Deep Feature Extraction and Music Language Modelling for Amateur Vocal Percussion Transcription

Dmitrii Mukhutdinov

Master of Science
Artificial Intelligence
School of Informatics
University of Edinburgh
2020

Abstract

Vocal Percussion Transcription (VPT) is a particular instance of Automatic Music Transcription problem. It is a problem of transcribing vocal imitations of percussive beats, also commonly known as beatbox, into a symbolic representation, such as MIDI. Since vocal imitation is arguably the most accessible form of musical expression, an efficient method for vocal percussion transcription would be a valuable and accessible tool for music production. However, few systems for vocal percussion transcription exist, and they have limited usability as music production tools. This work investigates how the modern deep learning approaches can be combined with classic signal processing approaches in order to improve the accuracy and efficiency of vocal percussion transcription. We show that audio features extracted by convolutional autoencoders (CAEs) trained on an unlabeled dataset of miscellaneous musical sounds and vocal imitations allow for more accurate classification of individual vocal percussion sounds when compared to classic audio feature descriptors. We also investigate how a deep generative language model for drum tracks can be applied for vocal percussion transcription and discuss the challenges which emerge on an attempt of such application.

Declaration

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Dmitrii Mukhutdinov*)

Acknowledgements

I would like to thank my supervisor, Dr Kartic Subr, for providing invaluable advice and guidance over the course of this project. Also, many thanks to my friends and family for their love and moral support.

Table of Contents

Chapter 1

Introduction

Vocal percussion is a form of musical expression which involves imitation of a percussive instrument via one's mouth. Elements of vocal percussion are used in a number of musical genres, and it also exists as an independent art form, most commonly referred to as *beatboxing*. It is also one of the most accessible forms of musical expression, since a crude imitation of a simple percussive beats can be done by almost anyone. Due to this, a system which is capable of transcribing an audio recording of vocal percussion into a symbolic musical form, such as MIDI, would be a valuable tool in the music production toolkit. Such a tool would provide an accessible and versatile way of rapid prototyping of percussive tracks.

Developments in the larger field of automatic music transcription (AMT) make one believe that such a system should definitely be possible. A substantial body of research exists on automatic transcription of instrumental music, such as piano performances, and a wide variety of approaches have been utilized for this task. Drum transcription, which is a task conceptually similar to vocal percussion transcription, has been extensively studied as well [80]. In recent years, as in many other applications, end-to-end deep learning models have become state of the art both for polyphonic music transcription [32, 67, 36] and drum transcription [77, 78].

Nevertheless, when one attempts to implement a system for vocal percussion transcription (VPT) using approaches borrowed from a wider body of research on automatic music transcription in general, one quickly encounters unique challenges which are not characteristic for other AMT tasks. The most substantial of those challenges is the extreme scarcity of annotated vocal percussion recordings. There are very few such datasets openly available, which prohibits the application of end-to-end deep learning models to VPT due to known dependency of such models on large amounts of data for

successful training. As a consequence, all existing studies on VPT are based on classic signal processing and machine learning techniques, and virtually all of them perform training and evaluation using relatively small datasets collected by authors themselves. Another unique challenge stems from the substantial diversity of vocal imitations of percussive sounds produced by people, which exceeds the diversity of sounds produced by different musical instruments of the same type, e.g. different pianos or drum kits. This diversity is especially significant in the case of amateur vocal percussion; yet, being able to transcribe amateur vocal percussion is essential if one aims to implement a music production tool which is accessible to a broad range of users. Those two problems exacerbate each other, since both of them make it harder to produce a VPT model which generalizes well: on the one hand, the underlying distribution of audio data is more complex, and on the other hand, one has less data samples from this distribution.

This work aims to address those challenges and produce a better method for transcription of amateur vocal percussion recordings using particular transfer learning approaches. The first approach is using deep autoencoders trained on unlabeled audio for audio feature extraction, which was recently reported to produce promising results on the related problem of *query by vocalization* [43]. The second one is exploiting the regular rhythmic structure of vocal percussion via application of deep generative musical language model trained on symbolic MIDI recordings of regular drum tracks [72]. We attempt to combine those approaches with classic methods of onset detection and classification in order to produce a system which does not require larger amounts of annotated vocal percussion data to train than existing VPT systems but produces more accurate transcriptions.

This study was largely enabled by the recent emergence of AVP Dataset (section 3.3.1), a largest annotated dataset of amateur vocal percussion so far. Even though this dataset is still too small to be suitable for training of end-to-end deep learning models, it is large enough to perform a comprehensive model performance analysis, and its existence allowed us to conduct this study without collecting data on our own, which seems exceptional for studies on VPT. This dataset fits our purposes very well, since it contains short recordings produced by people with little to no experience in vocal percussion, and transcribing such recordings well is the primary purpose of VPT method if it is intended to be used as a tool for quick production of drum loops.

Chapter 2

Background and Related Work

In this section, we begin with an overview of two fundamental problems in music information retrieval (MIR) which appear as subproblems in our proposed method for VPT: onset detection and tempo estimation. Then, we make a review of musical language models and their applications to music transcription. We follow with a short overview of a problem of drum transcription and a comprehensive literature review of existing research on vocal percussion transcription. We assume that the reader is familiar with basic digital audio processing techniques (e.g. short-time Fourier transform (STFT), standard feature representations like MFCCs, etc.) and with machine learning methods, including deep learning, in general.

2.1 Music Information Retrieval

2.1.1 Onset detection

Onset detection is the task of detecting the starts of distinct acoustic events in the audio signal, such as a musical note. Musical acoustic events can typically be divided into an initial *attack* phase — the short sudden change in the signal — and a longer *decay* phase, when the signal gradually return to a steady state. The short period of quick non-linear change which includes the attack and the beginning of the decay is called *transient*, and usually contains the most information about the nature of the signal. *Onset* is the moment in time where attack begins (Fig. 2.1).

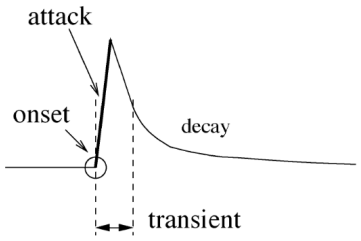


Figure 2.1: Diagram of a typical sound envelope of a musical event. Source: [9]

So, the problem of onset detection is reduced to the problem of finding the peaks of the function which represents how signal "changes" (and then finding the preceding local minima). The question is then how to define such function, called *onset novelty* function, in a meaningful way. The most straightforward (and the oldest) approach is using changes in local mean energy of the signal (usually formulated as *root mean square energy*), calculated with some window size and hop length [62]. However, energy-based onset functions are quite noisy and usually require extensive audio preprocessing to work well.

Functions based on spectral sound representation (STFT) usually work better. *High-frequency content* (HFC) function [40] is a simple examle of such function, defined as:

$$HFC(n) = \frac{1}{N}\sum_{k=0}^{N} k|X_k(n)|^2, \tag{2.1}$$

where n is time frame and k is a frequency bin of short-time Fourier spectrogram X. HFC is good at modeling percussive onsets, which usually happen as short bursts of noise across a wide range of frequencies. Another popular function is Specflux [40], which is basically a rectified Euclidean distance between consecutive frames of the spectra:

$$SF(n) = \sum_{k=0}^{N} \max(0, |X_k(n)| - |X_k(n-1)|)^2 \tag{2.2}$$

The rectification is done to only detect increases in frequency magnitudes and ignoring decreases (which are associated with *offsets*). A variety of modifications and preprocessing techniques exist for Specflux [25, 14], which were shown to perform quite well on different types of pitched musical events, and is one of the most popular signal processing methods for onset detection nowadays.

While those methods take into account only magnitudes of spectrograms, a Complex method [10] considers both magnitude and phase information. It is based on the fact that frequency phases for the consecutive frames of spectrogram change linearly for a steady-state signal. The onset novelty function is calculated as the magnitude of complex difference between current frame $X(n)$ and "expected" frame $\hat{X}(n)$ predicted by phases and magnitudes of previous frames. This method has shown good results on detecting vocal percussion onsets [19].

The process of determining suitable local maxima of onset novelty function is called *peakpicking* and involves thresholding (often relative to a local median of the function) in order to remove small insignificant maxima. Minimum delay time be-

tween onsets is also usually applied to filter out onsets which are too close to each other. The good parameter values for thresholding and inter-onset delay are usually determined empirically, and existing implementations of onset detection methods usually have default values which work well on large datasets of music [16, 41].

There are numerous ways to define an onset novelty function other than we described [9]. Deep learning approaches also exist: some focus on onset detection only [12, 63], some perform onset detection and event classification jointly [32, 77, 78]. However, they are way more computationally expensive, and in [19] it has been shown that HFC and Complex methods easily outperform DNNs when detecting vocal percussive onsets. In [54], Specflux has been shown to perform well on vocal percussion as well. Therefore, we consider those three methods for the purposes of our project.

2.1.2 Tempo estimation

The tempo is a fundamental characteristic of a musical piece which determines the perceived speed of performance. It is usually measured in beats per minute (BPM). Tempo defines how close in time musical events corresponding to the same beat are, and therefore can be estimated by using the information about time positions of musical events, provided by onset novelty function.

There are two most popular ways of performing such estimation. The first one is via using the *autocorrelation function* (ACF) of onset novelty function [18, 27, 7, 45]. Autocorrelation $R_{ff}(t)$ measures how similar the given signal f is to a copy of itself shifted in time by t. For a discrete real-valued signal $f(n)$, autocorrelation is defined as

$$R_{ff}(t) = \sum_{n \in \mathbb{Z}} f(n) f(n+t) \tag{2.3}$$

In practice, of course, signal f is finite, so the sum range is bounded to the length of the signal and the signal is padded with a copy of itself. Maxima of the autocorrelation of an onset novelty function are the values of delay for which onset novelty function is the most similar to itself, and therefore correspond to the dominant intervals between onsets. Since tempo and inter-beat interval are interchangeable values, tempo can be inferred via finding the maximum of ACF.

Another popular method works with picked onset positions instead and is called *inter-onset interval (IOI) clusterization* [22, 23]. The name is self-descriptive: the method is based on clustering times between pairs of onsets (not necessarily successive ones) and selecting the mean of the largest cluster as a dominant interval, which

determines the tempo.

Depending on the structure of the musical piece, both methods can (and usually do) produce several tempo candidates which are multiples of each other, e.g. 60 BPM and 120 BPM. In order to choose only one candidate, some prior restriction on the tempo is usually assumed, either in the form of the range of accepted tempos or a prior distribution.

Both approaches can be used to estimate either global or local tempo (e.g. using windowing for the latter), and many methods for tempo estimations are based on them. There is, of course, alternative approaches, such as based on joint probabilistic models for tempo and onset timings [48, 47] or deep neural networks [64, 13].

2.2 Automatic Music Transcription

Automatic music transcription (AMT) is a task of transcribing music from audio form to a symbolic form. There exist several *levels* of music transcription which are defined by the properties and complexity of the target symbolic representation. *Frame-level* transcription produces the estimates the target value (e.g. pitch) for each of short (e.g. 10ms) regular timeframes which span the duration of the musical piece. *Note-level* transcription produces a series of *notes* — distinct musical events, each having an associated onset time and a target value (pitch or class), and possibly additional attributes (duration, loudness) as well. *Stream-level* transcription produces several disjoint sequences (*streams*) of notes; each stream is usually associated with a particular instrument's voice. Finally, *notation-level* transcription produces a human-readable musical score (staff notation) [11]. Polyphonic piano transcription is arguably the most studied type of AMT, and state-of-the-art deep models achieve impressive results on that task [32, 67].

We will not review the literature on automatic music transcription at large: it is not necessary for understanding our work and conflicts with the concept of the page limit. We will focus only on relevant topics of musical language models for AMT and drum transcription.

2.2.1 MLMs for automatic music transcription

Natural language models, including deep ones, have been applied extensively for improving the accuracy of speech recognition [74, 38, 65]. Attempts to apply language

models trained on symbolic music data (music language models, MLMs) to the task of automatic music transcription in a similar manner have also been made.

Bourlanger-Lewandowski et al. [15] proposed a combination of RNN and restricted Boltzmann machine (RNN-RBM) as a language model for polyphonic MIDI tracks and demonstrated that how using such model for postprocessing of the output of probabilistic multi-pitch estimator leads to better transcription accuracy. Sigtia et al. [66] presented a similar study, which used neural autoregressive distribution estimators (NADEs) [76] instead of RBMs in the similar combined RNN-NADE language model. The same authors later elaborated on their results, combining RNN-NADE language model with several types of deep acoustic models for polyphonic piano music [67].

While the aforementioned works focused *frame-level* transcription — i.e. splitting the timeline into short regular timeframes and assigning — the study by Wang et al. [79] presented a note-level transcription system, which produced sequences of events in form (*time*, *pitch*). They used an RNN-RBM model adapted to such music representation together with a CNN-based onset detector and pitch estimator.

The study by Ycart et al. [81], which is, to the best of our knowledge, is the latest study which investigates the application of MLMs to transcription, extends previous approaches by introducing a concept of *blending model* — an additional feed-forward neural network which is trained to combine the predictions of acoustic and language models in an adaptive manner. They used a CNN-based acoustic model for piano transcription and LSTM-based language model. They demonstrated that introduction of the blending model improves the overall transcription quality.

All of the aforementioned works are focused on polyphonic piano transcription. To the best of our knowledge, there are no similar studies on other types of music transcription which involve deep MLMs. Nevertheless, those results are encouraging and suggest that the application of MLMs can be also beneficial for transcribing other forms of music.

2.2.2 Drum Transcription

Drum transcription is conceptually different from the transcription of pitched music because all percussive sounds usually contain a wide range of frequencies and differ from each other primarily in timbre. Thus, while pitched melodies can be transcribed purely using signal processing techniques, such as *fundamental frequency estimation* [39], reliable drum transcription usually requires some kind of machine learning. Drum

transcription is predominantly note-level. Early approaches to drum transcription can be roughly split into the following categories, suggested in [52]:

- *Segment-and-classify*: separate the audio into segments using onset detection and attribute one or more drum classes to each segment [46, 61, 30];
- *Separate-and-detect*: separate the audio into separate signals for each of the drum classes and detect onset times for each signal separately [20, 21];
- *Match-and-adapt*: match the whole signal against some predefined acoustic patterns for each drum class, update the patterns to be closer to the best matches, repeat until convergence [84, 82].

