Coursework 1 - Beat Tracking

ECS7006 - Music Informatics

Primenta Aikaterini Maria - ec24786
Student ID: 240447658

The pre-trained, best model can be found here: https://drive.google.com/file/d/1vQ26ESexY5PYdBDIeBBzM9WjfbuDbNIe/view?usp=sharing

Introduction

Neural networks have become extremely popular nowadays for Music Information Retrieval (MIR) tasks due to their ability to learn complex patterns from audio data. While most of the traditional signal processing methods perform well, they have limited adaptability to diverse music styles and tasks and might miss detecting useful information. All the above reasons work as a strong motivation to follow a Neural network approach for the beat tracking task. Inspired by several methods implemented for beat tracking, such as Böck et al. and Davies & Böck, but also by the labs in ECS7013P Deep Learning for Audio and Music module, I experimented with multiple Neural Network architectures to detect the beat in the Ballroom dataset.

Method

1. Pre-processing

A) Dataset

The given dataset for this assignment is the Ballroom dataset (http://mtg.upf.edu/ismir2004/contest/tempoContest/node5.html), which contains 698 file audios from 8 different music genres (Cha Cha, Jive, Quickstep, Rumba, Samba, Tango, Viennese Waltz, Slow Waltz). Each audio file is approximately 30 seconds long. In addition, we were given the beat annotations for these files by Florian Krebs (https://github.com/CPJKU/BallroomAnnotations). Each file has 2 columns. The first column describes the time steps (in seconds) where a beat exists and the second one is the metrical positions of a bar. In this assignment, I did not try to detect the downbeats but focused only on the beat detection, so the second column was not used at all.

B) Input features

The load_data.py file is responsible for loading the dataset in a specific form so that it can be used effectively in the training process. The create_dataset function pairs the paths of the audio files together with their corresponding annotations. The dataset then is split into training and validation sets (80% training, 20% validation). The next step is to change the training and test sets in order to be compatible to our model. BeatDataset class is created for that reason. The primary idea was to use the pre-trained VGGish model to extract features from the audio samples. Thus, we use the mel-spectrogram function used for the VGGish to extract the mel-spectrograms for each audio sample. A designing choice here was to extract only 10 seconds from each audio (10th second to 20th second) to accelerate our training time. Another reason for that was to avoid some changes in the tempo as much as possible which would confuse our model.

One of the most important steps during pre-processing was to convert the annotations into a binary values so that they would serve as a good ground truth representation. In order to do that, I converted the time annotations into samples and for every single frame, I started checking if the beat samples fall within the specific frame. If that was true, I was marking the frame with 1 (beat exists). One issue that arises from that is that the frames overlap and thus a beat (a value 1 in the matrix of frames for the ground truth) might be included in more than one frame. As hop length is 160 samples and frame size is 400 (specifications from the VGGish itself) we concluded that a beat can be present in maximum 3 frames simultaneously. In the final approach, the beat exists in all frames. However, I tried 2 other ways for beat marking: 1) if beat exists in 2 consecutive frames, mark the first one 2) if beat exists in 3 consecutive frames, mark the middle one. The results did not improve with this assumption, which might be due to the strict constraint of the frame marking. Finally, Dataloader was used to create batches for feeding the model.

2. Architectures

I experimented with different architectures, as most of them underfitted. Here are some of the most significant ones:

A) VGGish and MLP

As mentioned before, my primary goal was to use the VGGish pre-trained model, as it is trained in a wide dataset of audios. Therefore, the first architecture was inspired by the lab in ECS7013P Deep Learning for Audio and

Music module, where we implemented a Neural network combining this model (unfreezing the last layer so that it can be trainable) and a classic MLP. The architecture can be found in the train_model.py file, commented out.

B) CNN

The next step was to try a different type of Neural Network, as it seemed like VGGish with MLP did not give satisfactory results. I decided to use a basic Convolutional Neural Network with three 2-dimensional convolutions with output sizes 32, 64, 128 respectively and kernel size 3. Max pooling was also used after each convolution with kernel size and stride 2. The architecture can be found in the train_model.py file, commented out.

C) VGGish + BLSTM

Inspired by the scientific paper of Böck et al., in which they used BLSTM for beat tracking, I decided to use BLSTM as well. Going back to the idea of using VGGish, I used the pre-trained VGGish for the first layers, again freezing all the parameters except the ones in the final layer and I added a single BLSTM layer. The architecture can be found in the train_model.py file, commented out.

