DEEP UNSUPERVISED DRUM TRANSCRIPTION

First Author
Affiliation1
author1@ismir.edu

Second Author
Retain these fake authors in submission to preserve the formatting

Third Author
Affiliation3
author3@ismir.edu

ABSTRACT

We introduce DrummerNet, a drum transcription system that is trained in an unsupervised manner. DrummerNet does not require any ground-truth transcription, and with the data-scalability of deep neural networks, it learns from a large unlabelled dataset. In DrummerNet, the target drum signal is first passed to a (trainable) transcriber, and a (fixed) synthesizer reconstructs the input signal from the transcription estimate. By training the system to minimize the distance between the input and the output audio signals, the transcriber learns to transcribe without ground truth transcription. In the experiments, DrummerNet performs favorably compared to many recent drum transcription systems, both supervised and unsupervised.

1 Introduction

Transcription is a music information retrieval task where the goal is to estimate the score $y$ when input audio $x$ is given. The majority of the recent transcription systems is based on supervised learning, where the transcriber is an analysis system $\hat{y} = F_a(x)$ that is trained with annotated pairs, $\{(x_m, y_m)\}_{m=1}^M$ to minimize the distance between $y$ and $\hat{y}$ [6, 7, 27, 31, 33, 34, 37, 38].

The trend is similar in drum transcription on which we focus in this paper. Among supervised learning approaches, a model can be based on frame-based feature extraction and classification [15], non-negative matrix factorization (NMF) for matching temporal activity and spectral templates with support vector machine classifier [10], and hidden-Markov model as a prediction model [25]. More attention has been given recently deep learning based models such as convolutional neural networks (CNNs, [13,34]) and recurrent neural networks (RNNs, [33,37,38]).

The progress of machine learning has improved transcription systems. However, the lack of a large-scale annotated dataset is one of the most frequently mentioned obstacles to further improvement, for which there have been proposals to use unlabelled data [42, 43]. However, they still rely on supervised learning combined with teacher-student learning [16] and convolutional auto-encoder. Using synthetic data is an way to address the data issue [7,39]. In parallel with those approaches, an annotation-free, therefore more scalable and generalizable alternative would be unsupervised learning.

Unsurprisingly, one of human’s music learning procedures, self-taught by trial-and-error, is close to unsupervised learning. For example, musicians learn to transcribe by (a) listening, (b) playing an instrument, (c) identifying the difference, and (d) making an adjustment. Can this be done without any supervision? Yes, if a person can notice the pitch difference (e.g., the pitch should be higher or lower). Applying this analogy, developing a transcription system based on unsupervised learning would be feasible if the system can test the estimation, measure the error, and corrects itself accordingly.

To realize such an unsupervised transcription system, we need a synthesis system, $\hat{x} = F_s(\hat{y})$, making the overall system $\hat{x} = F_s(F_a(x))$. During its training, the system is given $\{x\}_{m=1}^M$ and trained to minimize the distance between $x$ and $\hat{x}$ There have been a few systems relying on unsupervised learning as explained. In MIR, the system in [1] utilized sparse coding to learn a dictionary of time-frequency templates of piano and harpsicord, assuming a (matrix-)multiplication model with additive noise, $F_s(y) = Ay + e$. Yoshii et al. proposed to use sparse coding in a jointly-learned chord recognition and transcription system [44]. Berg et al. designed a probabilistic graphical model that parameterizes the spectral and temporal envelopes, note events, and note activations, in order to transcribe piano by inferring the parameters [2]. For drum transcription, many systems have used NMF to decompose a drum track spectrum into spectral templates and temporal activations (or transcription) of them [26,41]. To address the limit of the fixed spectrum template of NMF, variants of NMF were proposed [19,20,29]. Lastly, a similar system can be found in computer vision, where the parameters of input images are estimated by reconstruction using image renderers [18].

In this paper, we introduce DrummerNet, a deep neural network based drum transcription system that is trained by unsupervised learning. With a more end-to-end approach, DrummerNet is distinguished from previous research, [1,2,44], which have strong priors on the target sounds. In §2, we present the system design principle behind DrummerNet, followed by the details of DrummerNet in §3. In §4, the evaluation results are discussed along with ablation study We present our conclusion, the problems of our system and accordingly, future directions towards fully unsupervised learning in transcription/MIR in §5.
3.1 Analysis module $F_a$

The analysis module $F_a$, as illustrated in the top half of Figure 1, takes the audio signal $x$ as an input and processes it through a series of U-net variant [30], recurrent layers, and gated Sparsemax activation [21].