The fact that most drum tracks are polyphonic, i. e. several drums can be hit simultaneously, poses the key challenge in drum transcription since the spectra of simultaneous percussive sounds intersect a lot. This problem is tackled in various ways; e.g. segment-and-classify methods often utilize onset detection in multiple frequency bands [46].

There exist some works which use probabilistic models like HMMs to learn and employ prior knowledge of the rhythmic structure of drum tracks for transcription [68, 26]. However, with an emergence of more recent end-to-end deep learning methods [77, 78] this direction of research have been seemingly abandoned.

2.3 Vocal Percussion Transcription

Vocal percussion transcription can be (and often is) viewed as a kind of drum transcription. On the one hand, vocal percussion transcription is simpler than the general drum transcription, since vocal percussion is almost always monophonic: most people cannot convincingly imitate two percussive sounds happening in the same time due to the limitations of human vocal tract. Due to this, a simple ”segment-and-classify” approach is the most popular design pattern for VPT systems [33, 35, 34, 58]. On the other hand, VPT is more challenging in terms of the correct classification of percussive sounds due to the higher acoustic diversity of vocal imitations in comparison to real drum sounds and a smaller amount of openly available annotated data. Some of the works [69, 70] focus only on the task of individual beatbox sound classification, omitting the onset detection and segmentation steps, and thus do not propose an end-to-end transcription system. Other works approach VPT more like a speech recognition task

rather than a music transcription task [49, 28] and utilize models commonly applied to speech recognition. In fact, the existing literature on VPT can be split into three categories: *classification-only*, *segment-and-classify* and *speech-recognition-like*

2.3.1 Classification-only

A study by Sinyor et al. [69] aimed to figure out a combination of classification method and a feature set which would work the best for the classification of beatbox sounds. In order to do that, they utilized ACE [42]: an early Auto-ML library designed for music information retrieval tasks. They recorded a dataset of 1192 samples in total produced by three professional and three amateur beatboxers, each sample belonging to one of the five classes: kicks, p-snares, k-snares, opened and closed hi-hats. Using AdaBoost algorithm with C4.5 decision tree as base learners, a large set of spectral and temporal statistical features and a genetic feature selection algorithm, they achieved cross-validation 95.5% cross-validation accuracy for 5-class classification and 98.15% for 3-class classification (conflating different classes of snares and hi-hats together).

Stowell and Plumbley [70] investigated how well different acoustic features, calculated over a time frame with different delays since the true event onset, can separate the audio events into three classes: kicks, snares and hi-hats. They analyzed the performance of each feature independently by measuring KL-divergence of classes and also evaluated naive Bayes classifier with feature selection. The aim of the study was to determine a classification delay time which would yield good classification accuracy while still being acceptable for real-time VPT purposes. More importantly, for the purposes of this study, they collected *beatboxset1* [2] — a dataset of 14 professional beatbox recordings, which is currently the oldest openly available dataset of annotated vocal percussion.

2.3.2 Segment-and-classify

Kapur et al. [35] used energy-based onset detection and a neural network classifier with only a single feature (*zero-crossing rate*) to transcribe an input vocal imitation into a drum pattern of three classes: kicks, snares and hi-hats. They presented a GUI application which allowed user to train a classifier using individual imitations of each class. They reported cross-validation accuracy of 97.3% for their classifier; however, their test set only included 75 samples in total, recorded by two different people, which makes the generalizability of their method questionable. They also did not report any

metrics on full track transcription with onset detection.

BillaBoop system by Hazan [33] used a hybrid onset detection method based on HFC and mean energy for several frequency bands. It splits the detected events into separate attack and decay parts and extracts sets of feature descriptors from them separately. Classification to three classes (kick, snare, hi-hat) is performed using a random forest classifier. The system reported 90% accuracy on the test set of 62 utterances using 242 samples for training; however, no results on full transcription with onset detection were reported as well.

Hipke et al. [34] developed *BeatBox*, a GUI app which allows a user to provide personal imitations for a user-defined number of drum classes and uses k-NN classifier and an energy-based onset detection method to transcribe the vocal input. The paper does not report any quantitative metrics on the chosen transcription method, however, and instead focuses on a study of users' interaction with the app and their feedback.

Ramires [58] implemented an Ableton Live plugin for VPT called LVT which is capable of near real-time transcription. It uses HFC method for onset detection and uses a sequential forward feature selection method (SFS) when training a k-NN classifier on a user-specific dataset of imitations of three classes (kicks, snares, hi-hats). The study analyzes the importance of classifier personalization and actually measures the performance of the full system using F1-score on the dataset of 13 short recordings by 13 different participants, each recorded by three different microphones, which apparently makes this study one of the most comprehensive works on VPT for the time being. Also, the dataset used in this study is available online.

2.3.3 Speech-recognition-like

Nakano et al. [49] use HMM acoustic model for speech recognition trained on the speech recognition dataset in order to recognize beat patterns of *onomaetopoeic* phonemes (e.g. "din-don"). They only used two drum classes (kick and snare), to each of which a number of corresponding phonemes was attributed, so that the phoneme sequence produced by acoustic model could be turned into a sequence of class events. The latter was then used to extract a matching fixed drum pattern from the database, so no direct transcription of the input audio to symbolic form was actually performed.

Picart et al. [54] presented a study on onset detection and sound class recognition for beatbox performances. They collected a dataset which featured vocal imitations of pitched instruments as well as percussive imitations, featuring 1835 percussive of 5

classes and 1579 pitched imitations of 9 classes in total. The participants were professional beatboxers. They trained and evaluated an HMM acoustic model implemented using HTK Toolkit [83] in a manner similar to speech recognition systems, treating sequences of class events as word sequences, and achieved 9% word error rate (WER). However, the transcribed sequences did not contain musically relevant information, such as event onset times, and thus could not be considered a symbolic representation of music.

The most recent work on VPT (2020) by Evain [28] focused on recognizing individual beatbox sounds, which authors called *boxemes*, to a special alphabet for such sounds called *Vocal Grammatics* [4]. They used an HMM-GMM acoustic model implemented using Kaldi ASR toolkit [55] and recorded a dataset containing 80 different boxemes produced by one professional and one amateur beatboxers using five different microphones. The best result showed by the system was 15% *boxeme error rate* (BER, a metric equivalent to word error rate), which is impressive given the large number of different boxemes. However, the recordings only contained isolated repeated boxemes with pauses, and, as in the study by Picart et al., no information about onset times and rhythm was provided in the transcriptions.

2.3.4 Conclusion

Overall, comparing to other types of automatic music transcription, there is not much work done on vocal percussion transcription — the list of studies reviewed in this section seems to be virtually exhaustive. Most of these works are also quite similar in terms of design patterns: in fact, we observe only two major patterns in end-to-end VPT systems — ”segment-and-classify” and ”speech-recognition-like”. None of those works feature modern deep learning techniques, and virtually all of them involve the collection and annotation of custom datasets (which usually end up to be quite small); however, for some reasons, only for two of those studies [70, 58] the datasets are currently openly available. These are the two facets of the same problem of data scarcity. We found only one study which presented a system suitable for music production purposes together with comprehensive end-to-end evaluation results, and for which the system implementation itself and the evaluation dataset are available online — which is Ramires' study [58]. This motivates us to use the method proposed by Ramires as a baseline.

Chapter 3

Methodology and Implementation

3.1 Proposed approach

3.1.1 Research hypotheses

We attempt to improve upon existing results in two conceptually different ways. The first way is to improve the classification accuracy of individual imitations of percussive sounds by using better audio feature representations. Despite a high acoustic variability of vocal percussive imitations, human listeners can reliably understand which percussive sound is imitated, at least when the number of options is limited (e.g. it is very easy to tell a kick drum imitation from a hi-hat imitation, but telling a ride cymbal imitation from a crash cymbal imitation can be challenging). Therefore, a feature representation which places sounds which are *perceptually similar* from a human perspective close to each other in the feature space should allow for a reliable classification of vocal imitations. Recently, Mehrabi et al. performed a comparative study of acoustic feature representations for percussive sounds and their vocal imitations in terms of perceptual similarity [44, 43]. They conducted a human listening study to collect perceptual similarity ratings for pairs of real drum sounds and their vocal imitations from human listeners and compared several feature representations by how well the distance between two sounds in the feature space predicts a human-reported similarity rating. They found out that features extracted by deep convolutional autoencoders (CAEs), trained on a dataset of various synthesized and real musical sound as well as vocal imitations, perform significantly better than classic features like MFCCs in that sense, and propose to use them for *query-by-vocalization* (QBV) task — fetching the sound most similar to the vocal imitation from the database. Our first research hypoth-

esis is that such features will improve the classification accuracy for vocal percussion transcription purposes as well.

Our second research hypothesis is based on the fact that vocal percussive performances exhibit the same musical and rhythmical structure as real drum tracks, except for the fact that they are essentially monophonic. This brings us the idea that using the *musical language model* for real drum tracks to provide a prior distribution on possible percussive event sequences can improve the quality of vocal percussion transcription. Such a *musical language model* can be trained independently on MIDI recordings of real drum tracks, which are abundant [31, 56], unlike annotated beatbox recordings. The facts that an application of LSTM-based MLM have been shown to improve performance of polyphonic piano transcription [67, 79] and that there exist good LSTM-based generative models for drum tracks [72] suggest that this idea is worth considering.

3.1.2 System architecture

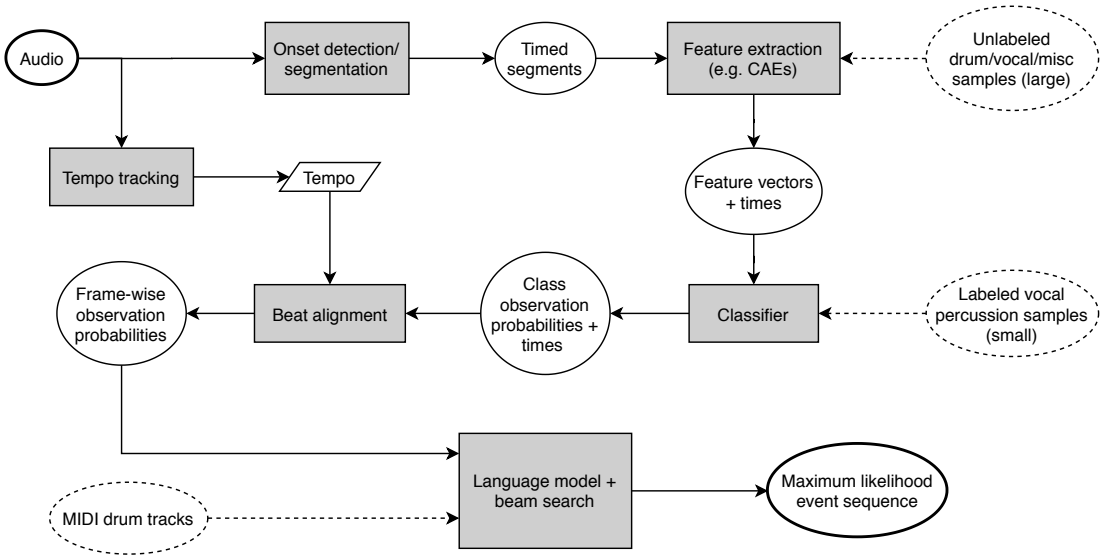


Figure 3.1: Diagram of proposed system architecture. White solid ovals represent the data flow in evaluation time, gray rectangles represent functional parts of the system, dashed ovals represent data necessary in training time.

Most of the existing VPT methods are based on "segment-and-classify" approach, which involves onset detection followed by segmentation of the audio between onsets and classifying each segment. Such approach results in a *note-level* transcription: a

series of classifier prediction probabilities $p(\gamma_T|x_T)$, where each classified segment x_T is associated with its precise start time T.

We propose a way to refine those predictions using a *frame-based* RNN language model, which operates on frames associated with positions of 16-th notes along the beat. The probability distribution over event sequences Γ represented by such model factorizes as:

$$p(\Gamma) = p(\gamma_1, \gamma_2, \ldots, \gamma_n) = p(\gamma_1) \prod_{t=1}^{n-1} p(\gamma_{t+1}|\gamma_{k<t+1}), \tag{3.1}$$

where t is *discrete* time (a frame number), and $\gamma_{k<t}$ are all the events preceding γ_t.

If we have a way of obtaining frame-level *observation probabilities* $p(x_t|y_t)$ for each frame, we then can infer the most likely event sequence given observations and a prior distribution $p(\Gamma)$ modeled by RNN via maximizing the posterior:

$$\begin{aligned} \arg\max_{\Gamma} p(\Gamma|X) &= \arg\max_{\Gamma} p(X|\Gamma)p(\Gamma) \\ &= \arg\max_{\Gamma} p(\gamma_1)p(x_1|\gamma_1) \prod_{t=1}^{n-1} p(\gamma_{t+1}|\gamma_{k<t+1})p(x_{t+1}|\gamma_{t+1}) \end{aligned} \tag{3.2}$$

Here is how we can obtain those frame-level probabilities from note-level transcription produced by "segment-and-classify" method. First, we need to transform precise onset times to discrete frame numbers. We do so by estimating a global tempo of the recording and quantizing onset times to the nearest beginning of a 16th note accordingly to the estimated tempo. This gives us probabilities $p(\gamma_t|x_t)$ for each discrete time frame t, where, if denoting a silence event (the absence of detected onset) as $\emptyset$, $p(\emptyset|x_t) = 1$ for each time frame t which does not have an associated onset and $p(\emptyset|x_t) = 0$ for each t which has one. Second, we notice that

$$p(x_t|\gamma_t) = \frac{p(\gamma_t|x_t)p(x_t)}{p(\gamma_t)} \tag{3.3}$$

However, the marginal observation probabilities $p(x_t)$ do not depend on event sequence Γ, so they can be omitted when maximizing (3.2). Therefore, after taking a logarithm for computational reasons, (3.2) becomes

$$\arg\max_{\Gamma} \left[\log p(\gamma_1|x_1) + \sum_{t=1}^{n-1} \left[\log p(\gamma_{t+1}|\gamma_t, h_t) + \log p(\gamma_{t+1}|x_{t+1}) - \log p(\gamma_{t+1}) \right] \right] \tag{3.4}$$

The only part of this equation which is not yet known is marginal probabilities of event classes $p(\gamma_t)$. They can be estimated simply via counting class occurrences in a large set of symbolic drum tracks (e.g. [31]). However, during the model evaluation, we (surprisingly) found out that using a naive assumption that class probabilities are uniform works better. Having all the probability estimates, we can compute an approximate solution to (3.4) via beam search algorithm (similarly to [67]). We use the beam size of 10 and number of 2 forward steps per iteration as hyperparameters for beam search throughout all experiments with the language model. The resulting maximum likelihood event sequence Γ is the output of transcription procedure.

