Deep Plunder

David Braun [1]

Abstract

We use a deep recurrent neural network to identify which regions of music recordings contain isolated music production samples. We use parallel multiprocessing to generate 50 hours of fake music consisting of randomly chosen library samples and real song excerpts sequenced with various augmentations. The neural network uses supervised learning with an audio spectrogram as input and a one-dimensional labeled region as output. We achieve low error rates on our synthesized train and test datasets but inadequate qualitative results on real music. We believe our network has potential to generalize to real music, but to avoid time-consuming manual labeling, we must research better data generation and augmentation techniques.

1. Introduction

Many genres of music involve sampling sounds from one source and applying them to new compositions. Pop musicians might rely on short recordings of guitar strums, bass plucks, drum hits, and synthesizer one-shots. The genre of plunderphonics, established by John Oswald in the 1980s, pushes sampling to its extreme by making songs only out of sampled material[1]. In this paper we seek to enhance the abilities of artists who plunder sampling material. This act of plundering once involved digging through dusty crates of vinyl records or carefully extracting clips in digital audio workstations. With the advent of neural networks, we hope to enable artists to sample music on a larger scale with ease. We acknowledge that increased automation makes the practice of sampling a more passive activity. On the other hand, saving time in plundering samples allows artists to find more samples in the same amount of time or to spend more time composing.

Our network takes an audio spectrogram as input and returns a one-dimensional vector whose length is the number of frames in the spectrogram. The vector's values indicate the likelihood of the audio frame being a sound library sample with no interference from other sounds. A simple script processes the vectors and saves extracted regions to new audio files.

2. Related Work

Onsets-Frames In the task of automatic piano transcription, Hawthorne achieved state of the art performance by showing that training on onsets and frames performs better than using frame information only (2017). To predict MIDI, Hawthorne's architecture took audio spectrograms of piano performances as input and returned two two-dimensional outputs, onsets and frames. Each output's shape is the number of Short-time Fourier transform (STFT) time bins by the number of possible note pitch predictions. The onsets indicate the moment at which piano note starts. The entire audio spectrogram goes through two parallel convolutional networks. One path leads to a bidirectional LSTM network that ultimately predicts onsets with cross-entropy loss. This onset-prediction path is concatenated with the other convolutional network and enters a second bidirectional LSTM network that predicts frames. In a later work, a similar path for note offset prediction is concatenated like the onset prediction path (2018).

Few-Shot Learning In the task of Automatic Drum Transcription (ADT), Wang trained a Prototypical Network on a synthetic dataset and evaluated on real polyphonic drum music (2020). At inference, Wang's few-shot model takes a handful of drum sounds as input and returns regions containing similar sounding sounds. Wang's model generalizes well to sounds not seen during training. The model's embedding network is a convolutional network that processes overlapping 250 ms windows converted to spectrograms. The results are concatenated together forming a two-dimensional matrix with the same size as the input. If multiple drum sound classes are heard at the same time, they will still be predicted. In contrast, we desire a system that identifies regions with only a single class of sound occurring.

[1]Center for Computer Research in Music & Acoustics, Stanford University, Stanford, CA, USA. Correspondence to: David Braun <braun@ccrma.stanford.edu>.

CS229: Machine Learning, Stanford University, Fall 2020. Copyright 2020 by the author(s).

[1]For sampling-based music, we recommend DJ Shadow's *Entroducing.....* (1996), The Avalanches' *Since I Left You* (2000), J Dilla's *Donuts* (2006), and Girl Talk's *All Day* (2010).

Drum Fills Lopez-Serrano used random forests as a classification scheme and a technique originally for Singing Voice Detection (SVD) to detect drum-only passages in a labeled music dataset (2017). This SVD method involves an explicit feature transformation of an input spectrogram. For example, a horizontal median filter reveals harmonic content, and a vertical median filter reveals percussive content. We choose to not manipulate our spectrogram features because a deep-learning algorithm can potentially find better features.

Scaper Salamon created *Scaper*, a Python library for generating soundscapes relevant to Sound Event Detection (SED) (2017). This library allows users to place foreground sounds such as car horns or power-tool drilling on top of background sounds such as ambient street noise or low-volume weather. Users can specify probability distributions for many aspects of data augmentation. *Scaper*'s features influence our data synthesis and augmentation strategies.