**U-net** The U-net consists of 1D convolutional layers, max-pooling layers, and concatenations between the encoder and the decoder. In the encoding part, there are a convolutional layer $(128, 3, 1)$, and 10 convolutional layers $(50, 3, 1)$ interleaved with max-pooling of size 2. As a result, the encoder outputs $z \in \mathbb{R}^{N/1024}$ which has a receptive field size of 3,072 time steps.

In the decoder, there are only 6 convolutional layers $(50, 3, 1)$, interleaved with i) a concatenation with the feature map at the same depth in the encoder and ii) $\times 2$ bi-linear interpolation. We call the output of decoder $r \in \mathbb{R}^{N/16}$, the representation based on which the transcription is estimated. The asymmetry between the encoder and the decoder makes the length $r$ to be shorter by the factor of $4^2 = 16$ compared to the input $x$. Assuming the input audio is sampled at 16 kHz, $r$ has a sampling rate of 1,000 Hz.

**Recurrent layers** We use three recurrent layers (GRUs [8]) along {time-axis, bi-directional, 100-channel}, {time-axis, uni-directional, 50-channel}, and {channel-axis, uni-directional, $K$-channel} respectively. These three recurrent layers allow us to i) being bi-directional so that the onset at $n$ can be determined by the vicinity of $n$, both the past and the future, ii) enforcing temporal dependency, and iii) enforcing component-wise dependency. The width (or the hidden vector size) of the third recurrent layer is equal to $K$, the number of drum components in the synthesizer, in order to map each channel to each drum component respectively.

**Sparsemax** In an ideal case of transcription, there would be a local sparsity along both time- and channel-axes – because the drum events would not repeat with a rate of 1,000 Hz (which is faster than 16-beat on 240 BPM) nor all the $K$ drum components would be activated simultaneously. Although sparsity is one of the properties that can be achieved by the auto-regressive nature of the recurrent layers, we add Sparsemax [21] activation to explicitly encourage it. Sparsemax has two important properties – i) its output always sum to 1 (same as Softmax), ii) the output is highly likely to be sparse with actual zeros (unlike Softmax). In DrummerNet, two Sparsemax layers are applied in parallel, one along channel-axis (=instrument-axis) and the other time-axis within a non-overlapping window size of 64. This design choice is based on an assumption that there is mostly only a few onsets among notes (channel-axis sparsity) and within 64 samples at $\hat{y}$, or 64 ms (temporal sparsity). The outputs from these two Sparsemax layers are then multiplied element-wisely.

---

1 The implementation of DrummerNet is attached for review and will be publicly available.

2 This is the sampling rate of input audio in our experiment.

---

### Table 1: Symbols used in this paper

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n, N$</td>
<td>The temporal index/length of audio input</td>
<td></td>
</tr>
<tr>
<td>$k, K$</td>
<td>The index/total number of drum components</td>
<td>$K = 11$</td>
</tr>
<tr>
<td>$x, y$</td>
<td>Mixture and transcription</td>
<td>$\in \mathbb{R}^N$</td>
</tr>
<tr>
<td>$\hat{x}, \hat{y}$</td>
<td>Estimations of mixture/transcription</td>
<td>$\in \mathbb{R}^N$</td>
</tr>
</tbody>
</table>

### Figure 1: Block diagrams of DrummerNet structure.

Trainable modules are illustrated as block boxes and fixed modules are as rounded grey boxes.

2 System Design Principles

Training the proposed DrummerNet is similar to the previous unsupervised learning approaches for music [1, 2, 44], each of which trains a system to output $\hat{x}$ that reconstructs the original signal $x$. The difference between $\hat{x}$ and $x$ works as a proxy of that between $\hat{y}$ to $y$.

There are three conditions under which unsupervised learning of a transcription can be done successfully. First, the output of the analysis module $F_a$ must be in a form of transcription – a set of discrete events representing the timings and intensities of notes. Second, the synthesis module $F_s$ must synthesize the audio signal given transcription input $\hat{y}$. Third, all the components in the network must be differentiable as we rely on backpropagation to train it.