The full system diagram is shown on Figure 3.1. Following the structure of our primary evaluation dataset, AVP Dataset (section 3.3.1), we aim to support four basic percussive event classes: kick drum (*kd*), snare drum (*sd*), closed hi-hat (*hhc*) and opened hi-hat (*hho*).

3.2 Implementation

3.2.1 Onset detection and segmentation

We consider three onset novelty functions: HFC, Specflux and Complex (see 2.1.1). All of those are spectral methods calculated over the STFT of the audio signal. We use the implementation from Aubio library [16] for each method, using default library parameters for audio preprocessing and peak picking (since those parameters are unavailable via Python bindings to Aubio). We consider several values for STFT window size and hop length for each method and select the best combination of method and values of those parameters (see Section 4.1).

Since the peaks of onset novelty function can appear during the attack phase of the musical event, making a given segment start exactly at the detected onset time may result in the small (and probably important) part of the event beginning not included into the segment. To avoid this, we employ *backtracking* — shifting the onset time back to the nearest local minimum of onset novelty function — and segment the audio between backtracked onset positions.

3.2.2 Tempo estimation and beat alignment

If application of language model is enabled, we perform quantization of detected onsets to the nearest 16-th note accordingly to estimated global tempo. We use tempo

estimation routine provided by Librosa library [41], which is based on the autocorrelation of onset novelty function (see 2.1.2), and uses log-normal distribution with a given mean as a prior for the tempo (we use prior mean of 90 BPM). We used the same onset novelty function as for onset detection, depending on the chosen onset detection method.

We assume that the time of the first onset is the beginning of the track (so the first detected onset is always quantized to the first 16-note of the first bar). If two or more onsets are quantized to the same note, only the first of them remains, the latter are discarded. The original onset times of remaining onsets are remembered, since they are matched against the ground truth onsets during evaluation.

3.2.3 Feature extraction

3.2.3.1 Baseline features

The first baseline feature set we use throughout our experiments is the set of first 20 MFCCs extracted from the first 4096 frames of the sound segment. MFCCs are computed over a single FFT frame of length 4096 obtained using a Hanning window. The choice of this feature set is due to the extensive use of MFCCs in speech recognition and existing VPT systems alike [35, 33, 49, 28]

The second baseline feature set is the one used by Ramires in his LVT system [58]. It is a set of two temporal and 52 spectral features calculated over the first 4096 frames of the audio segment. Temporal features are *root mean square energy* and *number of zero crossings*. Spectral features are:

- Statistical moments of the spectra: *centroid*; *spread*; *skewness*; *kurtosis*;
- Spectral form descriptors: *slope*; *decrease*; 95% *roll-off*; *spectral flux*; *spectral flatness* measured individually for each of the four frequency bands (250-500 Hz, 500-1000 Hz, 1000-2000 Hz, 2000-4000 Hz);
- 20 *MFCCs* and 20 *BFCCs* — the latter being the same as MFCCs but using Bark frequency scale instead of Mel scale.

This feature set is originally intended to be used with a nearest neighbors classifier which uses Euclidean distance metric. Also, the original LVT system applies a Sequential Forward Selection (SFS) feature selection algorithm during training. Since the original LVT software is implemented as a Max for Live plugin for a proprietary

DAW Ableton Live, evaluating the original system requires the purchase of the DAW. However, the re-implementation of the system in Python is simple, since it is based on the same segment-and-classify approach as our system, and using the k-NN classifier trained with SFS feature selection without a language model in our architecture yields a system which is conceptually identical to LVT. We perform this re-implementation using *scikit-learn* [53] and *mlextend* [59] libraries. Both of the baseline feature sets are implemented using Librosa library [41].

3.2.3.2 Deep feature extractors

For deep feature extraction, we use convolutional autoencoders proposed by Mehrabi et al [43]. It is a family of symmetric fully convolutional neural networks with the same basic architecture which differ in the sizes of strides and kernels. They are supposed to take the two-dimensional time-frequency audio representation (such as Mel spectrogram) as an input. The basic model architecture is a fully convolutional autoencoder with four 2D convolutional layers in its encoder/decoder. The encoder layers 1-4 have 8, 16, 24 and 32 kernels respectively, and the encoder layers, symmetrically, have 32, 24, 16 and 8 kernels. Each of the (de)convolutional layers is followed by a batch normalization layer and ReLU activation layer. In the decoder, deconvolutional layers are implemented as a bilinear 2D upsampling layer followed by a convolutional layer with stride (1, 1). A single-channel convolutional layer with stride (1, 1) is used as an output layer after the last ReLU activation of the decoder.

All the convolutional layers except the first encoder and last decoder layers are reported to have the same kernel size (10, 10) in the original paper. However, we note that using an even kernel size leads to incorrect size ratios between input and output tensors: e.g. for the decoder convolutional layer after the upsampling layer, the input and output tensors should have the same $W \times H$ dimensions, but using a kernel size (10, 10) with padding (5, 5) results in an increase of tensor size by 1 in each dimension. It could be resolved via an asymmetric padding, but we chosen to simply use kernel size of (9, 9) instead. Mehrabi et al do not report padding settings and the kernel size of an output convolutional layer, so we used symmetric zero padding for all layers and the output kernel size of (5, 5).

The original paper investigates 11 different subtypes of this basic architecture, which differ in the kernel size of the first encoder and last decoder layers and the stride sizes of internal convolutional layers of the encoder (upsampling layers in case of the decoder). They can be divided in three subgroups: *square*, *tall* and *wide* (Ta-

Type	L1/8 kernel	Strides of conv./upsampling layers				Encoded layer size (128×128 input)
		L1/8	L2/7	L3/6	L4/5	
square-1	(5, 5)	(2, 2)	(2, 2)	(2, 2)	(2, 2)	2048
square-2	(5, 5)	(2, 2)	(2, 2)	(2, 2)	(4, 4)	512
square-3	(5, 5)	(2, 2)	(2, 2)	(4, 4)	(4, 4)	128
tall-1	(5, 3)	(2, 2)	(2, 2)	(2, 2)	(2, 4)	1024
tall-2	(5, 3)	(2, 2)	(2, 2)	(2, 4)	(2, 4)	512
tall-3	(5, 3)	(2, 2)	(2, 4)	(2, 4)	(2, 4)	256
tall-4	(5, 3)	(2, 2)	(2, 4)	(2, 4)	(4, 4)	128
wide-1	(3, 5)	(2, 2)	(2, 2)	(2, 2)	(4, 2)	1024
wide-2	(3, 5)	(2, 2)	(2, 2)	(4, 2)	(4, 2)	512
wide-3	(3, 5)	(2, 2)	(4, 2)	(4, 2)	(4, 2)	256
wide-4	(3, 5)	(2, 2)	(4, 2)	(4, 2)	(4, 4)	128

Table 3.1: Summary of types of CAE architectures

ble 3.1). *Square* ones have symmetric kernel and stride sizes of different scales; *tall* ones have kernels which are larger in the frequency dimension and strides which are larger in time dimension; *wide* ones are the opposites of tall ones. Therefore, tall ones are expected to preserve more spectral information, while wide ones are expected to preserve more temporal information.

The time-frequency audio representation we use as an input to CAEs follows Mehrabi et al. We construct the magnitude spectrogram using the window size of 4096 frames ($\sim$93ms) and hop size of 512 frames. Then we apply Bark filterbank with 128 bands to the spectrogram and scale the magnitudes of the resulting *barkgram* using Terhardt's ear model equal-loudness curve [73]. The barkgram computation routine was implemented using Librosa library [16], and CAEs themselves were implemented using PyTorch.

Since the encoders are fully convolutional, they can accept barkgrams of any time length, and the size of encoded representation would change accordingly (Fig. 3.2). During training, we use barkgrams with the time length of 128 frames; the barkgrams computed over audio segments are either truncated or padded with zeros to that size. During the evaluation, we test different barkgram lengths and evaluate how does it affect the classification quality.

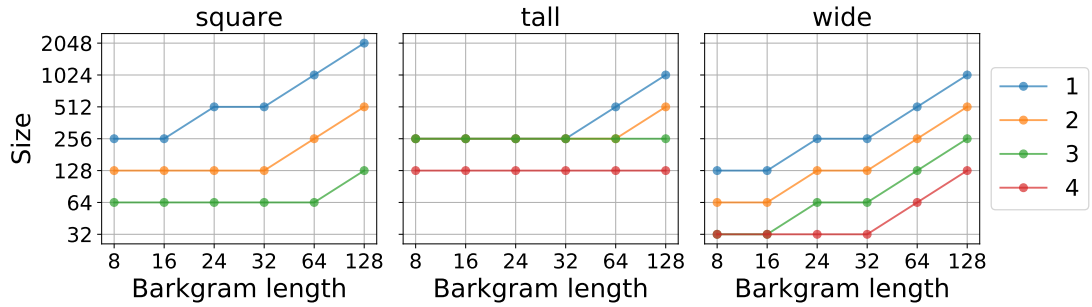


Figure 3.2: Correspondence of different lengths of input spectrograms to resulting sizes of feature representations for different CAE architectures, in log scale.

3.2.4 Segments classification

3.2.4.1 k-NN classifiers

The primary type of classifier we use is the k-nearest neighbours classifier which uses Euclidean distance metric and inverse-distance weighting of the neighbours. The motivation for this choice is twofold. First, in the original paper by Mehrabi et al., the Euclidean distance between CAE feature representations was used to predict human-reported similarity ratings, which suggest that exactly this type of distance metric should be used if we want to determine which sounds from the training set are the most perceptually similar to the query sound. Second, this is the classifier used by several previous works on vocal percussion transcription, including the Ramires' [58] LVT system, which we reimplement in Python and use as a baseline. The choice of inverse-distance neighbours weighting is motivated partially by the first of the aforementioned facts, and partially by the need for obtaining class probabilities for classified segments when using a language model: using only class counts for obtaining probability estimates from k-NN classifier results in only a small finite set of possible probabilities for each class, which is likely to restrict the extent of influence of the language model.

We evaluate k-NNs with different numbers of neighbours k. Regardless of the feature set, we use normalization of training data before k-NN training. As has been mentioned in section 3.2.3.1, we also use SFS feature selection during training the k-NN with Ramires' features.

3.2.4.2 Deep neural network classifiers

The second type of classifier we use is the deep neural network built on top of the convolutional deep feature extractor (Section 3.2.3.2). We obtain the classifier network for a given type of autoencoder architecture and a given length of input via taking the convolutional encoder, flattening its output and passing it to the smaller fully-connected network with three hidden layers of size 512 with ReLU activations. The motivation for considering this type of classifier is the following. On the one hand, we would like to test whether or not deep networks can generalize well when trained on a limited number of annotated vocal percussion samples available. On the other hand, the performance of k-NN classifiers is known to deteriorate when the feature space dimensionality grows, due to so-called "curse of dimensionality" [37], and the CAE-extracted features can be significantly high-dimensional for certain types of CAE architectures and input lengths (Fig. 3.2). Therefore, we hypothesize that if the dimensionality of CAE features would turn out to be high enough to significantly negatively affect the performance of the k-NN classifier, the DNN classifier would surpass it.

3.2.5 Language model

We use a pre-trained DrumsRNN model as a language model in our experiments [72]. DrumsRNN is one of the models present in the Google Magenta library [71], which uses TensorFlow [5] as backend. It is a recurrent neural network with three LSTM layers, each having 256 recurrent cells. DrumsRNN is originally designed for modelling polyphonic drum tracks which include nine types of drums: kick, snare, closed hi-hat, open hi-hat, low tom, mid tom, high tom, crash cymbal and ride cymbal. To model polyphonic drum events (the ones which include several different drums struck simultaneously), each possible subset of drum classes is modelled as a separate event type, resulting in $2^9 = 512$ distinct event types, including the silence event. Those are encoded using one-hot encoding before being passed as an input to NN. Apart from the encoded event type, DrumsRNN also receives six binary counters, which encode information about the next event's position along with the beat, quantized to a sixteenth note (a semiquaver). If the next event is starting in position of n-th semiquaver since the beginning of the track, then i-th binary counter (indexing from 0) has the value of $n/2^i \bmod 2$. Therefore, in case of tracks in $4/4$ meters, six such binary counters provide unique encodings for positions of all semiquavers across four bars. For each time step t, the model outputs softmax probability estimates $P(\mathbf{c_{t+1}} = i|\mathbf{c_t},\, h_t)$ for each

of 512 possible event types for next event $\mathbf{c_{t+1}}$. Here, h_t is RNN's hidden state.

Since amateur beatbox performances are essentially monophonic, the language model designed for polyphonic drum tracks has to be modified to produce probability estimates for monophonic percussive events. We do so by interpreting each polyphonic event $\mathbf{c_t}$ as an intermediate event in a process of sequence generation, s.t. given that $\mathbf{c_t}$ happened, each of the monophonic events $\gamma_t \in \mathbf{c_t}$ is equally likely to happen. More formally,

$$P(\gamma_t|\mathbf{c_t}) = \begin{cases} 1/|\mathbf{c_t}|, & \text{if } \gamma_t \in \mathbf{c_t} \\ 0, & \text{otherwise.} \end{cases} \tag{3.5}$$

Therefore, probability estimates for monophonic events can be obtained as follows:

$$P(\gamma_{t+1}|\gamma_t, h_t) = \sum_{\mathbf{c_{t+1}}} P(\gamma_{t+1}|\mathbf{c_{t+1}}) P(\mathbf{c_{t+1}}|\mathbf{c_t} = \{\gamma_t\},\, h_t) = \sum_{\mathbf{c_{t+1}} \ni \gamma_{t+1}} \frac{P(\mathbf{c_{t+1}}|\{\gamma_t\},\, h_t)}{|\mathbf{c_{t+1}}|} \tag{3.6}$$

Moreover, since we do not aim to support such drum events as toms and cymbals, we interpret those events as equivalent to musically most similar events among supported ones: low toms are equivalent to kicks, mid and high toms are equivalent to snares, cymbals are equivalent to opened hi-hats.

Such modifications may seem arbitrary and not necessarily preserving of the learned rhythmic structure of drum tracks; however, we found that the modified model is capable of producing monophonic drum tracks which sound believable. This is, of course, does not prove that such a model is representative of the rhythmic structure of beatbox recordings; this fact can only be checked during evaluation.