3. Dataset and Features

3.1. Overview

We synthesize music consisting of music production samples and excerpts of real songs. For convenience, we'll refer to sounds from the music production library as simply "drum sounds" or "the drum library" for the remainder of this paper. Our resulting training set consists of 2,826 minutes of synthesized music, and our test set consists of 174 minutes of synthesized music[2]. For training, we dynamically resample to 16kHz mono audio and create mel-scaled spectrograms with log amplitude and 229 logarithmically-spaced frequency bins, as was done by Hawthorne (2017). We use an FFT window of 2048, a Hanning window, and a hopsize of 512.

3.2. Music Production Library Data

We have a private collection of thousands of samples such as synthesizer one-shots and drum hits. We use `librosa`'s `onset_detect` function to identify the starts and ends of the drum sounds (McFee et al., 2017). We save the onset-offset info in JSON for quick recall when generating music. Figure 1 shows a verification of our automated onset and offset detection. Our technique is effective in most cases, but some sounds with very long tails end up with offsets labeled too early.

3.3. iTunes Data

We use iTunes to create a playlist of one thousand hours of music (16,407 songs) and export the playlist to a tab-separated file. We choose songs that involve sounds typically found in a sample library such as ours. Each song has a bit-rate above 200 kbps and a duration between one minute and 6 minutes.

[2]We recognize that this is an improper train/dev/test split.

3.4. Synthesized Data

3.4.1. Multiprocessing

We use Python's multiprocessing framework to generate fake music (Van Rossum & Drake Jr, 1995). One process loads short sections of iTunes songs into a queue. A parallel process loads drum sounds into a queue as well as their onset-offset info from JSON. A queue cycles when reaching the end of a file list. We use a CPU with 8 threads to generate fake music. Each thread randomly takes from both queues and sequences the audio. The thread appends excerpts until it has one minute of material. Then it saves the audio to a 16-bit 44.1kHz stereo WAV file with *scipy* (Virtanen et al., 2020). It also saves a tab-separated file (TSV) containing onset-offset pairs of drum regions. Figure 2 shows the information from the TSV imported as an annotation layer on top of the generated audio waveform.

3.4.2. Rebalancing

Our drum library has an unequal amount of sounds for each class. For example, there are more hi-hat sounds than Maracas. Therefore, we allow under-represented sounds to appear with a frequency as much as 30 times higher than otherwise but without exceeding the frequency of the most common class (snare drums).

3.4.3. Additional Data Augmentation

We use *librosa* again to re-pitch the drum sounds in integer semitones ranging from -3 to 3. We use a suite of easing functions to fade the starts and ends of iTunes songs with random sharpness (Christianos, 2019). We use the same library to fade the ends of drum sounds longer than 4 seconds to a randomly shorter duration. Our pre-processed onsets and offsets of the drum sounds enable us to align sounds with a desired amount of overlap, but we choose zero overlap.

4. Methods

4.1. Neural Network Design

Our network is most similar to a system that previously achieved state of the art performance in automatic piano transcription (Hawthorne et al., 2017). As discussed in Section 2, this architecture showed that training on onsets and frames performs better than using frame information only. Therefore our architecture has three outputs: onsets, frames, and offsets. Piano transcription involves predict-