3 DrummerNet

In this section, we introduce the proposed system structure. We specify the number of channels, kernel size, and stride as (channel, kernel, stride). All the convolutional and recurrent layers use an exponential linear unit as an activation function [9].

---

$\hat{x}$: audio (output)

$\hat{y}$: estimated transcription

$\hat{\gamma}_g$: sparsemax

$\hat{\gamma}_r$: Sparsemax

$\sum$: sparsemax

$x$: audio (input)

$F_a$: analysis module

$F_s$: synthesis module

U-net: The U-net consists of 1D convolutional layers, max-pooling layers, and concatenations between the encoder and the decoder. In the encoding part, there are a convolutional layer $(128, 3, 1)$, and 10 convolutional layers $(50, 3, 1)$ interleaved with max-pooling of size 2. As a result, the encoder outputs $z \in \mathbb{R}^{N/1024}$ which has a receptive field size of 3,072 time steps.

In the decoder, there are only 6 convolutional layers $(50, 3, 1)$, interleaved with i) a concatenation with the feature map at the same depth in the encoder and ii) $\times 2$ bi-linear interpolation. We call the output of decoder $r \in \mathbb{R}^{N/16}$, the representation based on which the transcription is estimated. The asymmetry between the encoder and the decoder makes the length $r$ to be shorter by the factor of $4^2 = 16$ compared to the input $x$. Assuming the input audio is sampled at 16 kHz, $r$ has a sampling rate of 1,000 Hz.

**Recurrent layers** We use three recurrent layers (GRUs [8]) along {time-axis, bi-directional, 100-channel}, {time-axis, uni-directional, 50-channel}, and {channel-axis, uni-directional, $K$-channel} respectively. These three recurrent layers allow us to i) being bi-directional so that the onset at $n$ can be determined by the vicinity of $n$, both the past and the future, ii) enforcing temporal dependency, and iii) enforcing component-wise dependency. The width (or the hidden vector size) of the third recurrent layer is equal to $K$, the number of drum components in the synthesizer, in order to map each channel to each drum component respectively.

**Sparsemax** In an ideal case of transcription, there would be a local sparsity along both time- and channel-axes – because the drum events would not repeat with a rate of 1,000 Hz (which is faster than 16-beat on 240 BPM) nor all the $K$ drum components would be activated simultaneously. Although sparsity is one of the properties that can be achieved by the auto-regressive nature of the recurrent layers, we add Sparsemax [21] activation to explicitly encourage it. Sparsemax has two important properties – i) its output always sum to 1 (same as Softmax), ii) the output is highly likely to be sparse with actual zeros (unlike Softmax). In DrummerNet, two Sparsemax layers are applied in parallel, one along channel-axis (=instrument-axis) and the other time-axis within a non-overlapping window size of 64. This design choice is based on an assumption that there is mostly only a few onsets among notes (channel-axis sparsity) and within 64 samples at $\hat{y}$, or 64 ms (temporal sparsity). The outputs from these two Sparsemax layers are then multiplied element-wisely.
Onset Enhancement

Median-Filtering

Onset Enhancement

CQTs

x → Median-Filtering

Onset Enhancement

CQTs → Mean Absolute Error

\[ x \xrightarrow{\text{Median-Filtering}} \text{Onset Enhancement} \xrightarrow{\text{CQTs}} \text{Mean Absolute Error} \]

Figure 2: The block diagrams of loss calculation

Upsampler Finally, the low temporal resolution of the Sparsemax output is addressed by zero-insertion upsampling by the factor of 16. By this, we modify the temporal quantization rate of events, unlike upsampling in digital signal processing.

3.2 Synthesis module \( F_s \)

The synthesis module \( F_s \) consists of \( K \) parallel 1D convolutional layers and a channel-wise summing operator. The kernel of each layer is not trained but fixed to the known waveform of each drum component to convert a transcription of a component \( \hat{y}_k \) into a track \( \hat{x}_k \). The tracks are summed to generate the final output \( \hat{x} = \sum_{k=1}^{K} \hat{x}_k \), the synthesized audio signal.

In the implementation, we use \( K = 11 \), using ‘Subclass’ in Table 2, following [36]. Ones with asterisks are excluded due to their scarcities in the source of isolated drum recordings we use, which are, 12 virtual drum instruments provided with Logic Pro X. The multiple drum kits we use include kits for rock, pop, funk, and soul\(^3\) to avoid the network from overfitting to a specific drum kit. During training, a drum kit is randomly chosen every batch.