3.3 Data for system evaluation

3.3.1 Amateur Vocal Percussion Dataset

Amateur Vocal Percussion (AVP) dataset [19] is, to the best of our knowledge, currently the largest openly available dataset of annotated vocal percussion recordings. It contains recordings of 28 participants with little to no prior experience with vocal percussion. All of the recordings in the dataset were recorded using MacBook Pro's built-in microphone.

AVP dataset consists of two subsets: *Personal* and *Fixed*. In each of those, there are five recordings provided per each of the included participants. Four of those recordings containing repeated utterances which imitate one of the four drum classes: kick (*kd*), snare (*sd*), closed hi-hat (*hhc*) and opened hi-hat (*hho*). The fifth recording features participants' free rhythmic improvisation. The distinction between *Personal* and *Fixed* subsets is the following. In the *Personal* subset, participants used imitations of drum sounds they came up with themselves and were most comfortable with. In the *Fixed* subset, participants used predefined phonemes to imitate each of the drum classes: ”pm” for the kick drum, ”ta” for the snare drum, ”ti” for the closed hi-hat and ”chi” for the opened hi-hat. The dataset contains 9780 individual vocal imitations in 280 recordings in total.

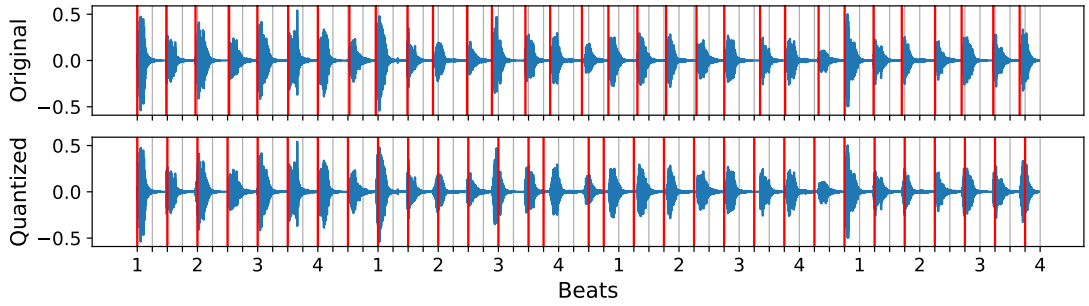


Figure 3.3: Sample improvisation recording from AVP Dataset. Red lines indicate onsets positions in time, gray lines indicate positions of semiquaver notes accordingly to detected tempo. Lower plot shows how the implied regular rhythmic structure breaks down after onsets quantization due to inaccuracies in the participant's performance — note how all the onsets are shifted one semiquaver behind the beat in the second part of the track.

We use recordings which contain repeated utterances as a source of individual sound segments for classifier training and use recordings of improvisations as evaluation data. However, we notice that although improvisation tracks in AVP Dataset are conceptually simple drum loops, many of them exhibit irregularities in tempo and rhythmic structure, since participants have little experience in vocal percussion. This may result in beat alignment which does not represent implied rhythmic structure (Fig. 3.3), and an inaccurate beat alignment, in turn, is likely to cause the language model to produce misleading transition probabilities for observed events.

3.3.2 Synthesized data

We mentioned before that there are two key concerns regarding the application of the language model designed for polyphonic drum tracks to the analysis of vocal percussion performances. The first concern is that such a language model may not necessarily represent typical vocal performances well enough. The second concern is that the performance of the language model may be disrupted by inaccuracies in the beat alignment, which are likely in case of amateur vocal percussion performance.

In order to understand whether or not those concerns are valid, we synthesized additional sets of audio files using vocal percussion samples from AVP Dataset. The procedure to synthesize the files was as follows. A set of vocal imitations of drum sounds, produced by the same person, was prepared. For each of four drum classes, two utterances were manually selected, one shorter than another. A percussive SoundFont file [60] was then manually created out of resulting 8 utterances using the Polyphone SoundFont editor [3]. Then, ten monophonic percussive MIDI tracks were generated randomly using the modified DrumsRNN language model, each track having the length from 4 to 8 bars and tempo in a range between 80 and 100 BPM. Audio files were then synthesized from generated MIDI tracks via Fluidsynth synthesizer [51] using the prepared SoundFont.

The procedure was repeated for the first five participants from both Fixed and Personal subsets of AVP dataset. This resulted in two sets of synthesized vocal percussion tracks, each containing 50 tracks: *AVP-gen Fixed* and *AVP-gen Personal*.

Since all of the synthesized tracks have known tempos and do not exhibit irregularities in onset timing, the beat alignment procedure should not produce irregular rhythmic patterns when applied to those tracks. Moreover, since their ground truth symbolic representations (MIDIs) were directly sampled from the language model itself, the language model is clearly representative of their structure. Thus, we can use this synthesized data in order to determine whether or not the application of the language model as proposed can improve transcription accuracy in principle.

3.4 Training of deep models

3.4.1 CAE feature extractors

In order to train convolutional autoencoders which are capable of representing real (or synthesized) drum sounds as well as vocal imitations, we need training data of

both kinds. We construct our training set via processing and combining data from four openly available datasets: *beatboxset1*, *ENST-Drums*, *200 drum machines* and *VocalSketch*.

beatboxset1 [2] is the oldest annotated dataset of vocal percussion recordings available online. It contains 14 tracks in total produced by 14 different participants, all of which are experienced beatboxers. For each track, two variants of ground-truth annotations containing onset times and classes of events are provided by two different experts. Besides of common imitations of kicks, hi-hats and snares, the recordings also contain breathing sounds, humming, singing and miscellaneous other vocalizations which do not fall into any particular category. Therefore, the precise transcription of those recordings is a challenging task which is out of the scope of this project. We use this dataset only as a source of vocal percussion samples produced by professional beatboxers for training our deep models. We obtain individual samples via segmenting the recordings using the first variant of ground truth onsets and extracting only the segments which correspond to supported event classes (kicks, snares, opened and closed hi-hats). This yields the dataset of 2317 utterances: 627 kicks, 703 snares, 882 closed hats and 105 opened hats.

ENST-Drums [29] is the dataset of annotated real drum performances which are recorded using separate microphones for each drum. We use this dataset as a source of individual drum sounds for deep feature extractors training, which we obtain via segmenting the recordings of kick, snare and hi-hat drums by ground truth onsets, obtaining 1376 individual samples. We do not include samples which correspond to any other drum class to our training set

200 drum machines [1] is the dataset of 6749 synthesized drum sounds produced by 200 different drum machines, including standard drum classes (kicks, snares, hats, cymbals, etc.) as well as more acoustically diverse sounds.

VocalSketch [17] is the large dataset of vocal imitations of various sounds, including imitations of drums, other musical instruments, synthesized sounds and every day sounds like a bird chirping. It contains 4429 vocal imitations in total. Mehrabi et al. also use this dataset as part of the training set in their original paper.

This results in a mixed dataset of 14817 real, synthesized and vocally imitated sounds. We randomly split it into training and validation sets, allocation 10% of data for validation.

All the samples in training and validation sets are transformed to 128×128 bark-grams as described in section 3.2.3.2. We train the autoencoders to reconstruct the

barkgrams using Adam optimizer with learning rate 10^{-3}, batch size 50 and mean squared error (MSE) loss for 25 epochs and select the model with best validation loss. The training process is repeated for all 11 variants of CAE architectures described in section 3.2.3.2.

Since our training set is different and smaller than the one used by Mehrabi et al. (which is reported to contain $\sim$ 39k individual sounds, more than twice larger than our dataset of $\sim$ 15k sounds), and also since unreported nuances of implementation could differ between our and the original CAE variants, we partially replicated the results of the original paper using the same data and methodology in order to make sure that those differences do not lead to a significant drop in the quality of the features. Those efforts are reported in Appendix A.

3.4.2 DNN classifiers

The training set for DNN classifiers is comprised of individual utterances from both Fixed and Personal parts of AVP Dataset (3317 and 2875 samples respectively, excluding utterances from improvisations), as well as utterances from *beatboxset1* (2317 samples). The resulting dataset contains 8509 vocal percussion samples, each one belonging to one of four classes (*kd*, *sd*, *hhc*, *hho*). 10% of this dataset are randomly allocated for validation.

Since the DNN classifiers we use are based on CAEs which we train separately in an unsupervised way, we initialize the weights of convolutional part of DNN classifier with the weights of previously trained encoder part of the CAE of the corresponding architecture before training. Then, we train the network in a supervised manner by minimizing cross-entropy loss using Adam optimizer. During training, the dropout layer with probability $p = 0.2$ is applied after each fully-connected layer, and the batch size of 50 is used.

We split the supervised training process into two phases, each lasting for 30 epochs: FCN head training and fine-tuning. During the first phase, the learning rate is 10^{-3} and only the parameters of fully-connected layers are updated. During the second phase, the learning rate is $2 \cdot 10^{-4}$ and all layers, including the convolutional ones, are updated. The model which achieves the best classification accuracy on the validation set over all epochs is chosen as the best one.

Chapter 4

Evaluation and Discussion

We evaluate the performance of onset detection and transcription using precision, recall and F1 metrics. During the evaluation, we match estimated onsets with ground truth using a tolerance window of 50 ms, following the established standard of MIREX onset detection challenge [24]. We use routines from *mir_eval* Python library [57] for calculation of the scores.

4.1 Onset detection

Onset detection is the first step in our system's pipeline, which defines which audio segments are passed further to the feature extraction and classification steps. Since our pipeline is built in such a way that each detected onset corresponds to an audio segment which then gets classified, onset detection is a bottleneck — even if the classification is perfect, the performance of the whole system cannot exceed the performance of the onset detection method. Therefore, we should choose the best onset detection method among available ones and use it as a default.

We compare three methods (HFC, Complex and Specflux) by their precision, recall and F1-score on the whole AVP dataset (both Personal and Fixed parts, including hits-only recordings). All three of those methods accept window size and hop size of STFT as parameters. We perform a grid search for each method using values (512, 1024, 2048) for window size and values (128, 256, 512) for hop size. The results are shown in Fig. 4.1. We see that Complex method is the best in terms of precision, but is the worst in terms of recall, which means that it yields less false positives but more false negatives. However, all methods do significantly better in terms of recall than in terms of precision, which means that each method yields more false positives than false

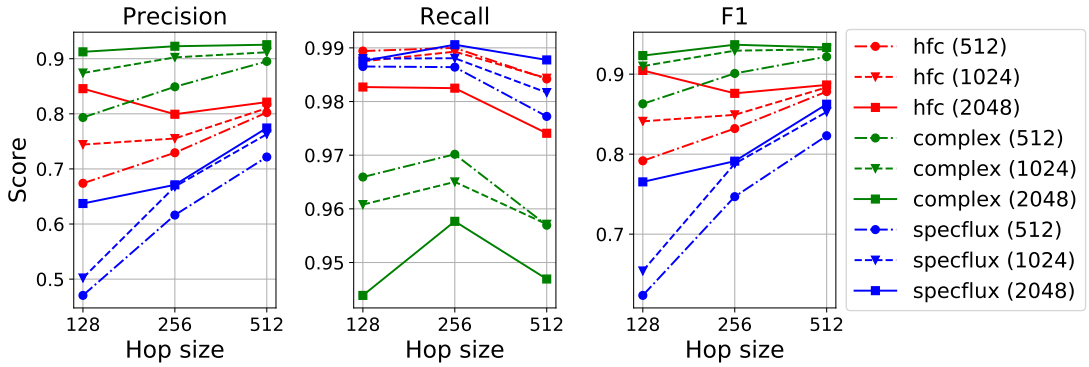


Figure 4.1: Precision, recall and F1 score for three onset detection methods with different window sizes and hop sizes.

negatives. Since Complex method wins more in precision than it loses in the recall, it is consistently better in terms of F1-score, and it yields the best F1-score with a window size of 2048 and hop size of 256. Therefore, we choose Complex method with those parameters as a default onset detection method used throughout the experiments that follow.

4.2 Model and feature selection for classification

4.2.1 k-NN classification

We compare the performances of k-NN classifiers using different audio feature representations in the task of classification of isolated utterances of AVP Fixed and AVP Personal datasets, excluding utterances from improvisations recordings. Since the resulting datasets are balanced ($\sim$ 800 utterances per class for Fixed, $\sim$ 700 — for Personal), we use 5-fold cross-validation accuracy as metric.

All eleven architectures of CAE feature extractors are first compared using 3-NN classifier with several input spectrogram lengths (Fig. 4.2). The best configurations from each of the three CAE families (*square*, *tall* and *wide*) were then compared with the baseline features (20 MFCCs and Ramires' features) (Fig. 4.3). In the case of Ramires' features, SFS feature selection based on 5-fold cross-validation score has been applied on the dataset before computing the final validation score.

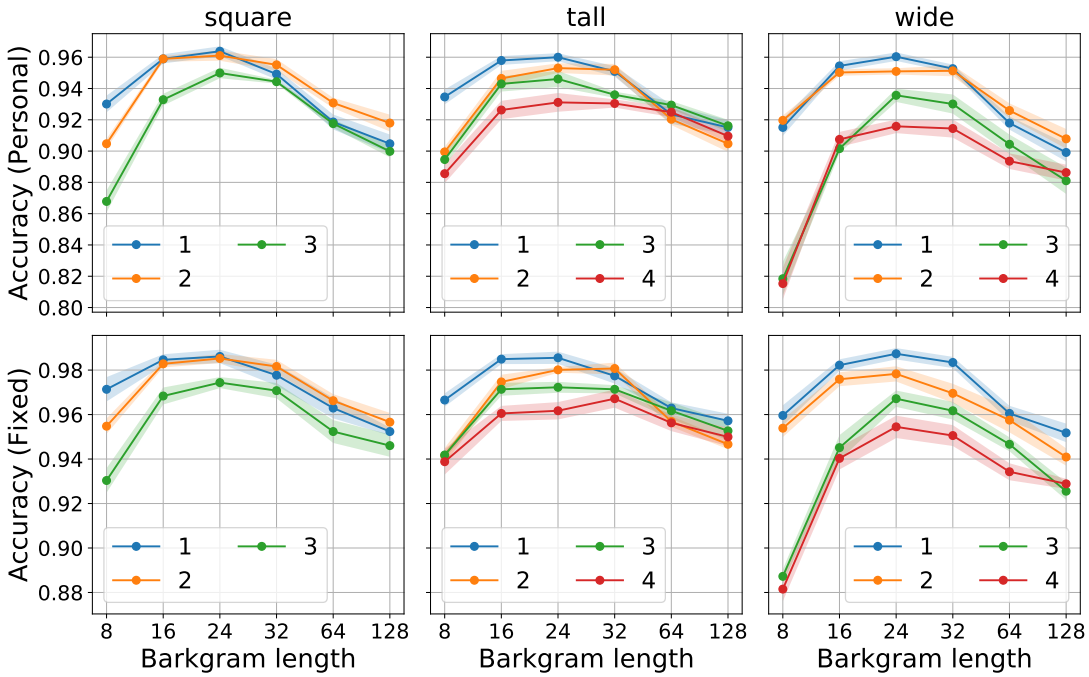


Figure 4.2: 5-fold cross-validation scores of k-NN ($k = 3$) with deep feature extractors using different lenghts of input spectrograms (top row: **AVP Personal** segments, bottom row: **AVP Fixed** segments) Lines indicate mean scores over 5 folds, bands indicate standard error of the mean score.

