.2 Porovnanie s referenčným prístupom

Na záver sme ešte skúsili porovnať naše výsledky voči prístupu [67] vďaka dostupnosti referenčného datasetu, ktorého čo najvernejšiu kópiu sme sa snažili pre toto porovnanie zrekonštruovať

Po ≈ 3 trénovacích epochách sme trénovanie ukončili, porovnanie výsledkov je načrtnuté v tabuľke nižšie.

Tabuľka 21: *Porovnanie s referenčnými prístupmi.*

Approach	**Recorded (10)**	**Synthesized (25)**	**Combined (35)**
Poliner and Ellis [67]	**56.5%**	**72.1%**	67.7%
Nam et al [63]	-	-	**72.5%**
Ryynanen and Klapuri [74]	41.2%	48.3%	46.3%
Marolt [57]	38.4%	40.0%	39.6%
MLP-Spec	56.2%	61.2%	59.2%
RNN-Spec	46.3%	56.1%	52.1%
BiRNN-Spec	54.5%	63.5%	59.9%

Kôli nedostatku času sme nestihli naše modely optimalizovať na validačnej množine dát, na rozdiel od konkurenčných prístupov, preto sú tieto výsledky iba dočasné porovnanie a pre overenie vhodnosti navrhnutého konceptu.

*Hoci je tento model zahrnutý v porovnaní, keďže sa veľkosťou aj parametrami líši od ostatných, bol trénovaný odlišne, tak ako je opísané vyššie.

W ďalšej práci bz sme mali optimalizovať hyperparametre ako veľkosti vrstiev, regularizačné parametre, dĺžky trénovacích behov a podobne. Takto by sme chceli optimalizovať naše modely na validačných dátach za účelom vykonania správneho vyhodnotenia a porovnania.

F.5 Zhodnotenie

V tejto práci sme navrhli, implementovali a overili niekoľko rôznych metód na riešenie problému automatickej transkripcie hudby súčasne.

Porovnali sme navrhnuté metódy medzi sebou a aj voči konkurenčným prístupom.

Ukázali sme, že aj tie najjednoduchšie architektúry sú schopné dávať zaujímavé, konkurencie schopné výsledky.

Naše súčasné modely však pravdepodobne kôli obmedzeniam časovým aj pamäťovým sú nie sú trénované optimálne.

V ďalšej práci by bolo vhodné sa venovať experimentovaniu s regularizáciou a rôznymi veľkosťami modelov a datasetov.

Navyše sme ukázali, že učenie zo surového audia použitím architektúry WaveNet dáva zaujímavé výsledky pri iných ako generatívnych úlohách, hodné ďalšieho skúmania. Aj keď veľkosti modelu a trénovacích časov sú oveľa väčšie ako pri ostatných, jednoduchších modeloch, kapacita je oveľa väčšia a teda model môže byť schopný naučiť sa o hudbe z danej reprezentácie oveľa viac. Tento smer preto konštatujeme zaujímavý a hodný ďalšej práce vo výskume MIR a špeciálne problémovej oblasti AMT.