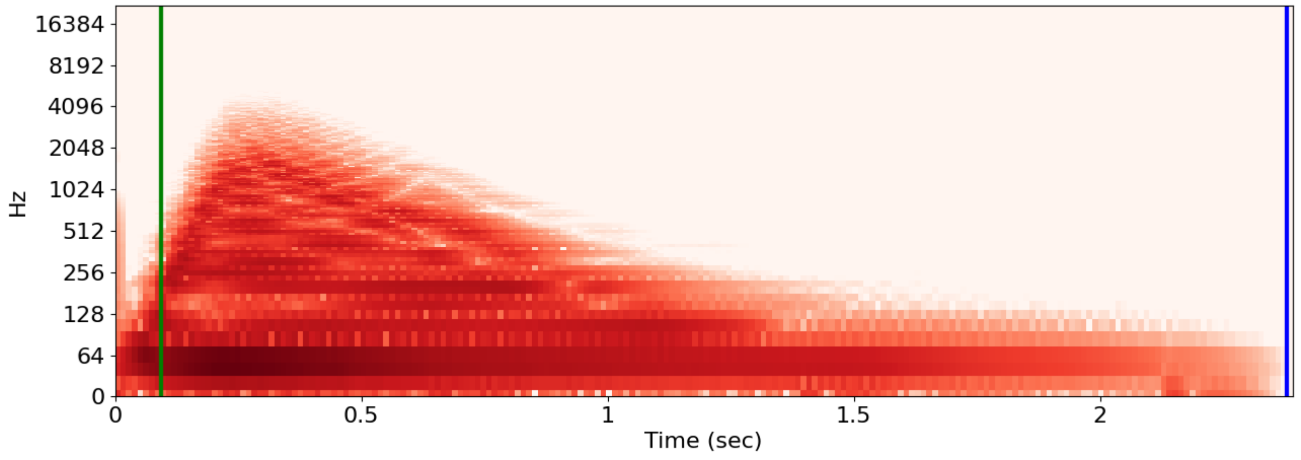

Figure 1. Example spectrogram of a synthesizer sound with onset and offset detection depicted in green and blue respectively. To detect offsets, we simply flip the signal and perform onset detection again. This works well in most cases.

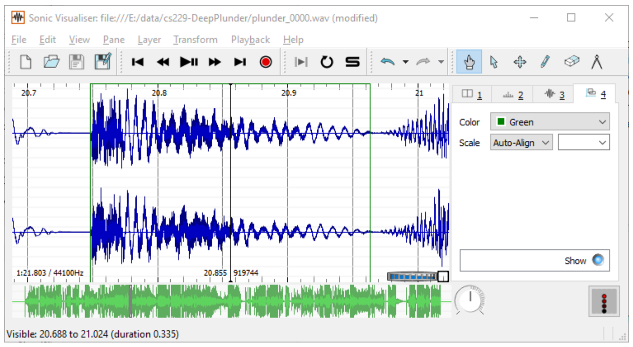
Sonic Visualiser: file:///E:/data/cs229-DeepPlunder/plunder_0000.wav (modified)
File Edit View Pane Layer Transform Playback Help
20.7
20.8
20.9
21
Color Green
Scale Auto-Align
Show
1:21.803 / 44100Hz
20.855 919744
Visible: 20.688 to 21.024 (duration 0.335)

Figure 2. Example of a generated song displayed in Sonic Visualiser with an annotation layer (green rectangle) depicting a labeled drum sound (Cannam et al., 2006).

ing two-dimensional vectors because a piano has multiple pitches, but we are predicting only a single kind of note, the presence of drum sounds. Therefore our outputs are one-dimensional vectors.

Both Hawthorne and Wang used convolutional networks at the start of their networks. One such stackable encoder unit uses 2D convolutions followed by batch normalization, an activation layer, 2D max-pooling and dropout. Out of a desire to try something new, we swapped the two-dimensional aspects of this unit to one-dimensional ones. The 2D convolution became a bidirectional LSTM that takes the mel-frequency bins as its features (Berglund et al., 2015). The 2D max-pooling became 1D max-pooling, which allows the duration of the output sequence to stay the same and for our new unit to be stacked. Figure 3 shows an overview of the network. In contrast to Wang and Hawthorne, we can process audio of any duration without processing an audio input in overlapping batches and blending the results.

We start our code-base by porting Kim's PyTorch implementation of Onsets-Frames to Tensorflow 2 (Kim, 2019). We also use Kim's evaluation code to interface with *mir_eval* (Raffel et al., 2014).

4.2. Loss Metrics

We use cross-entropy loss for all outputs and weight them equally. We also use a weight of .001 for the kernel regularizer on all bidirectional LSTM cells. Our total loss is the sum of the three cross-entropy losses and this regularization loss.

4.3. Hyperparameters

We specify a single hyperparameter for overall model size. For the 1D max-pooling layers, we tried different pool sizes. We also tried different percentages for the dropout layers.

4.4. Training

We use an `Adam` optimizer (Kingma & Ba, 2017) in TensorFlow (Abadi et al., 2015) with a learning rate of 0.0006, decay-rate of 0.98, and decay-steps of 10000. We use gradient clipping with a value of 3.0. We train for 50 epochs with early stopping on total loss. To maximize data I/O throughput, we use a mini batch-size of 16. We choose 80 steps per epoch to process about half the training set per epoch. We run our validation data every 5 epochs and choose the best model based on total loss on the validation set[3]. We manually picked hyperparameter values over several trials

[3]In retrospect, we could have tied several decisions to frame loss rather than total loss.