D) CNN + BLSTM

The final combination was to combine the CNN with the BLSTM, without using the VGGish. This is also the architecture that I maintained in the final version of my code. The CNN follows the same structure as before, including batch normalisation and I added the 1 BLSTM layer at the end. It is important to mention here that the output dimension of all these architectures is 64, as this is the time steps dimension that the VGGish was extracting when calculating the mel-spectrogram. No activation function was added to any of the architectures as it is applied to the loss function in the training.

3. Training

The training process is the train function into the train_model.py file. It follows the guidelines of the lab in ECS7013P Deep Learning for Audio and Music module. I used BCEWithLogitsLoss loss function, which uses the sigmoid function as activation function, but with tuning the parameter pos_weight so that the positive values (1 for beat) have equal weight with the negative ones, as they are obviously fewer. For optimiser, I use Adam with learning rate 0.01 (also experimented with 0.001). The training takes 50 epochs to finish and the batch size is 32, which can be reduced for accelerating training, especially if we are using GPU.

Evaluation

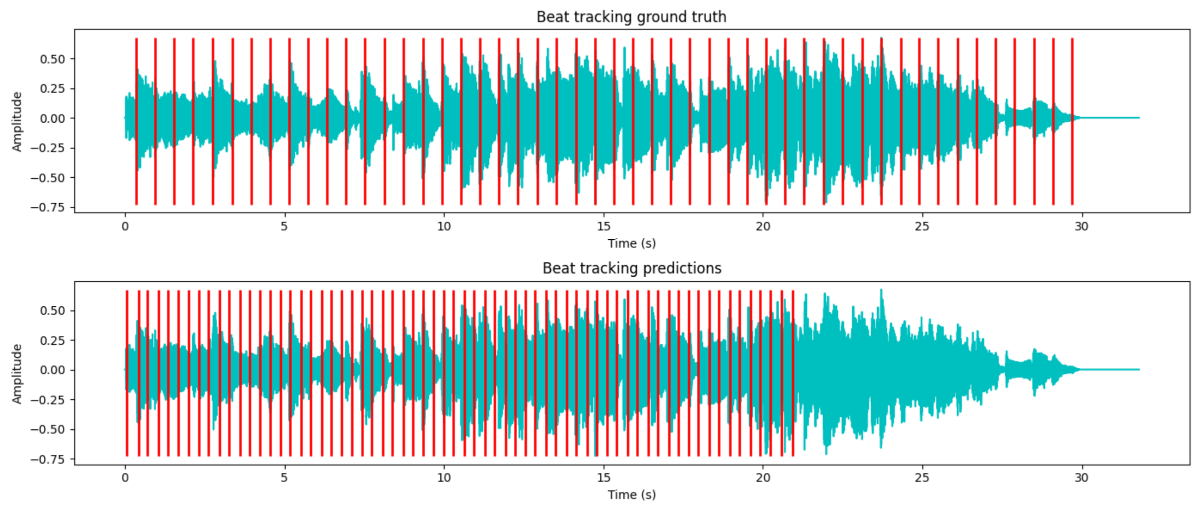
The evaluation occurs after each iteration. In order to compare the predictions with the ground truth, a post-processing step was introduced, inspired by Böck et al. I perform peak picking (see peak_picking function) based on the predictions of our model using the find_peaks function from scipy library. The main parameter is distance (fixed to 30), because according to the documentation and the audio labs article (https://www.audiolabs-erlangen.de/resources/MIR/FMP/C6/C6S1_PeakPicking.html), this parameter is best for beat tracking to ensure that beats are some frames apart.

The metrics used for evaluation were the loss, precision, recall and f1 score. I created custom functions for each of the 3 scores, as I needed to well define the True Positives, False Positives and False Negatives.

The final model does not perform well in the Ballroom dataset. Here are some values of the metrics that we get:

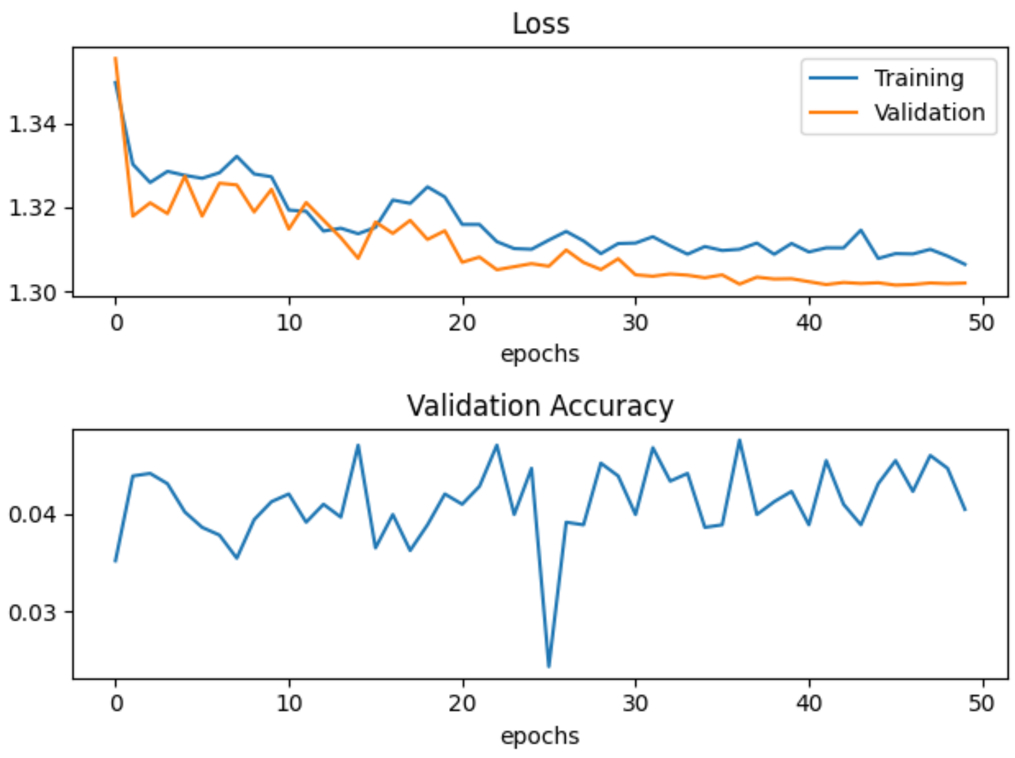
Precision: 6.43%
Recall: 3.78%
F1 score: 4.76%

As we can see, all three scores are really low, even after 50 iterations, which means that the model is incapable to learn the features and the patterns of the dataset. We tried to plot one song's waveform together with the beats from the dataset to compare with the ground truth:



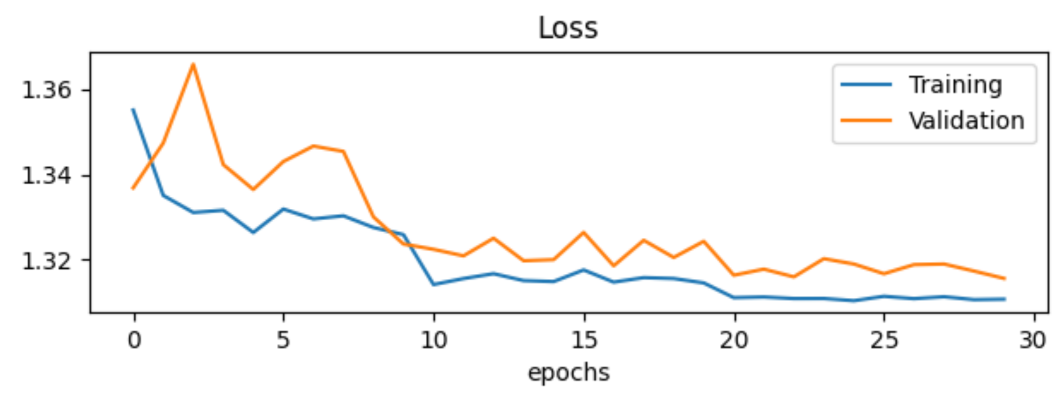
We observe that the detection occurs only for the first 21 seconds, which might be due to the CNNs focusing on enhancing the predictions in the first frames. In addition, we can see that the beats are not distinct and totally evenly spaced, which is totally reasonable given the low evaluation scores.

