Onset Detection Challenge 2022

Laurenz Hundgeburth - Team: NeuraBeats

This report describes my final approach and highlights some experiments I did during this challenge. The code of my submission is available on Github (https://github.com/LaurenzBeck/music-processing-challenge), where I also documented the "technical" experiments I did to get to know some libraries in the `notebooks` directory.

First Plans

I wanted to use this challenge as an opportunity to try out fastai and dvc.

I just finished an online course on the Deep Learning framework fastai and I also found the fastaudio library on GitHub (https://github.com/fastaudio/fastaudio), which contained a lot of data primitives and preprocessing functionality, to deal with .wav audio files. I did encounter some installation problems due to overly strict dependency constraints and I faced some bugs applying the first official tutorials, which is never a good sign.

DVC (data version control) is an extension of git, that enables full reproducibility, by not only versioning code, but the complete ML lifecycle including the data, configuration, artifacts, metrics and models. Additionally, it provides a data pipeline functionality (specified in a `dvc.yaml` file), that efficiently caches and versions intermediate results.

Unfortunately, I wasn't allowed to make the dataset public, that is why a third party can't access my dvc storage backend and call `$dvc repro` to reproduce the whole pipeline. Only the code, metrics and configurations are accessible publicly.

Given my uncertainty on how to approach the modelling part with fastaudio (I have no experience with sequence-to-sequence modelling) and the clarification, that we should implement as much as possible ourselves, I decided to go for a more hands-on approach, where I frame the problem as a tabular binary classification problem.

Tabular Approach

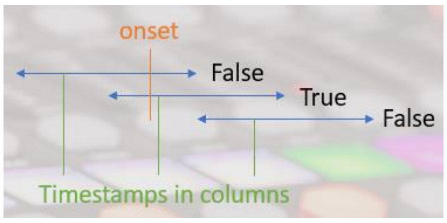
I first looked into scipy, librosa and madmom for io handling and possible features. I liked the processor interface of madmom and decided to go for a sliding-window approach based on the `FramedSignalProcessor` class from madmom. Feature extraction for each frame in the signal was not very challenging, as in all of the libraries, I looked into supported `NumPy.arrays` as input. During my experiments, I tried a lot of different features, including a superflux implementation from (https://github.com/CPJKU/madmom_tutorials/blob/master/processors.ipynb), NumPy summary statistics functions like mean and std, signal features like a signals energy, its rms, the sound pressure level, or the average spectral density using the welch method, and finally other statistical features including coefficient of variation, skew and the kurtosis. At that time, I hadn't implemented the model training and validation part yet, that is why I included all of them at this stage.

The label alignment was a harder problem to solve, and two very mean errors in my implementation caused me so much headache and confusion. For this sliding window approach, I had to decide on a frame length and hop size (or implicitly through frames per second) and whether the frames should overlap or not.

Having longer frames increases the spectral resolution of the FFT computations and I thought that longer frames might also be more likely to capture the peak information through the summary statistics I used. The hop size determines the possible resolution of my predictions, as I discretize the space of possible onset locations. One tradeoff I went for was using overlapping frames, which allowed me to use long frames and small hop sizes. The disadvantage now is that multiple frames might contain the same onset.

One possible solution I thought of was allowing this and including an additional merging step after the binary onset predictions, but ultimately, this approach was too complicated for me at that time.

That is why I went for the solution of only labelling the window/frame as containing an onset, if the onset timestamp is at the start of the frame, before the start of the next frame. This way, my model not only had to learn if there is an onset in the frame but also where it is.



Having a columnar dataset of timestamps, features and binary onset labels, I was able to tackle the modelling using fastai's tabular learning utilities. I spent a lot of time and effort at this stage tuning and experimenting with model-specific details, but ultimately decided that this effort could have been better spent on the preparation of the data and the extraction of the features (learned this the hard way here). My final modelling configuration and tricks included feature normalization, a simple MLP (hiddel_layers=[256, 128, 128]), Monte Carlo dropout, Lamb optimizer, class-weighted label smoothing CE loss, one-cycle learning rate policy, batch_size: 128, epochs: 64 learning_rate: 0.003 weight_decay: 0.0003.

I evaluated my model using a consistent 80/20 train/val split on a file-level (not on the dataset columns) to better estimate generalization capabilities. I used classification metrics, but also the onset_f1_score function from the mir_eval library, which was also used to calculate the final challenge score on the test set.

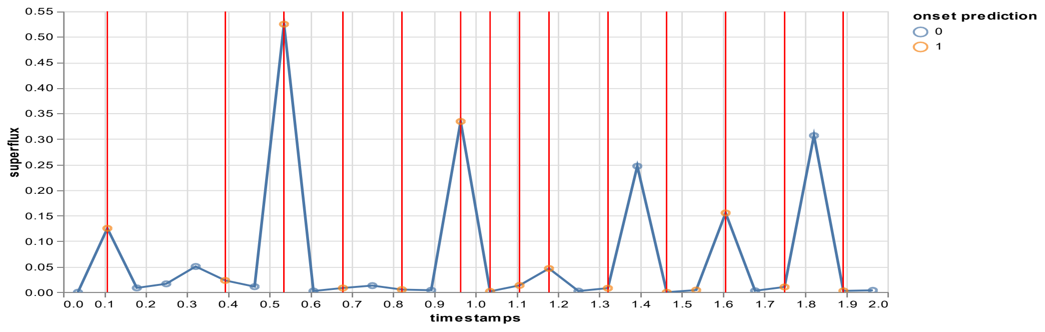
Additional Filtering and Diffs

With very mediocre results of maximally 0.4 F1-score on the test set, I thought that maybe giving the model access to more temporal context might improve the results. Therefore, I decided to use an exponentially weighted moving average filter with spans of three and six to every feature in the dataset (again on file level). Inspired by the difference step in the superflux features, I also decided to add the difference to the last feature for every feature and also filtered these new feature diffs. Unfortunately, this brought only a minor performance gain.

Non-overlapping Frames

Only after writing the first draft of this report, did I notice, that I was so distracted by the label alignment problem, that I didn't question my approach. Noticing this, I finally tried a frame length and fps combination without overlap (14 fps, 3150 frame size) and the F1-score improved to 0.42 on the test set.

Although this is only a 5% improvement over the last approach, I am more content with the plots of the features and the onsets, as they finally correlate better (red lines are actual onsets aligned to the frame grid):



Reflexion

I look back on this project with mixed feelings. I am super content and proud about the engineering part with dvc, but at the same time, it took way too much time to set up and it diverted my attention from doing more diverse experiments.

Although knowing the advantage of a data-centric vs. a model-centric approach to such an ML project, I again fell into the trap of indulging in model and training tweaks instead of focusing on a strong foundation with good features. Exploring a new Deep Learning framework maybe wasn't a good idea for such a challenge with limited time. I wished I had spent more time exploring different features.

Another thing that bothered me was that my classification metrics were coupled to the frame configuration, which made it very difficult to compare experiments with different frame settings. One funny example was a local minimum I found by setting the frame width and hop size in such a way, that the labels were almost evenly distributed (50% onsets). The crude time frame in combination with the high acceptance interval of +-50ms in the evaluation code led to good results, even though the model was arguably bad.