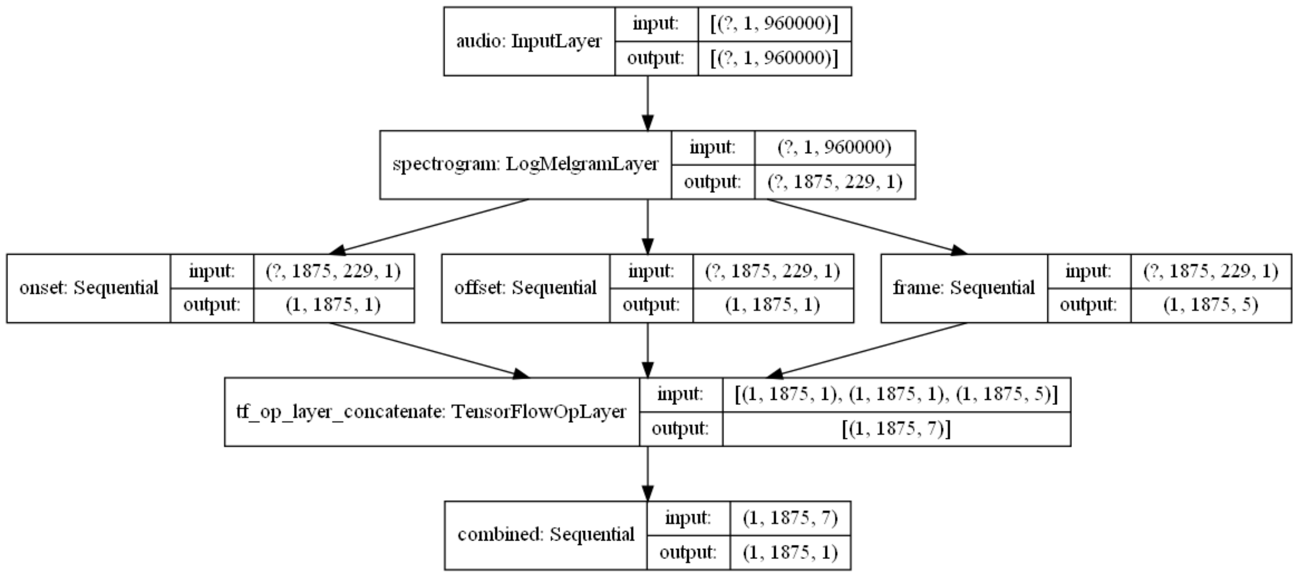

Figure 3. Model with 60 seconds of 16kHz mono audio going into a spectrogram shaped (1875, 229). The onset, offset, and frame layers contain our stacked bidirectional LSTM units inside a `Sequential` layer. Our model has 3,563,948 trainable parameters and 4,342 non-trainable parameters. The number of parameters in the model does not depend on the number of STFT input frames.

to arrive at our final model.

5. Results

5.1. Metrics and Analysis of Test Set

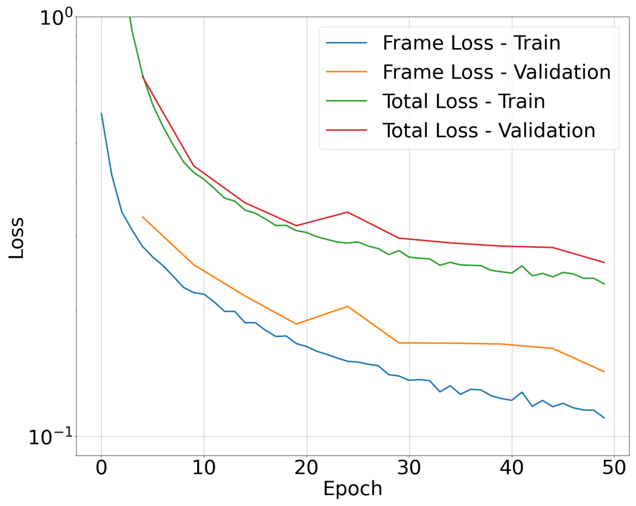

Figure 4. Losses from training with logarithmic scale. The divergence between validation loss and training loss is undesirable.

Figure 4 shows our training history. We reached 50 epochs, and our early stopping didn't interrupt. The validation loss on frames is worse than the training set, indicating a problem with over-fitting and high variance. A better training period might require fewer model parameters, fewer features, or more regularization. Validation losses on onsets and offsets were better than those of training. There is likely a competing relationship between the three cross-entropy losses that could be explored in future research.