4.2.2 DNN classifiers validation

DNN classifiers based on the ”largest” CAE architectures from three families (*square-1*, *tall-1*, *wide-1*) were trained on the combined dataset of utterances from AVP Dataset and beatboxset1 as described in Section 3.4.2. DNN architectures corresponding to input barkgram lengths of (24, 32, 64, 128) were compared by their best validation scores. The results are summarized in Table 4.1.

4.2.3 Discussion

On the Figure 4.2, we see that cutting/padding the input barkgram of the classified to 24 time frames yields the best classification accuracies for almost all variants of CAE feature extractors on both datasets. This pattern is also seen in the validation scores of DNN classifiers: longer inputs generally lead to lower accuracy on the validation set, with the two out of three CAE encoder types reaching the best scores with the input length of 24 frames.

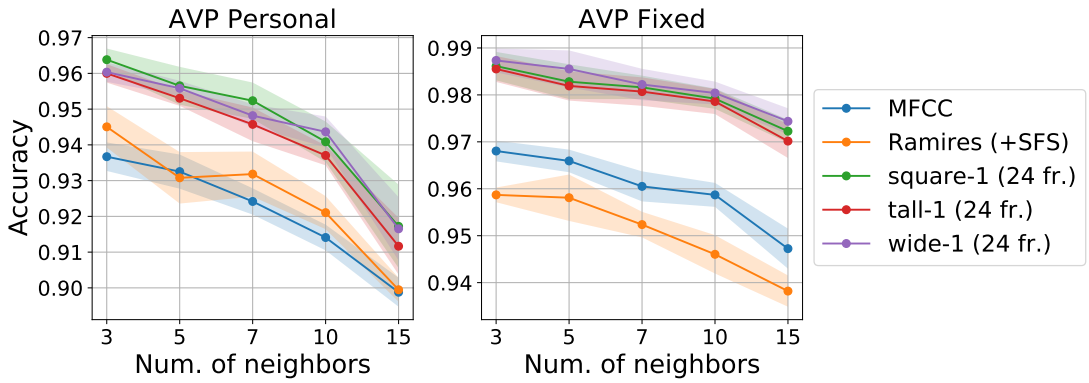


Figure 4.3: 5-fold cross-validation accuracy scores for k-NN with baseline features and with three best CAE feature sets. Results are given for different numbers of neighbors k. Lines indicate mean scores over 5 folds, bands indicate standard errors of the mean score. CAE features outperform baselines on both datasets.

CAE type	Input barkgram length			
	24	32	64	128
square-1	**0.9683**	0.9647	0.9612	0.9506
tall-1	0.9718	**0.9765**	0.9659	0.9518
wide-1	**0.9718**	0.9683	0.9612	0.9542

Table 4.1: Best accuracy scores on validation set during training of different configurations of DNN classifiers. Best scores for each CAE type are in bold.

This can be explained by the fact that vocal imitations of percussive sounds are generally short, and that for each particular audio segment, the most relevant acoustic information is at its beginning: during the *attack* and *transient* phases of the sound. Cutting or padding the barkgram of a classified segment to 24 frames corresponds to taking into account only the first ~ 0.28 seconds of the segment (for sample rate 44100 Hz). The mean duration of the segment is 0.456 s for AVP Fixed and 0.433 s for AVP Personal. Thus, using 24 frames of the segments' barkgram means taking into account the first $2/3$ of the classified segment on average. Apparently, providing less or more audio information to the CAE leads to less informative feature representations: there is not enough information when the barkgram is shorter and there is too much irrelevant data (or zero padding) when it is longer. The validation scores of DNN classifiers also suggest that the problem with longer barkgram lengths is not the curse of dimensional-

ity which starts to affect k-NN classifier when the size of feature representation grows, since the performance of DNNs also decreases with the longer inputs.

Moreover, for each CAE family, the architectures which yield larger feature representations (the ”first” ones) perform generally better than those which yield smaller ones. This means that for successful k-NN classification it is more important to have more informative representations than to reduce their dimensionality.

Figure 4.3 shows that the ”largest” CAE feature representations from all three CAE families, computed over the first 24 frames of the segment barkgram, outperform both baseline features (20 MFCCs and Ramires' features with feature selection) on both datasets. There is no significant difference between the performances of different CAE families. We also see that an increasing number of neighbours k for k-NN generally leads to a decrease in performance for every feature set.

Interestingly, there is no significant difference between the performance of MFCCs and pre-selected Ramires' features on AVP Personal dataset; and on the AVP Fixed dataset, MFCCs features even outperform Ramires' ones. This is surprising since 20 MFCCs are included in Ramires' feature set, so the feature selection procedure is supposed to select them if they are the best performing subset of features. However, this is not necessarily true, since sequential forward selection is not an exhaustive feature selection. The algorithm does not remove previously selected features, so the only way 20 MFCCs can be selected is by being incrementally added as the features which improve the validation score the most for each of the first 20 steps of the algorithm. This can easily be not the case if other features perform better than individual MFCCs during early steps of the algorithm, which evidently happens in case of AVP Fixed dataset. As a result, the algorithm selects a suboptimal set of features.

Also, the classification accuracy on AVP Fixed is generally higher than the accuracy on AVP Personal for each feature set. This is expected since AVP Fixed set, which consists of fixed onomatopoeic utterances for each drum class, is more homogeneous than AVP Personal set, which consists of participants' personal imitations of drum sounds. Personal vocal imitations predictably differ more between people than fixed ones.

4.3 Transcription of AVP dataset

4.3.1 Experiments

We analyze the performance of the whole system on the problem of transcription of improvisation recordings from AVP Dataset using precision, recall and F1 metrics. We compare three DNN classifiers (based on *square-1*, *tall-1* and *wide-1* CAEs, using input length 24), 3-NN classifiers with three sets of CAE features which performed the best during cross-validation (*square-1*, *tall-1* and *wide-1* using input length 24, see Section 4.2.1), and two baselines — 3-NN with MFCC features and 3-NN using Ramires' features and SFS feature selection. Each classifier is evaluated with and without the application of the language model.

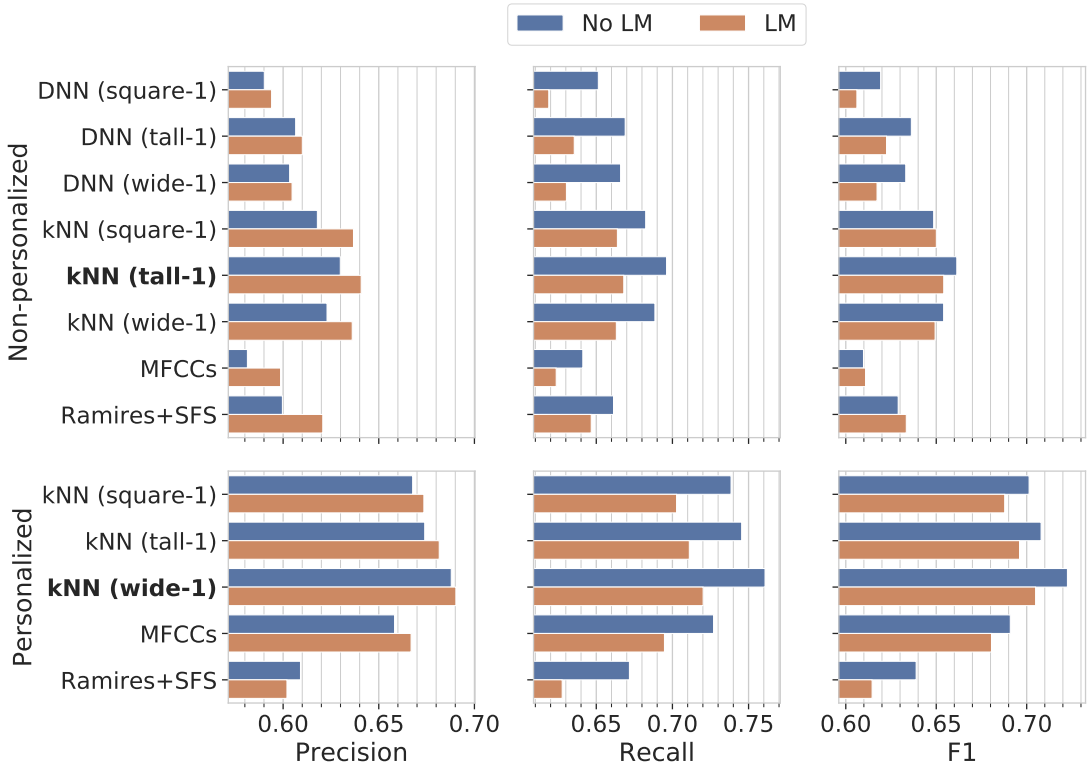


Figure 4.4: Precision, recall and F1-score obtained on transcription of improvisations from **AVP Personal** dataset. Top row: non-personalized training/evaluation, bottom row: personalized training/evaluation. Model with the best F1-score is shown in bold.

We train and evaluate all k-NN based classifiers in two ways: *non-personalized* and *personalized*. The former means that the classifiers are trained using the individual utterances of all participants included in the dataset (3317 samples by 28 participants in

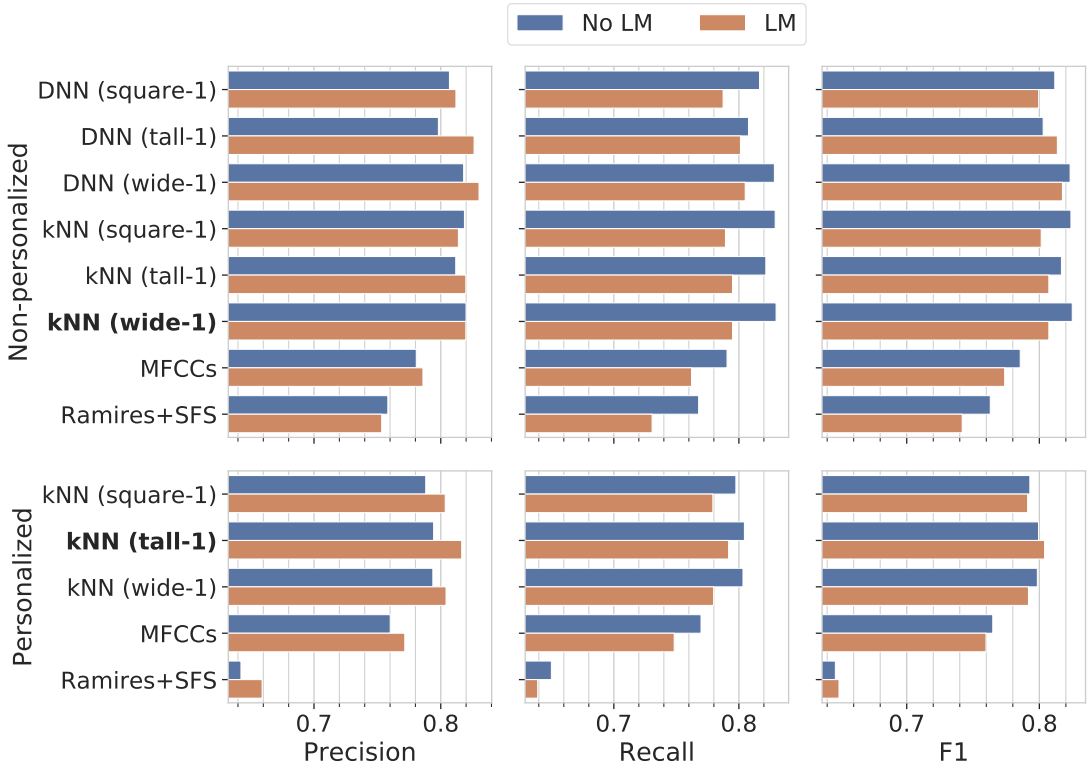


Figure 4.5: Precision, recall and F1-score obtained on transcription of improvisations from **AVP Fixed** dataset. Top row: non-personalized training/evaluation, bottom row: personalized training/evaluation. Model with the best F1-score is shown in bold.

AVP Fixed, 2875 samples by 25 participants in AVP Personal), and the improvisations of all participants are transcribed using the same classifier. The latter means that we train an individual classifier for each participant using only utterances produced by them ($\sim 90 - 150$ samples per participant) and transcribe the improvisation recording of that participant using this individual classifier. Results for AVP Personal dataset are shown in Fig. 4.4, for AVP Fixed — on Fig. 4.5.

4.3.2 Discussion

First, we see that k-NNs with CAE-extracted features show the best performance on both datasets, both for non-personalized and personalized setups. All three variants of those outperform both of the baselines for each setup; they also clearly outperform DNN classifiers on Personal dataset but are roughly on par with them on Fixed dataset. This shows that DNN classifiers, trained on the joint dataset of vocal utterances from AVP Fixed, AVP Personal and beatboxset1, did not generalize well enough: they are

able to classify relatively homogeneous onomatopoeic utterances from AVP Fixed as well as k-NNs trained only on such utterances but failed to do so in case of more heterogeneous personal imitations.

Second, we see that using personalized classifiers decreases performance on AVP Fixed but significantly increases it on AVP Personal for all feature sets. Again, this confirms the fact that it is much harder to generalize over personal vocal imitations of different people than over the same phonemes uttered by different people: while in the latter case, having different people's utterances in the training set apparently helps to avoid overfitting, in the former case it simply introduces too much noise to predictions. The overfitting is the most apparent in case of Ramires' features with feature selection, which displays the most significant drop in performance when trained in a personalized setup on AVP Fixed.