3.3 Learning

Because we cannot compute the loss at the transcription directly in unsupervised learning, we should carefully design a loss function at the audio level, \( L_x(x, \hat{x}) \), so that minimizing it would lead to minimizing \( L_y(y, \hat{y}) \), the transcription loss. To do so, \( L_x \) should be able to differentiate KD/SD/HH (kick drum, snare drum, and hi-hat respectively) while being invariant to varying drum kits. Perceptually, there are clear differences among KD, SD, and HH – KD is in the low-frequency band and impulsive while SD is in the mid-frequency band, relatively tonal, impulsive but with a relatively longer envelope. HH is more complicated to describe due to its variation from the playing technique: Closed- and pedalled-HH’s are in the high-frequency band, impulsive, with a relatively low energy, and open-HH is similar but with a longer, noisy envelope and louder.

We thus define and use onset spectrum similarity which is designed to represent the similarity based on the onset part of sounds in the spectrum domain. As illustrated in 2, it is measured by i) applying median-filtering based drum extraction [12] which enhances onsets (with a FFT size of 1024 and median filter length of 31 on both axes), ii) converting to multi-resolution CQTs (constant-Q transform) for both \( x \) and \( \hat{x} \), and then iii) calculating the mean absolute difference between them.

Among many spectral magnitude representations, we use CQT in the decibel scale since the octave-band frequency axis is known to match well to human auditory perception [23]. We follow the implementation of ‘pseudo-CQT’\(^4\) which multiplies linear-to-octave filterbanks to an STFT. As a result, the CQT covers nearly 8 octave bands

---

\(^3\) Brooklyn, Heavy, Liverpool, Neo Soul, Detroit Garage, Motown Revisited, Portland, Sunset, Speakeasy, SoCal, Smash, and Slow Jam. All with velocity=98.

\(^4\) http://librosa.github.io/librosa/
from 32.07 Hz (≈1) to 8 kHz, the Nyquist frequency of our experiment, with a 12-band/octave resolution.

Figure 3 shows the effect of onset enhancement. It preserves the characteristics of the drum components in the transient part, removing after-onset components. This process makes \( L_x \) and \( L_y \) more similar, as the non-transient parts vary more among drum kits due to their random and noisy nature. In a preliminary experiment, for example, the network tried to reconstruct all the non-transient components of SD using tom-toms and HHs, resulting in non-sparse and severe false-positive detection of onsets.

4 Experiments and Analysis

4.1 Setup

For the training of DrummerNet, we use an in-house dataset of drum stems that are crawled from many websites. This dataset consists of 3,940 unique tracks, 225 seconds on average, 249 hours in total. Since the dataset was crawled from various web sites, some of the details such as the distribution of drum components are hard to identify. The tracks are however mostly well-known western rock/pop music. As an alternative to this in-house dataset, there are other drum datasets such as one in [7] (3,758 drum sample recordings (×8 second = over 8 hours) and 60,000 synthesized drum loops (×8 second = over 133 hours)) and [39] (4,197 drum tracks, 259 hours). We use in-house one because it’s not synthesized, hence provides more diversity.

Each audio file is resampled to 16 kHz and downmixed to mono. The training batch size is 16 and for each audio file, we randomly select a 2-second segment. On average, there are 112.5 segments in a track, therefore training with 443,250 (=3,940 × 112.5) items would be approximately one epoch. With a Nvidia Tesla P100 and a batch size of 32, it took about 9 hours to train over single epoch. We implement the proposed DrummerNet using Pytorch 1.0 [24]. We use Librosa 0.6.3 [22] and Madmom 0.16 [4] for audio processing and peak-picking.

We use a heuristic peak-picking method introduced in [5]. This method selects a peak \( \hat{y}[n] \) at \( n \) if it satisfies the three conditions in Eq. (1).

\[
\begin{align*}
\hat{y}[n] &= \max(x[n - w_m], \ldots, x[n + w_m]) \\
\hat{y}[n] &\geq \text{average}(x[n - w_a], \ldots, x[n + w_a]) + \delta \\
n &> n_{tp} + w_w,
\end{align*}
\]

where the max window \( w_m = 50 \) ms, average window \( w_a = 100 \) ms, threshold \( \delta = 0.2 \), waiting window \( w_w = 50 \) ms, and \( n_{tp} \) is the last detected peak. We mainly use F1 score along with Precision and Recall using mir_eval [28]. The tolerance window is 50 ms.

