Audio and Music Processing Solution Description

Onset Detection

This uses the SuperFlux algorithm, which of course is an extension to the spectral difference based LogFiltSpecFlux (LFSF) algorithm. This algorithm has 7 hyperparameters ($w_1, \ldots, w_5, \delta$ and the size of the maximum-filter s_{mf}) which we optimiized using Bayes Search on the provided set of extra onset annotations. We chose this search procedure over grid-search, as in our experience it leads to better results with fewer parameter evaluations. To implement this, we wrapped the detection script in a simple sklearn estimator class, defined the score as in evaluate.py and then used the parameter search provided by scikit-optimize. This search would usually be intended for cross validation, but by manually defining a single fold with every datapoint in the validation set we can use it for our intended purposes. As the parameter choice $s_{mf} = 1$ was found to be optimal, our implementation actually chose to do LSFS rather than SuperFlux (a maximum-filter with size one does not change the data).

Tempo Estimation

Our first attempt at tempo estimation was based on TempoCNN as described in the original paper. Sadly, this approach did not work out, as the model only ever got a p-score of around 0.3. We believe this to be due to a lack of training data. To alleviate the data shortage, we experimented with three data augmentation strategies. First, as described in the paper, we implemented random interpolation and clipping along the time axis of the spectrogram while adjusting the tempo to match. Second, we used the probabilities often provided in the ground truth tempo annotations to randomize which tempo would be used as the target, to better reflect the annotation results. Lastly, because at this point we suspected the issue to be a lack of coverage over the [30-286]bpm range used in the paper, we tried "lowering the resolution". Concretely, instead of rounding to the nearest bpm, we would round to e.g. the nearest multiple of 2 in the hopes of increasing the number of samples per classification bin. Unfortunately, all this did not come to fruition and was replaced by the simpler algorithm described below. One thing that may have saved this approach would have been testing different model architectures, but as this can get quite time consuming we decided against it.

Ultimately, we used simple Autocorrelation to find the most prevalent periodicity in the onset detection function (ODF). Of the tempos in the interval [60-200]bpm the one with the highest autocorrelation is chosen as our estimate. Further, we also return half the found tempo as our second estimate, simply because it worked well in practise.

Beat Detection

As both our onset and tempo detection algorithms worked quite well, we chose to try the simplest algorithm imaginable first and go from there. That is, we generate a pulse train from an estimated tempo and then use cross-correlation with the ODF to find the phase (start) of this pulse train. Finally, we go through each pulse and either pick the onset nearest to the pulse middle within some threshold or the pulse middle itself as a beat. Of course, as we estaimate two tempos, we do this with both of them and then compare their cross-correlation at their corresponding phases with each other. This comparison was based on the idea that a tempo with twice the bpm should have close to twice the

`cross-correlation` to actually be better, as a doubled tempo has twice as many peaks in the pulse train and can therefore gather twice as many values from the ODF.

For choosing good values for both the width of each pulse in the pulse train and the multiplier for the cross-correlation comparison we again employed `Bayes Search` as described above. The final value for the `multiplier` was close to `1 instead of 2` as we had expected and the score also plateaued in that region. This leads us to believe that the best choice is actually to `always pick our higher bpm tempo estimate`, which is the one for which autocorrelation was the highest.

The estimates generated using this approach immediately put us at the top of the online ranking, so we decided to just stick with this method.

Code

Throughout development we usually began by implementing our ideas in a jupyter notebook with little regard for readability and then tranferred the code to `detector.py` if the idea worked out. Therefore, all of our algorithms used for the final predictions are fully implemented in `detector.py`. However, things we only did once, like Bayes Search for the hyperparameters, we did not bother cleaning up and they remain in their jupyter notebooks.

The code for TempoCNN and its training is also included in this submission to show our work, but it is not used anywhere due to the reasons stated above.

Usage

To handle dependencies we used an anaconda environment. The file `env.yml` should be enough to create the `music` environment we used. Should there still be problems, then use the file `music_env.yml` as it is a direct export of our environment.

We used the given template for implementing our algorithms, so the file can be used quite similarly. The only thing that changed is the need to specify the needed hyperparameters. The following shows the command with the best hyperparameter choices found by Bayes Search:

```
$ conda activate music
$ python detector.py train train.json 1 3 4 3 11 0.01556 0.08216 7 1.3
$ python evaluate.py train train.json
Onsets F-score: 0.7419
Tempo p-score: 0.7134
Beats F-score: 0.6317
```

The test predictions were generated using the following command:

```
$ python detector.py test final_predictions.json 1 3 4 3 11 0.01556 0.08216 7 1.3
```

Results

Our submission acchieved the following scores on the test data:

```
Onsets F-score: 0.758
Tempo p-score: 0.816
Beats F-score: 0.654
```