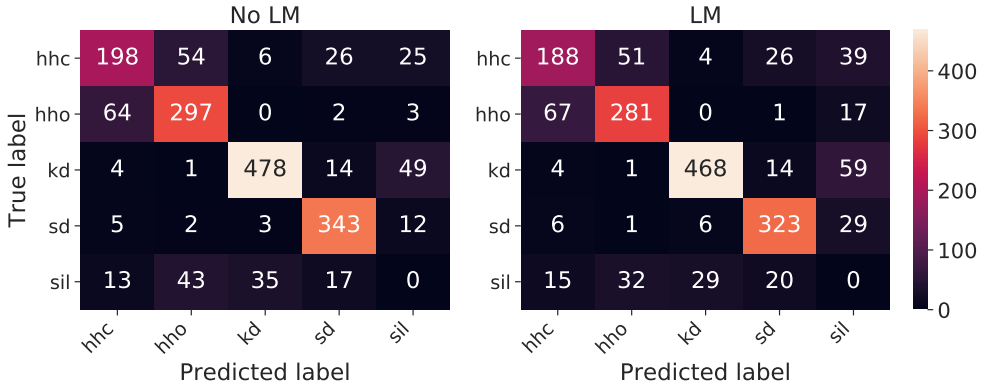
Finally, we see that the application of the language model generally leads to slightly better precision but overall worse F1 score, i.e. it causes slightly less false positives but significantly more false negatives. In the section that follows, we investigate why this happens in more detail.

4.4 Influence of the language model

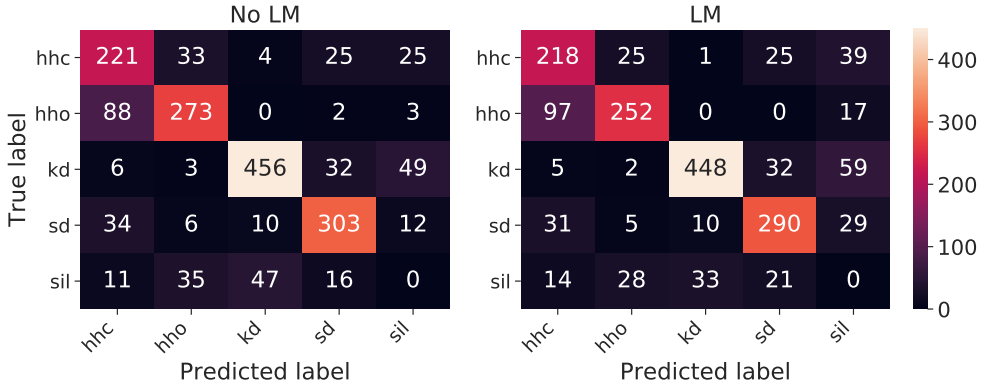
4.4.1 Experiments

In an attempt to understand why the application of the language model degrades the transcription quality, we check three hypotheses. The first one is that imprecise onset timing and tempo in the amateur vocal performances lead to irregular quantization and beat alignment, which leads to rhythmically improbable event observation probabilities for the language model (see Section 3.3.1). The second one is that low number ($k = 3$) of considered nearest neighbours for k-NN classification causes very small or zero observation probabilities for event classes other than the most probable one, which leaves little freedom for the language model to fix classification errors. The third one is that the language model designed to represent polyphonic drum tracks is simply not a good model for amateur vocal percussion tracks.

We check these hypotheses as follows. First, we analyze confusion matrices produced by the classifier which perform the best on AVP Fixed dataset in non-personalized setup (k-NN with *wide-1*) and the baseline k-NN with MFCC features (Fig. 4.6). We do that in order to see patterns of classification errors which speak for or against the



(a) k-NN (wide-1)



(b) MFCCs

Figure 4.6: Confusion matrices produced by (a) 3-NN with *wide-1* CAE features and (b) 3-NN with MFCC features with and without the application of language model on AVP Fixed improvisations (non-personalized setup). *sil* denotes the silence event (i.e. the absence of the onset at a given time $\pm$50ms tolerance window)

first hypothesis.

Second, we evaluate k-NN classifier with MFCC features using different values for the number of nearest neighbours $k \in (3, 5, 7, 10, 15)$ on both AVP Fixed and AVP Personal in non-personalized setup. We measure resulting precision, recall and F1 measures for each of the drum classes (*kd*, *sd*, *hhc*, *hho*) separately and also in total (using total numbers of true positives, false positives and false negatives). Additionally, we perform the same evaluation but skipping the onset detection phase altogether and providing the ground truth onsets instead (Fig 4.7). This allows us to gain evidence for or against the first and second hypotheses.

Third, we evaluate our system on the data synthesized from MIDI generated via the same language model as described in 3.3.2 (AVP-gen Fixed and AVP-gen Personal

Dataset	Detected onsets		Ground-truth onsets	
	No LM	LM	No LM	LM
AVP Fixed	89	144	0	41
AVP Personal	31	106	0	53

Table 4.2: Total number of undetected true onsets observed during transcription of real vocal percussion improvisations. The onset detection method used is the same in all cases; the increase in undetected onsets observed when applying a language model is due to onsets discarded during quantization.

sets, each containing 50 recordings). With the synthesized data, we know the exact tempo which was used to generate the track, and we pass it as a known parameter to the beat alignment routine. We exclude the participants whose utterances were used for data generation from the training set for k-NN. We also perform the same evaluation using ground truth onsets (Fig. 4.8). Since the synthesized tracks are generated using the same language model, this gives us evidence for or against all three hypotheses.

4.4.2 Discussion

The first important observation is that the numbers of the false positive predictions of silence — i.e. undetected true onsets — stay the same when the classifier changes, but grows significantly for each drum class when the language model is applied (Fig 4.6). The independence of those numbers from the classification method is expected, since undetected onsets never get classified — those numbers only depend on the onset detection method. But the application of the language model itself also cannot directly influence the number of undetected onsets. However, before LM is applied, onsets are *quantized*, and those onsets which get quantized to an already occupied 16-th note get discarded. Apparently, a significant number of such discarded onsets are actually correctly detected onsets, which can happen if onsets are not positioned correctly along the 16-th note grid inferred via tempo estimation. This is a strong evidence in favor of the first hypothesis.

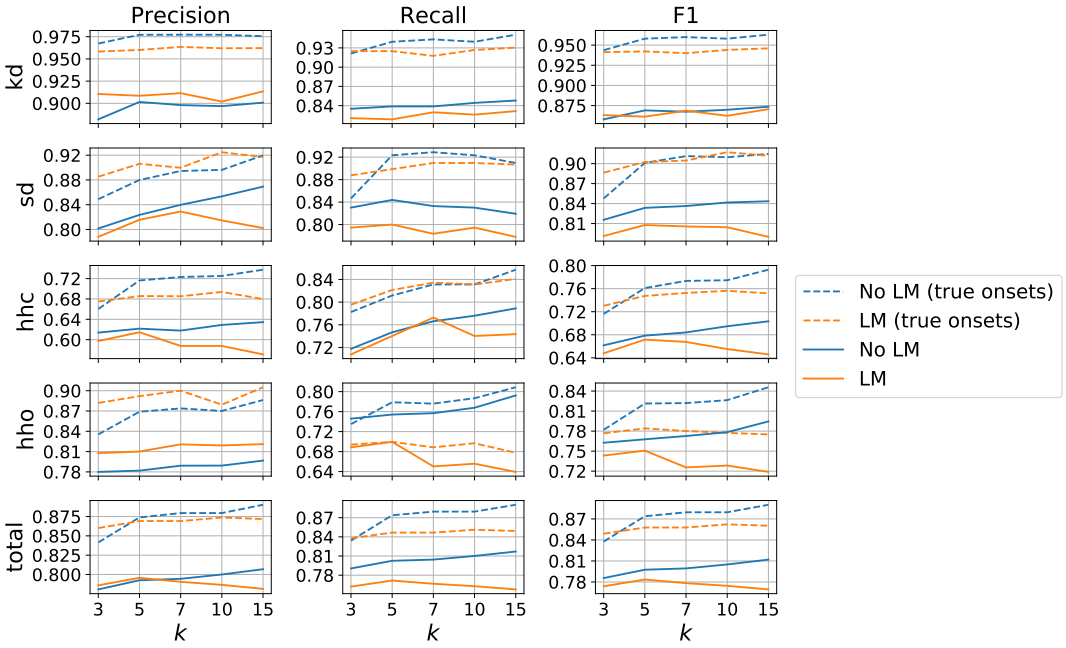
Figure 4.7 suggests that the low number of nearest neighbors k is not a significant factor, since the quality of transcription with a language model does not really grow with increasing k for both Fixed and Personal datasets. It also shows that providing ground-truth onsets to the pipeline still does not result in any performance gains for

the setup with a language model in comparison to a setup without one. We have also determined that when language model is applied using ground truth onsets, quantization still causes a significant number of those onsets being discarded (Table 4.2). This, again, strongly suggests that tempo irregularities in the recordings and imprecision of tempo estimation inevitably lead to bad event quantization, which causes transcription errors even before the improperly quantized events are passed to a language model.

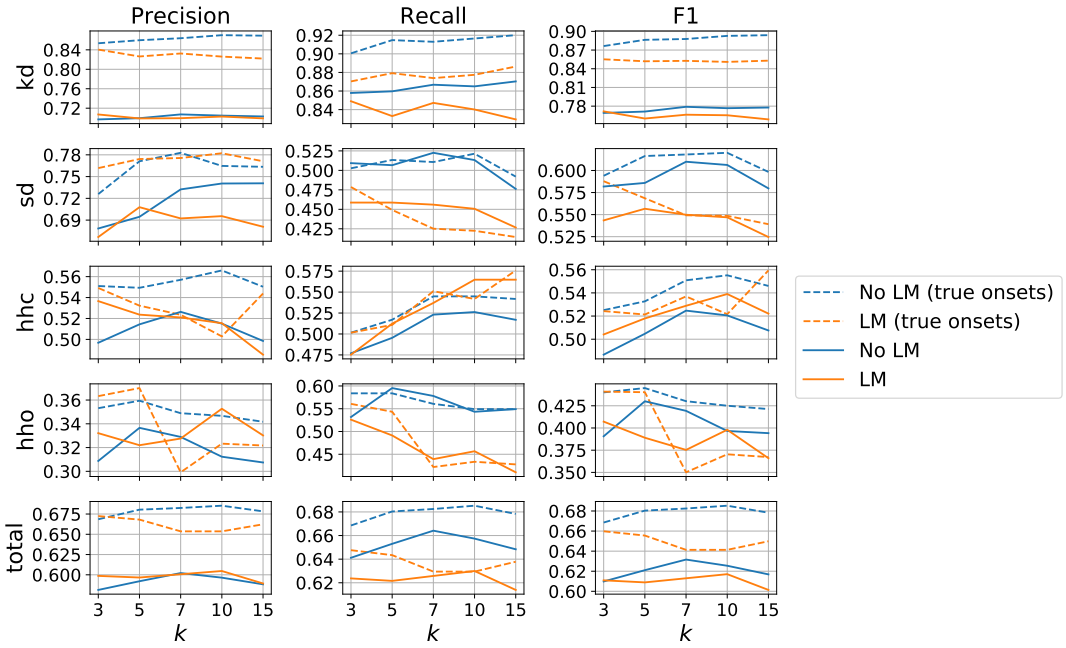
Using synthesized data with known tempo allows us to rule out those factors during analysis. Figure 4.8 shows that applying a language model to a transcription of such data when using ground truth onsets results in higher or equivalent total F1-score on both Fixed and Personal sets. Moreover, on the Personal set, it also results in a better or equivalent score when using detected onsets. However, this improvement is small. Lack of improvement when using detected onsets can still be explained by errors in onset detection. Given how sharply the quality of transcription of kick drum events (*kd*) increases on both datasets when the ground truth onsets are provided, we can conclude that kick onsets are not reliably detected on synthesized data. Since kick drum events are usually the backbone of the rhytmic structure, lack of those events observed may disrupt the predictions of the language model.

The lack of significant and consistent improvement when using a language model with ground truth onsets, however, requires further explanation. We can see a consistent improvement in total F1-score for AVP-gen Personal dataset, but it is quite small. However, the overall performance on the AVP-gen Personal is quite poor, which is explained by the fact that no personal utterances of participants selected as sample sources for track synthesis are present in the training set. If the language model is permissive enough, significant improvements over the initially noisy classifier predictions are unlikely. The lack of improvement on AVP-gen Fixed dataset, where overall performance is way better, is more puzzling, but can also be explained by excessive permissiveness of a language model. We also tested the language model with non-uniform marginal class probabilities (results not included due to the page limit) using class counts from *beatboxset1* and Groove MIDI Dataset [31], but this made overall performance even worse.

Overall, we see that our approach for language model integration turns out to be impractical due to high sensitivity to errors produced by onset detection and naive quantization based on a global tempo. Ruling out those errors using synthesized data also leaves us with inconclusive results; further research is needed to determine better ways of language model application to vocal percussion transcription task.