After training, we test the system on three public datasets; IDMT-SMT-Drums (‘SMT’, 104 drum recordings, total 130 minutes [10]), Medley-DB Drums (‘MDB’, 23 tracks, total 20 minutes [36]), and ENST-drums (‘ENST’, 61 minutes [14], drum-only tracks as known as ‘wet-mix’ of ‘minus-one’ subset). According to [40], a task is DTD if tracks are drum-only, or more precisely KD/SD/HH-only, and the system annotates KD/SD/HH, which is the case of SMT dataset. A task with the system annotating KD/SD/HH but with drum tracks consisting of more than those three components, e.g., tom-toms and cymbals, is named DTP in [40]. Following this convention, we evaluate DTD with SMT, and DTP with MDB/ENST.

4.2 Trend of Performance over Training

We do not employ a stopping strategy but train the network for \( 6 \times 10^6 \) items (about 13 epochs). As illustrated in Figure 4, the overall performance gradually increases as the training proceeds and is approaching a converging at the end of training. This indicates that the proposed loss function works as a good proxy of the transcription loss. After the initial phase of training, the performance differences among datasets remain consistent. This is probably due to the different characteristics of drum tracks in each dataset, as will be discussed in Section 4.4.

4.3 Relative Performance against Baselines

Since the training of the system is completely blind to the target task, we should consider our experiment setup as transfer learning scenario which measures the generalization capabilities; which is more realistic than dataset split scenario, where a dataset is split into training/evaluation subsets, hence the learning is still limited to a certain dataset. In other words, our experiment is equivalent to ‘eval-cross (trained on DTP)’ experiment in [40]. This was done on SMT, which allows us to compare the scores. Overall, the performance of DrummerNet is favorable to those of recent drum transcription systems. With average F1 score of 0.869 on SMT, the proposed unsupervised DrummerNet outperformed 9 out of 10 systems. The 9 systems include ones with deep neural networks and supervised approach, ‘ReLUUs’, ‘RNN’, ‘lstimB’, ‘tanhB’, and ‘GRUs’ [33, 34, 37, 38], as well as ones with NMF and

![Figure 4: The (three components averaged) F1 scores of DrummerNet over training items on each dataset (SMT, MDB, ENST). ‘AVG’ indicates the overall average F1 scores of three datasets.](https://www.audiolabs-erlangen.de/resources/MIR/2017-DrumTranscription-Survey)
The comparison between DrummerNet and the NMF/unsupervised learning-based systems [10, 41] implies that the proposed deep neural network structure effectively learns relevant representations. Furthermore, DrummerNet allows constant-time inference unlike NMF and other factorization-based approaches which require iterative optimization in the test time.

What is more interesting is its generalizability. In Figure 10 (b) from [40], all the deep learning based systems (RNN, tanhB, RelUts, lstmB, GRUts - RNN-based ones) present deteriorating performance in the transfer learning scenario (eval cross) compared to the dataset split scenario (eval subset). However, less data-driven approaches (SANMF, NMF, PFNMF, AM1, AM2 - NMF-based ones) present similar or even increased performances. This implies i) the distributions of datasets are fairly different and biased to certain types of drum tracks, and ii) therefore, a transcription system trained with those datasets is also biased accordingly. We conjecture that this is due to the small sizes of those datasets. In contrast, it is relatively easy to unbiase DrummerNet – one only needs to control the distribution of the style/genre/sounds of drum tracks without annotating every note.

4.4 Qualitative Analysis

In this section, we analyze the performance and the behavior of DrummerNet in detail – by components, datasets, and metrics, as illustrated in Figure 6. We notice two clear trends: First, across all the three datasets and the metrics, detecting KD was the easiest, followed by SD and HH (except the precision on SMT). Second, SMT seems the easiest, followed by MDB and ENST. What would be the reasons?