Table 1. Metrics on Test Set

OUTPUT	METRIC	VALUE (%)
NOTE	PRECISION	95.6 ± 06.3
NOTE	RECALL	41.9 ± 10.7
NOTE	F1	57.5 ± 10.7
NOTE	OVERLAP	65.6 ± 11.6
NOTE W/ OFFSET	PRECISION	52.9 ± 16.8
NOTE W/ OFFSET	RECALL	23.6 ± 10.6
NOTE W/ OFFSET	F1	32.1 ± 12.5
NOTE W/ OFFSET	OVERLAP	91.7 ± 09.1
FRAME	PRECISION	96.6 ± 03.8
FRAME	RECALL	59.1 ± 17.2
FRAME	F1	71.8 ± 14.4
FRAME	ACCURACY	57.9 ± 16.8
FRAME	MISS ERROR	40.9 ± 17.2
FRAME	FALSE ALARM ERROR	02.1 ± 02.6
FRAME	TOTAL ERROR	43.0 ± 17.1

Table 1 shows our evaluation with the *mir_eval* library (Raffel et al., 2014). For the `Note` and `Note w/ Offset` entries, we used the `precision_recall_f1_overlap` function. An estimated `Note` is assumed correct if its onset is within ± 50 ms of a labeled frame. This rather strict requirement might explain the low recall scores. For `Note w/ Offset`, there's an additional requirement that a correctly estimated frame have an offset value within 20% of the labeled frame's duration around the labeled frame's offset, or within 50 ms, whichever is larger. The `Frame` metrics use the `multipitch.evaluate` function, which has different assumptions. Overall, our results show high precision with moderate recall, considering the strict requirements of the metrics.

We also use *scikit-learn* for evaluation (Pedregosa et al., 2011). Figure 5 shows the precision-recall plot. Frames show a good tradeoff, while the onsets fare worse. One explanation is that onsets exist in single STFT time-frames, and the network returns high onset prediction values for a wider time window.

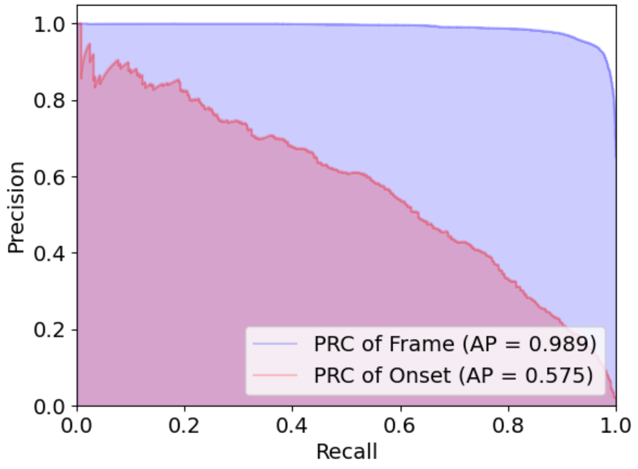

Figure 5. Precision vs. Recall: The average precision (AP) of frames is higher than that of onsets.

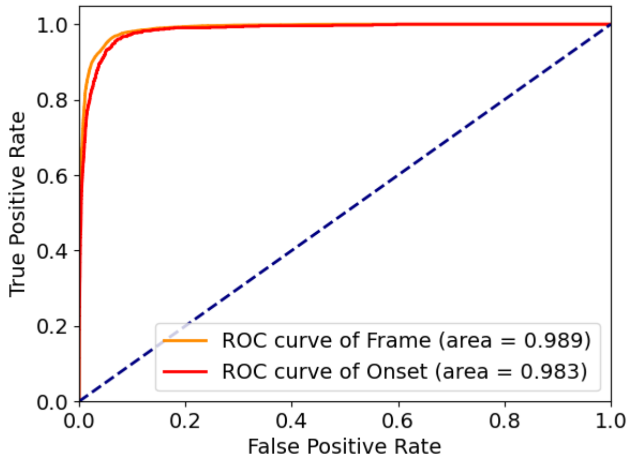

Figure 6. Receiver Operating Characteristic Curves

Figure 6 shows the Receiver Operating Characteristics (ROC) of frames and onsets. The area under the curve is high for each.

5.2. Automatic Sampling of Test Dataset

Our algorithm returns one-dimensional vectors whose length is a number of spectrogram frames. We convert this time axis to the higher sampling rate of the input audio. We iterate through the predicted onsets and look for values above a threshold of 0.2. When an onset is found, we move frame-by-frame until the predicted frame output is less than a different threshold, which happens to also be 0.2[4]. Then we save the clip to a WAV file. As expected based on our evaluation metrics, the algorithm effectively extracts clips from the test set.