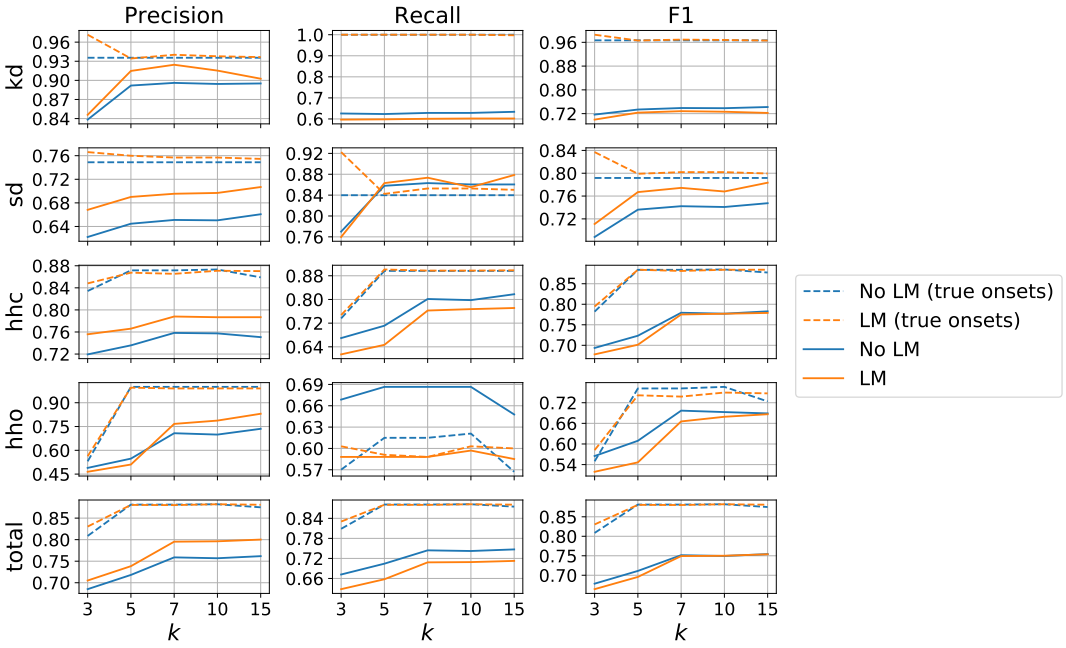


(a) AVP Fixed

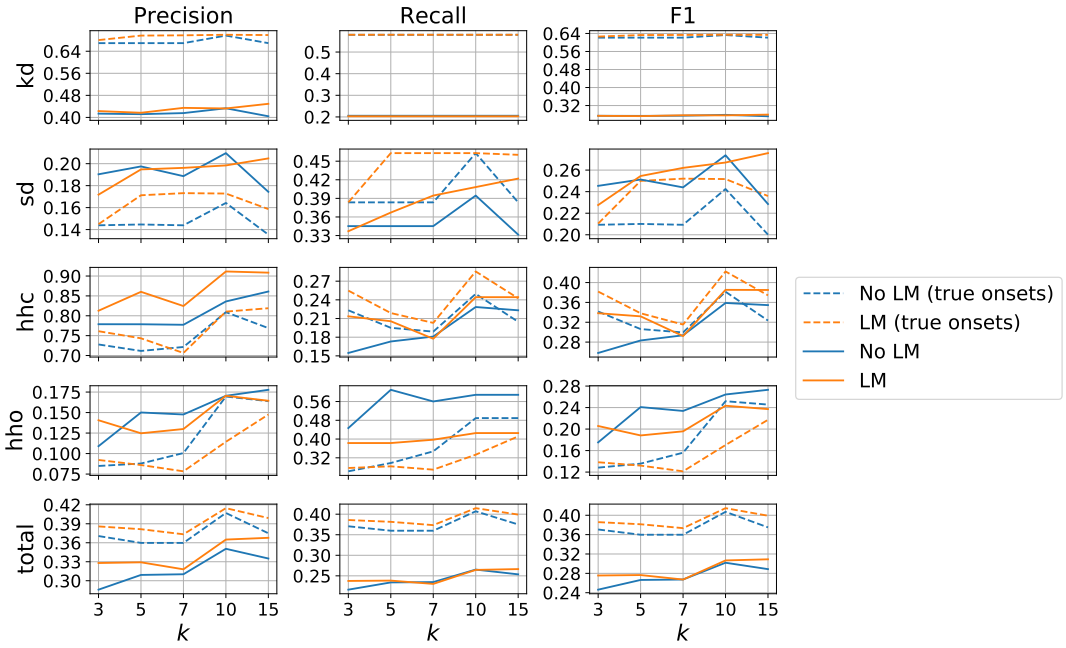


(b) AVP Personal

Figure 4.7: Comparison of system performance with and without language model on transcription of **real data** (improvisations from AVP Fixed and AVP Personal). Solid lines represent scores obtained when including onset detection step, dashed lines represent scores obtained when using ground truth onset times instead. k is the number of neighbors used in k-NN. Using a language model (LM) generally leads to a drop of transcription accuracy for all drum classes.



(a) AVP-gen Fixed



(b) AVP-gen Personal

Figure 4.8: Comparison of system performance with and without language model on transcription of **synthesized data** (*AVP-gen Fixed* and *AVP-gen Personal* sets). Ground-truth tempo is provided to the system for each experiment. Solid lines represent scores obtained when including onset detection step, dashed lines represent scores obtained when using ground truth onset times instead. k is the number of neighbors used in k-NN.

Chapter 5

Conclusion

5.1 Summary

This work proposes a novel method for vocal percussion transcription (VPT) which is based on a combination of traditional signal processing methods and deep learning techniques. Like most existing methods for VPT, our one is based on a ”segment-and-classify” approach: using onset detection to determine the positions of percussive events and classifying corresponding individual segments. Our contribution to the existing work in the field is twofold. First, unlike our predecessors, we use audio feature representations extracted by deep convolutional autoencoders instead of classic audio feature descriptors, such as MFCCs. Second, we propose a way to adapt a deep generative model for drum tracks as a language model for vocal percussion transcription.

We evaluated several variations of the proposed method using several types of deep feature extractors and two types of classifiers (k-NN and DNN) on the largest openly available annotated dataset of vocal percussion, Amateur Vocal Percussion (AVP) dataset. We have shown that using a k-NN classifier with CAE-extracted features results in better transcription quality when compared to using MFCCs and an approach proposed in LVT system [58], which is one of the latest VPT systems described in the literature. We have also shown that DNN classifiers based on the same CAE architectures we used for feature extraction cannot generalize well over a diverse set of amateur vocal imitations of percussive sounds and that they perform worse than k-NN classifiers trained on a smaller but more specific dataset. Moreover, we demonstrated that using a small dataset of a single person’s imitations for classifier training is significantly better than using a joint dataset of imitations produced by several people when transcribing recordings by this particular person. This shows that a practical

VPT method should necessarily be adaptable for a particular individual user.

However, we also found out that our approach to language model integration is detrimental to transcription accuracy. We present evidence that rhythmic irregularities in amateur vocal percussion are the major contributing factor to this because due to them our straightforward approach to beat alignment based on global tempo estimation produces semantically incorrect alignments. We find that errors in onset detection are also likely to sabotage the performance of the language model, and therefore conclude that our proposed approach of language model integration is too sensitive to errors produced by other parts of the analysis pipeline to be practical.

To the best of our knowledge, this is the first study on vocal percussion transcription which features deep learning techniques and is also the first study on the application of language modelling to VPT. It is also the first study on VPT which utilizes AVP dataset for training and evaluation. Moreover, since, to the best of our knowledge, all the existing systems for VPT have been implemented as standalone applications or plugins for digital audio workstations (such as Ableton Live), this project is possibly the first project on vocal percussion transcription which is implemented entirely in Python using popular open-source libraries. We make our code open source[1] for reproducibility and to help future research in the field to be built upon our efforts.

5.2 Future work

The primary direction of future work is fixing the method of language model adaptation so that it actually improves the transcription quality. Most importantly, naive rhythm quantization method based only on a global tempo estimate should be replaced with a more robust approach, which accounts for tempo variations and onset timing errors, such as metric HMM [48, 47]. Furthermore, to mitigate the negative impact of errors in onset detection, a probabilistic approach to onset detection can be considered (such as [6], so that onset detection, quantization and classification steps can be combined in a joint probabilistic model. And, certainly, different types of language models trained on different MIDI datasets should be compared.

Besides, further modifications to the architecture and training process of CAEs might be considered to improve the feature extraction quality further. Variational autoencoders, especially with non-Gaussian priors [50, 75], might be a promising research direction.

[1] `https://github.com/flyingleafe/vxs-vpt`

Bibliography

[1] 200 drum machines. http://www.hexawe.net/mess/200.Drum.Machines/. Accessed: 2020-01-06.

[2] beatboxset1. https://archive.org/details/beatboxset1. Accessed: 2020-01-06.

[3] Polyphone soundfont editor. https://www.polyphone-soundfonts.com/. Accessed: August of 2020.

[4] Vocal grammatics. http://www.vocalgrammatics.fr/. Accessed: 2020-08-20.

[5] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, pages 265–283, 2016.

[6] Samer A Abdallah and Mark D Plumbley. Probability as metadata: event detection in music using ica as a conditional density model. In *Proc. 4th Int. Symp. Independent Component Analysis and Signal Separation (ICA2003)*, pages 233–238. Citeseer, 2003.

[7] E Aylon and N Wack. Beat detection using plp. *Music Inf. Retrieval Evaluation eXchange (MIREX)*, 2010.

[8] Douglas Bates, Deepayan Sarkar, Maintainer Douglas Bates, and L Matrix. The lme4 package. *R package version*, 2(1):74, 2007.

[9] Juan Pablo Bello, Laurent Daudet, Samer Abdallah, Chris Duxbury, Mike Davies, and Mark B Sandler. A tutorial on onset detection in music signals. *IEEE Transactions on speech and audio processing*, 13(5):1035–1047, 2005.

[10] Juan Pablo Bello, Chris Duxbury, Mike Davies, and Mark Sandler. On the use of phase and energy for musical onset detection in the complex domain. *IEEE Signal Processing Letters*, 11(6):553–556, 2004.

[11] Emmanouil Benetos, Simon Dixon, Zhiyao Duan, and Sebastian Ewert. Automatic music transcription: An overview. *IEEE Signal Processing Magazine*, 36(1):20–30, 2018.

[12] Sebastian Böck, Andreas Arzt, Florian Krebs, and Markus Schedl. Online real-time onset detection with recurrent neural networks. In *Proceedings of the 15th International Conference on Digital Audio Effects (DAFx-12), York, UK*, 2012.

[13] Sebastian Böck, Florian Krebs, and Gerhard Widmer. Accurate tempo estimation based on recurrent neural networks and resonating comb filters. In *ISMIR*, pages 625–631, 2015.

[14] Sebastian Böck and Gerhard Widmer. Maximum filter vibrato suppression for onset detection. In *Proc. of the 16th Int. Conf. on Digital Audio Effects (DAFx). Maynooth, Ireland (Sept 2013)*, volume 7, 2013.

[15] Nicolas Boulanger-Lewandowski, Yoshua Bengio, and Pascal Vincent. Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. *arXiv preprint arXiv:1206.6392*, 2012.

[16] Paul Brossier. Aubio, a library for audio labelling. `https://aubio.org/`. Accessed: April of 2020.

[17] Mark Cartwright and Bryan Pardo. Vocalsketch: Vocally imitating audio concepts. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pages 43–46, 2015.

[18] Matthew EP Davies, Paul M Brossier, and Mark D Plumbley. Beat tracking towards automatic musical accompaniment. In *Audio Engineering Society Convention 118*. Audio Engineering Society, 2005.

[19] Alejandro Delgado, SKoT McDonald, Ning Xu, and Mark Sandler. A new dataset for amateur vocal percussion analysis. In *Proceedings of the 14th International Audio Mostly Conference: A Journey in Sound on ZZZ*, pages 17–23, 2019.

[20] Christian Dittmar and Daniel Gärtner. Real-time transcription and separation of drum recordings based on nmf decomposition. In *DAFx*, pages 187–194, 2014.

[21] Christian Dittmar and Christian Uhle. Further steps towards drum transcription of polyphonic music. In *Audio Engineering Society Convention 116*. Audio Engineering Society, 2004.

[22] Simon Dixon. Automatic extraction of tempo and beat from expressive performances. *Journal of New Music Research*, 30(1):39–58, 2001.

[23] Simon Dixon. Evaluation of the audio beat tracking system beatroot. *Journal of New Music Research*, 36(1):39–50, 2007.

[24] Stephen Downie and Yun Hao. Mirex 2018 evaluation results, 2018.

[25] Chris Duxbury, Mark Sandler, and Mike Davies. A hybrid approach to musical note onset detection. In *Proc. Digital Audio Effects Conf.(DAFX,'02)*, pages 33–38, 2002.

[26] Georgi Dzhambazov. Towards a drum transcription system aware of bar position. In *Audio Engineering Society Conference: 53rd International Conference: Semantic Audio*. Audio Engineering Society, 2014.

[27] Daniel PW Ellis. Beat tracking by dynamic programming. *Journal of New Music Research*, 36(1):51–60, 2007.

[28] Solène Evain, Benjamin Lecouteux, Didier Schwab, Adrien Contesse, Antoine Pinchaud, and Nathalie Bernardoni. Human beatbox sound recognition using an automatic speech recognition toolkit. 2020.

[29] Olivier Gillet and Gaël Richard. Enst-drums: an extensive audio-visual database for drum signals processing. In *ISMIR*, pages 156–159, 2006.

[30] Olivier Gillet and Gaël Richard. Transcription and separation of drum signals from polyphonic music. *IEEE Transactions on Audio, Speech, and Language Processing*, 16(3):529–540, 2008.

[31] Jon Gillick, Adam Roberts, Jesse Engel, Douglas Eck, and David Bamman. Learning to groove with inverse sequence transformations. In *International Conference on Machine Learning (ICML)*, 2019.

[32] Curtis Hawthorne, Erich Elsen, Jialin Song, Adam Roberts, Ian Simon, Colin Raffel, Jesse Engel, Sageev Oore, and Douglas Eck. Onsets and frames: Dual-objective piano transcription. *arXiv preprint arXiv:1710.11153*, 2017.

[33] Amaury Hazan. Towards automatic transcription of expressive oral percussive performances. In *Proceedings of the 10th international conference on Intelligent user interfaces*, pages 296–298. ACM, 2005.

[34] Kyle Hipke, Michael Toomim, Rebecca Fiebrink, and James Fogarty. Beatbox: end-user interactive definition and training of recognizers for percussive vocalizations. In *Proceedings of the 2014 International Working Conference on Advanced Visual Interfaces*, pages 121–124, 2014.

[35] Ajay Kapur, Manj Benning, and George Tzanetakis. Query-by-beat-boxing: Music retrieval for the dj. In *Proceedings of the International Conference on Music Information Retrieval*, pages 170–177, 2004.

[36] Jong Wook Kim and Juan Pablo Bello. Adversarial learning for improved onsets and frames music transcription. *arXiv preprint arXiv:1906.08512*, 2019.

[37] Nikolaos Kouiroukidis and Georgios Evangelidis. The effects of dimensionality curse in high dimensional knn search. In *2011 15th Panhellenic Conference on Informatics*, pages 41–45. IEEE, 2011.

[38] Ke Li, Hainan Xu, Yiming Wang, Daniel Povey, and Sanjeev Khudanpur. Recurrent neural network language model adaptation for conversational speech recognition. In *Interspeech*, pages 3373–3377, 2018.

[39] Robert C Maher and James W Beauchamp. Fundamental frequency estimation of musical signals using a two-way mismatch procedure. *The Journal of the Acoustical Society of America*, 95(4):2254–2263, 1994.

[40] Paul Masri. *Computer modelling of sound for transformation and synthesis of musical signals.* PhD thesis, University of Bristol, 1996.

[41] Brian McFee, Colin Raffel, Dawen Liang, Daniel PW Ellis, Matt McVicar, Eric Battenberg, and Oriol Nieto. librosa: Audio and music signal analysis in python. In *Proceedings of the 14th python in science conference*, volume 8, pages 18–25, 2015.