The first trend is strongly related the proposed loss function. KD has the least within-class variability while being the most distinguishable component (the largest mutual-class variability) due to its solitary frequency range. SD and HH share mid- and high-frequency ranges while their sounds can vary significantly across drum kits – i.e., larger within-class variability and smaller mutual-class variability. A common pattern, as a result, is the false positive of HH due to SD and vice versa. This is presented in Figure 7, where SD has many false positives due to HH.

The second trend is caused by the mixed use of the probability and the onset velocity in the DrummerNet. Although its transcription \( y \) is the estimated amplitude of drum components, the peak-picking method treats \( \hat{y} \) as if it were a probability. This mismatch becomes problematic when the velocities of drum events in a track vary drastically as in the case of MDB and ENST. A failure case is demonstrated in Figure 7, where the HH with strong accents on several occasion caused DrummerNet to miss many of the other HH peaks.

4.5 Ablation Study

We conduct ablation study where the performance of DrummerNet is compared with its variants. We report F1 scores averaged over datasets and components, as shown in Figure 8. Please refer to the caption in Figure 8 for the definitions of the system names.

Sparsemax (DFL vs. SOFT) Among all the variants in this experiment, we observe the most dramatic change in the performance when we replaced Sparsemax with Soft-
Comparing MEL and STFT, the melfrequency compression helps better detection of KD but not SD nor HH. This is explained by the different frequency band weighting of STFT and melspectrogram. Since melfrequency is linear below 1 kHz and logarithmic above 1 kHz [32], melspectrogram allocates relatively more bins below 1 kHz. This means that the loss function in MEL is biased towards the low-frequency range, resulting in the training favoring KD over the others.

**Onset Enhancement (DFL vs. NOE)** The onset enhancement is shown to be boosting the performance, but not significantly (0.017). From the learning curve, we observe that removing the onset enhancement in the loss function results in large performance degradation during the initial phase of training. This is mainly due to false-positives in the non-transient part.

**Recurrent layers (DFL vs. CONV)** Overall, replacing three recurrent layers with three convolutional layers does not make significant differences (0.011). This may means i) a long-term relationship might not provide additional information, probably because the transcription largely depends on local information, and ii) the mutual conditioning in the last recurrent layer is not effective in our experiment. In an informal analysis, we observed that with recurrent layers, \( \hat{y} \) still has some local temporal correlation, e.g., the activation is smeared over time, probably because that is better to reconstruct the input audio.

**5 Conclusion**

We introduced DrummerNet, a deep neural network that is trained to transcribe drum tracks without a labelled dataset. In the experiment, DrummerNet achieved a strong performance compared to existing systems that are trained with supervised learning, showing its generalizability towards a real-world drum transcription scenario. Ablation study showed that Sparsemax, CQT, and onset-enhancement played an important role for a successful training of DrummerNet.

The experiment also showed rooms for further improvements. Considering the discreteness of musical notes, a reinforcement learning approach may be more suitable [35], making the prediction more sparse and replacing the peak-picking with some trainable action. The onset-enhancement on audio similarity is a carefully-chosen function in order to approximate \( L_y \) when \( x \) and \( \hat{x} \) are given, but the approximation is limited. This is because the exact drum sounds in \( x \) are not given, therefore a perfect reconstruct of (onsets of) the input audio \( (L_y(x, \hat{x}) = 0) \) does not mean a perfect transcription \( (L_y(y, \hat{y}) = 0) \). An alternative way would be measuring a similarity on a (perceptual) representation domain instead of the audio, for example, by learning a loss using forward-backward consistency (as known as cyclic loss [17]) or known audio features. Lastly, the current synthesizer module is limited to drums as it does not handle the duration of notes. A trainable synthesizer can be used to expand DrummerNet to the other instruments [3, 11].

**Figure 8**: The ablation study results, F1 scores averaged over three datasets per component (KD, SD, HH) and their overall average (AVG). The label indicates as follow: DFL (default DrummerNet as introduced), SOFT (two Softmax layers instead of Sparsemax), MEL (use 128-band melspectrogram instead of CQTs), STFT (use 1024-point STFT instead of CQTs), NOE (not onset enhancement in loss), CONV (3-layer convolutional layers instead of recurrent layers).

**Figure 9**: A transcription example of SOFT (DrummerNet with Softmax) , ‘Real Drum 01-12’ in SMT - the output of analysis module (top), after peak-picking (middle), and ground truth (bottom); KD, SD, HH (left to right).
6 References