5.3. Automatic Sampling of Real Music

Applying our clip extraction process to full songs from the iTunes playlist leads to incorrectly extracted audio, assessed qualitatively. Choosing different thresholds in the script results in more bad extractions. As a sanity check, we load the predicted onsets and frame vectors as annotation layers in Sonic Visualiser. The onsets find real drum transients well, but frame predictions are generally too low. This likely indicates that our generated music diverges from real music too much in terms of how drum sounds and non-drum sounds are interspersed.

Our data generation technique could have included several mistakes. We made the time spacing between sounds too large or too consistent. We didn't strike a good balance between the prevalence and duration of song excerpts and drum sounds. We didn't sufficiently augment the volumes, pitches, speeds, and various effects of all sounds. We didn't choose good volume fadings to apply to real song excerpts. We didn't strike a good balance in representing drum sounds of various classes.

6. Conclusion

We present a deep recurrent neural network that extracts drum sound regions from audio. We efficiently synthesize data for our network, train the network, and achieve high or acceptable performance metrics. However, our network performs inadequately on real music. For further research, we propose investigating the weaknesses of our generation process and setting up baselines with other classification systems.

[4]Hawthorne demonstrated a more sophisticated alignment technique as a post-process (2017).

Acknowledgements

The author would like to thank the brilliant and hard-working teaching assistants and lecturers of CS229 at Stanford.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL https://www.tensorflow.org/. Software available from tensorflow.org.

Berglund, M., Raiko, T., Honkala, M., Kärkkäinen, L., Vetek, A., and Karhunen, J. Bidirectional recurrent neural networks as generative models - reconstructing gaps in time series, 2015.

Cannam, C., Landone, C., Sandler, M. B., and Bello, J. P. The sonic visualiser: A visualisation platform for semantic descriptors from musical signals. 2006.

Christianos, F. easing-functions. https://github.com/semitable/easing-functions, 2019.

Hawthorne, C., Elsen, E., Song, J., Roberts, A., Simon, I., Raffel, C., Engel, J. H., Oore, S., and Eck, D. Onsets and frames: Dual-objective piano transcription. *CoRR*, abs/1710.11153, 2017. URL http://arxiv.org/abs/1710.11153.

Hawthorne, C., Stasyuk, A., Roberts, A., Simon, I., Huang, C. A., Dieleman, S., Elsen, E., Engel, J. H., and Eck, D. Enabling factorized piano music modeling and generation with the MAESTRO dataset. *CoRR*, abs/1810.12247, 2018. URL http://arxiv.org/abs/1810.12247.

Kim, J. W. onsets-and-frames. https://github.com/jongwook/onsets-and-frames, 2019.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization, 2017.

López-Serrano, P., Dittmar, C., and Müller, M. Finding drum breaks in digital music recordings. In *CMMR*, 2017.

McFee, B., McVicar, M., Nieto, O., Balke, S., Thome, C., Liang, D., Battenberg, E., Moore, J., Bittner, R., Yamamoto, R., Ellis, D., Stoter, F.-R., Repetto, D., Waloschek, S., Carr, C., Kranzler, S., Choi, K., Viktorin, P., Santos, J. F., Holovaty, A., Pimenta, W., Lee, H., and Brossier, P. librosa 0.5.1, May 2017. URL https://doi.org/10.5281/zenodo.1022770.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Raffel, C., McFee, B., Humphrey, E. J., Salamon, J., Nieto, O., Liang, D., Ellis, D. P., and Raffel, C. C. 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.

Salamon, J., MacConnell, D., Cartwright, M., Li, P., and Bello, J. P. Scaper: A library for soundscape synthesis and augmentation. In *2017 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, pp. 344–348, 2017. doi: 10.1109/WASPAA.2017.8170052.

Van Rossum, G. and Drake Jr, F. L. *Python tutorial*. Centrum voor Wiskunde en Informatica Amsterdam, The Netherlands, 1995.

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., Carey, C. J., Polat, İ., Feng, Y., Moore, E. W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E. A., Harris, C. R., Archibald, A. M., Ribeiro, A. H., Pedregosa, F., van Mulbregt, P., and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi: 10.1038/s41592-019-0686-2.

Wang, Y., Salamon, J., Cartwright, M., Bryan, N. J., and Bello, J. P. Few-shot drum transcription in polyphonic music, 2020.