[42] Cory McKay, Rebecca Fiebrink, Daniel McEnnis, Beinan Li, and Ichiro Fujinaga. Ace: A framework for optimizing music classification. In *ISMIR*, pages 42–49, 2005.

[43] Adib Mehrabi, Keunwoo Choi, Simon Dixon, and Mark Sandler. Similarity measures for vocal-based drum sample retrieval using deep convolutional auto-encoders. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 356–360. IEEE, 2018.

[44] Adib Mehrabi, Simon Dixon, and Mark B Sandler. Vocal imitation of synthesised sounds varying in pitch, loudness and spectral centroid. *The Journal of the Acoustical Society of America*, 141(2):783–796, 2017.

[45] Bertrand David Miguel Alonso and Gaël Richard. Tempo and beat estimation of musical signals. In *Proceedings of the International Conference on Music Information Retrieval (ISMIR), Barcelona, Spain*, 2004.

[46] Marius Miron, Matthew EP Davies, and Fabien Gouyon. An open-source drum transcription system for pure data and max msp. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 221–225. IEEE, 2013.

[47] Eita Nakamura, Emmanouil Benetos, Kazuyoshi Yoshii, and Simon Dixon. Towards complete polyphonic music transcription: Integrating multi-pitch detection and rhythm quantization. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 101–105. IEEE, 2018.

[48] Eita Nakamura, Kazuyoshi Yoshii, and Shigeki Sagayama. Rhythm transcription of polyphonic piano music based on merged-output hmm for multiple voices. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 25(4):794–806, 2017.

[49] Tomoyasu Nakano, Masataka Goto, Jun Ogata, and Yuzuru Hiraga. Voice drummer: A music notation interface of drum sounds using voice percussion input. In *Proceedings of the 18th Annual ACM Symposium on User Interface Software and Technology (UIST2005)*, pages 49–50, 2005.

[50] Eric Nalisnick, Lars Hertel, and Padhraic Smyth. Approximate inference for deep latent gaussian mixtures. In *NIPS Workshop on Bayesian Deep Learning*, volume 2, 2016.

[51] Jan Newmarch. Fluidsynth. In *Linux Sound Programming*, pages 351–353. Springer, 2017.

[52] Jouni Paulus. Signal processing methods for drum transcription and music structure analysis. 2010.

[53] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011.

[54] Benjamin Picart, Sandrine Brognaux, and Stéphane Dupont. Analysis and automatic recognition of human beatbox sounds: A comparative study. In *2015 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 4255–4259. IEEE, 2015.

[55] Daniel Povey, Arnab Ghoshal, Gilles Boulianne, Lukas Burget, Ondrej Glembek, Nagendra Goel, Mirko Hannemann, Petr Motlicek, Yanmin Qian, Petr Schwarz, et al. The kaldi speech recognition toolkit. In *IEEE 2011 workshop on automatic speech recognition and understanding*, number CONF. IEEE Signal Processing Society, 2011.

[56] Colin Raffel. *Learning-based methods for comparing sequences, with applications to audio-to-midi alignment and matching*. PhD thesis, Columbia University, 2016.

[57] Colin Raffel, Brian McFee, Eric J Humphrey, Justin Salamon, Oriol Nieto, Dawen Liang, Daniel PW Ellis, and C Colin Raffel. mir_eval: A transparent implementation of common mir metrics. In *In Proceedings of the 15th International Society for Music Information Retrieval Conference, ISMIR*. Citeseer, 2014.

[58] António Filipe Santana Ramires. Automatic transcription of vocalized percussion. 2017.

[59] Sebastian Raschka. Mlxtend: providing machine learning and data science utilities and extensions to python's scientific computing stack. *Journal of open source software*, 3(24):638, 2018.

[60] Dave Rossum and E Joint. The soundfont® 2.0 file format. *Joint E-Mu/Creative Tech Center white paper*, 1995.

[61] Vegard Sandvold, Fabien Gouyon, and Perfecto Herrera. Percussion classification in polyphonic audio recordings using localized sound models. In *Proc. International Conference on Music Information Retrieval*, pages 537–540. Citeseer, 2004.

[62] W Andrew Schloss. On the automatic transcription of percussive music–from acoustic signal to high-level analysis. 1986.

[63] Jan Schlüter and Sebastian Böck. Improved musical onset detection with convolutional neural networks. In *2014 ieee international conference on acoustics, speech and signal processing (icassp)*, pages 6979–6983. IEEE, 2014.

[64] Hendrik Schreiber and Meinard Müller. A single-step approach to musical tempo estimation using a convolutional neural network. In *Ismir*, pages 98–105, 2018.

[65] Changhao Shan, Chao Weng, Guangsen Wang, Dan Su, Min Luo, Dong Yu, and Lei Xie. Component fusion: Learning replaceable language model component for end-to-end speech recognition system. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5361–5635. IEEE, 2019.

[66] Siddharth Sigtia, Emmanouil Benetos, Srikanth Cherla, Tillman Weyde, A Garcez, and Simon Dixon. Rnn-based music language models for improving automatic music transcription. 2014.

[67] Siddharth Sigtia, Emmanouil Benetos, and Simon Dixon. An end-to-end neural network for polyphonic piano music transcription. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 24(5):927–939, 2016.

[68] Umut Şimşekli, Antti Jylhä, Cumhur Erkut, and A Taylan Cemgil. Real-time recognition of percussive sounds by a model-based method. *EURASIP Journal on Advances in Signal Processing*, 2011:1–14, 2011.

[69] Elliot Sinyor, Cory Mckay Rebecca, Daniel Mcennis, and Ichiro Fujinaga. Beatbox classification using ace. In *Proceedings of the International Conference on Music Information Retrieval*. Citeseer, 2005.

[70] Dan Stowell and Mark D Plumbley. Delayed decision-making in real-time beatbox percussion classification. *Journal of New Music Research*, 39(3):203–213, 2010.

[71] Google Brain Team. Magenta. `https://github.com/magenta/magenta`. Accessed: November of 2018.

[72] Google Brain Team. Magenta drumsrnn. `https://github.com/tensorflow/magenta/tree/master/magenta/models/drums_rnn`. Accessed: November of 2018.

[73] Ernst Terhardt. Calculating virtual pitch. *Hearing research*, 1(2):155–182, 1979.

[74] Shubham Toshniwal, Anjuli Kannan, Chung-Cheng Chiu, Yonghui Wu, Tara N Sainath, and Karen Livescu. A comparison of techniques for language model integration in encoder-decoder speech recognition. In *2018 IEEE spoken language technology workshop (SLT)*, pages 369–375. IEEE, 2018.

[75] Dustin Tran, Rajesh Ranganath, and David M Blei. The variational gaussian process. *arXiv preprint arXiv:1511.06499*, 2015.

[76] Benigno Uria, Marc-Alexandre Côté, Karol Gregor, Iain Murray, and Hugo Larochelle. Neural autoregressive distribution estimation. *The Journal of Machine Learning Research*, 17(1):7184–7220, 2016.

[77] Richard Vogl, Matthias Dorfer, Gerhard Widmer, and Peter Knees. Drum transcription via joint beat and drum modeling using convolutional recurrent neural networks. In *ISMIR*, pages 150–157, 2017.

[78] Richard Vogl, Gerhard Widmer, and Peter Knees. Towards multi-instrument drum transcription. *arXiv preprint arXiv:1806.06676*, 2018.

[79] Qi Wang, Ruohua Zhou, and Yonghong Yan. Polyphonic piano transcription with a note-based music language model. *Applied Sciences*, 8(3):470, 2018.

[80] Chih-Wei Wu, Christian Dittmar, Carl Southall, Richard Vogl, Gerhard Widmer, Jason Hockman, Meinard Müller, and Alexander Lerch. A review of automatic drum transcription. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 26(9):1457–1483, 2018.

[81] Adrien Ycart, Andrew McLeod, Emmanouil Benetos, Kazuyoshi Yoshii, et al. Blending acoustic and language model predictions for automatic music transcription. 2019.

[82] Kazuyoshi Yoshii, Masataka Goto, and Hiroshi G Okuno. Drum sound recognition for polyphonic audio signals by adaptation and matching of spectrogram templates with harmonic structure suppression. *IEEE Transactions on Audio, Speech, and Language Processing*, 15(1):333–345, 2006.

[83] Steve Young, Gunnar Evermann, Mark Gales, Thomas Hain, Dan Kershaw, Xunying Liu, Gareth Moore, Julian Odell, Dave Ollason, Dan Povey, et al. The htk book. *Cambridge university engineering department*, 3(175):12, 2002.

[84] Aymeric Zils, François Pachet, Olivier Delerue, and Fabien Gouyon. Automatic extraction of drum tracks from polyphonic music signals. In *Second International Conference on Web Delivering of Music, 2002. WEDELMUSIC 2002. Proceedings.*, pages 179–183. IEEE, 2002.

Appendix A

Replication of Mehrabi's results

In order to make sure that no significant mistakes were done during the reimplementation of CAEs proposed by Mehrabi et al., we partially replicate the experiments described in their paper. Those experiments aimed to analyze how good the distances between feature representations of real drum sounds and their imitations predict similarity ratings between pairs of sounds collected from human listeners. We use the same drum sounds, vocal imitations and rating scores as in the original paper, since the authors made their dataset public. The dataset contains 30 real drum sounds of six classes (kicks, snares, toms, hats and cymbals) and vocal imitations of those sounds produced by 14 different imitators (420 imitations in total).

The similarity ratings between imitations and real sounds were collected from 63 listeners using the following procedure. Each listener was presented with 30 test pages, each page containing a reference vocal imitation and six within-class real drum sounds, one of which is the imitated sound. The listeners then assigned ratings from 0 to 100 to each of the six real sounds based on their perceived similarity to the reference vocal imitation. Among 30 test pages, 28 were unique and 2 were random duplicates. The duplicate pages were used in order to filter out unreliable listeners. The listener was considered reliable if for at least one pair of duplicate pages their responses shown large positive Spearman rank correlation ($\rho \geq 0.5$). We performed this procedure for filtering out unreliable listeners and determined that 51 out of 63 listeners were reliable, same as reported by Mehrabi et al. However, we notice that we obtained a different number of non-duplicate ratings from those listeners than was reported in the original paper: 9087 vs expected 9126.

The deep feature representations obtained by 11 different types of CAEs as described in Section 3.2.3.2 were then calculated for all sounds in the dataset. Addi-

tionally, we calculated one of the baseline feature representations used in the original paper. It is obtained by calculating first 13 MFCCs (excluding MFCC 0) with first and second derivatives for the whole sound using window size of 4096 and hop size of 512. Mean and variance of those features are then calculated over the time axis, yielding 78 features in total. Moreover, in order to gain insight on how good our baseline feature set of 20 MFCCs over the first 4096 samples of the sound is in terms of perceptional similarity, we also calculated those.

Type	Size	Original		Replicated	
		AIC	Acc. (%)	AIC	Acc. (%)
square-1	2048	**1820**	**73.3**	63	90.0
square-2	512	1925	66.7	14	90.0
square-3	128	1958	66.7	**-32**	**90.0**
tall-1	1024	**1609**	**73.3**	**-110**	**90.0**
tall-2	512	1647	70.0	-6	86.7
tall-3	256	2361	63.3	118	83.3
tall-4	128	2523	56.7	23	83.3
wide-1	1024	1921	66.7	**-50**	**90.0**
wide-2	512	1866	73.3	26	90.0
wide-3	256	1395	83.3	70	80.0
wide-4	128	**1298**	**83.3**	18	80.0
MFCC (orig.)	78	2703	53.3	1544	30.0
MFCC (short)	20	—	—	**1259**	**66.7**

Table A.1: Comparison of original LMER fitting results to replicated ones. Lower AIC is better, higher accuracy is better.

For each feature set, Euclidean distances between feature representation of real and imitated sounds were then calculated for each real/imitated pair of within-class sounds, yielding 2520 distances. Those distances were normalized to the interval $(0,1)$ and then used as features for fitting a *linear mixed-effect regression* (LMER) model for predicting similarity ratings reported by listeners (which we also normalized to $(0,1)$). The model is given by a formula:

$$y_{ijk} = \nu_j + \beta_{1j} x_{ij} + \gamma_k + \varepsilon_{ijk}, \tag{A.1}$$

where y_{ijk} is the similarity rating between imitation i and imitated sound j provided by listener k, x_{ij} is the distance between i and j, ν_j and β_{1j} are fixed intercept and slope parameters different for each imitated sound j and γ_k is a random intercept for listener k. This model can be interpreted as 30 different linear regression models for each sound which also share a random intercept shift which depends on a listener. We fitted those models using *lme4* R package [8]. The quality of fitted models was evaluated using two metrics: Akaike's information criterion (AIC) and the percentage of the slopes β_{1j} which are reliably negative, i.e. indicate that the similarity rating is decreasing with increasing distance. The latter was denoted as *accuracy* and computed via obtaining Wald's 95% confidence intervals on slopes and calculating the portion of upper confidence bounds which is less than zero.

The results of the replication are summarized in Table A.1. We notice that AICs obtained by us are significantly lower than the original ones in general. This might be due to imprecise interpretation of LMER model structure by us, or due to different rating scaling (Mehrabi et al never mentioned that they scaled the ratings to $[0, 1]$, but we noticed that having ratings in range from 0 to 100 leads to AIC values larger than 80000; however, scaling had no effect on the accuracy). In any case, however, it shows that the direct one-to-one comparison between the original and replicated results may be meaningless. However, we can still compare the results based on how scores of individual features relate to each other. For example, we notice that the discrepancy between performances of the baseline and CAE features is even stronger in our results than in the original report: the accuracy we obtain with original MFCC-based features is significantly lower than the original one, but the accuracies obtained by CAE features are consistently higher than original. We also do not see the clear pattern of performance increasing with the decreasing size of *wide* autoencoders: in our case, in general, accuracy is decreasing with the decreasing of feature representation size, and the behavior of AIC metric does not exhibit any clear pattern. This prevents us from supporting the conclusion that for *wide* autoencoders the quality of extracted feature representations grows with the decrease of their size.

We also notice that our simple baseline feature set of 20 MFCCs performs better than the original MFCC-based baseline. Nevertheless, it still performs worse than any CAE feature set in our experiments. Therefore, even though our results differ from the original ones and do not support some of the original conclusions, the general (and the most important) conclusion — namely, the superiority of CAE features to MFCC features in terms of perceptual similarity preservation — definitely holds. Since this

conclusion is our primary motivation for using CAE features for vocal percussion transcription, we can proceed with the application of those features to the actual transcription task.