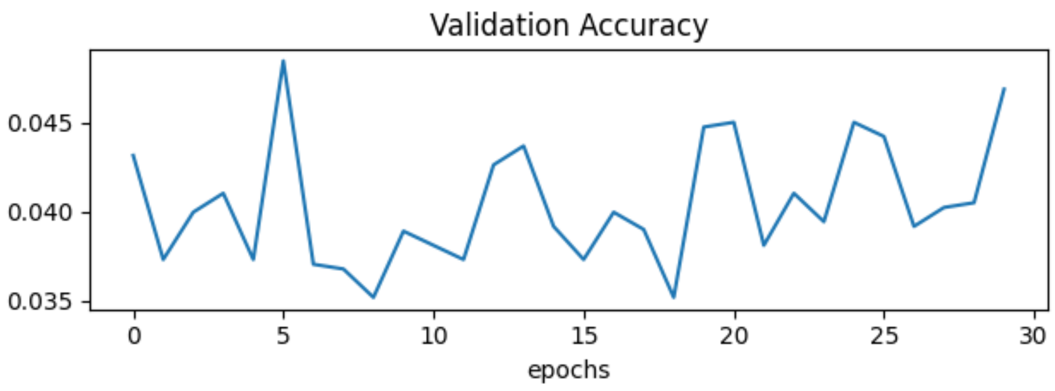
Plotting the results in loss function and accuracy, we can also notice that the model clearly underfits. However the loss decreases in both validation and training sets, which indicates that the model is learning. The accuracy has lots of fluctuations and extremely small values, proving the underfitting.

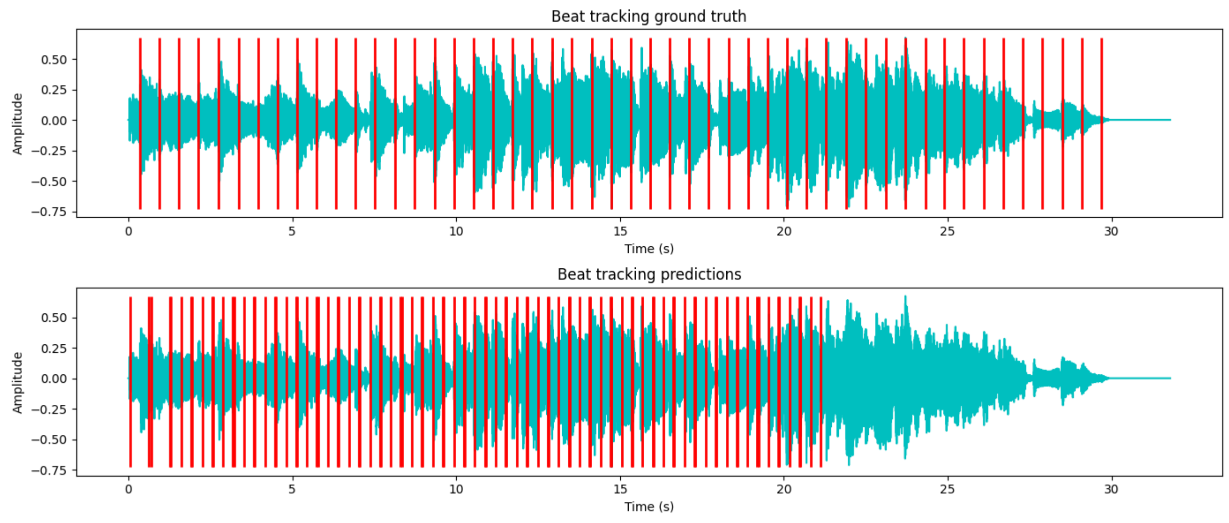


It is important to mention here that I experimented with multiple combinations of parameters for the architectures described above, especially focusing on changing the batch size, the learning rate, the parameters in the find_peaks function, adding more (or fewer) linear layers to change the dimensions and changing the number of layers in the BLSTM.

A satisfactory result of a 3-layer BLSTM (together with the CNN as the proposed architecture) can be seen in the figures below:







We notice that both validation and training loss is decreasing and accuracy keeps increasing. This aligns with Böck et al. approach, in which they used a BLSTM with 3 layers.

For testing, I have created a new file test_model.py which includes the specified beatTracker function. In this function, I extract the Mel-spectrogram of the audio file (make sure you change the path to your preferred audio file) with the same function as in the training process and pass it as input in my pre-trained model “best_model.pth”. Make sure that the path of the saved model is correct in this function. The predictions are again passed through the pick peaking function and then for every binary prediction, I calculate the time step in seconds, converting the index of the frame to seconds.

Mir_eval library was not used for evaluation, however, I tried to use it in a specific song to check the metrics. You can find the corresponding code commented out in the main function of the test_model.py.

Conclusion

In this work, we explored the performance of Neural Networks for beat detection. Constraints such as long duration of the training, limited computational power (GPU resources) and noisy dataset (not steady tempo) made it difficult to find a good architecture that could work sufficiently in the Ballroom dataset. However, the multiple and different experiments helped in understanding how the NNs might respond to such MIR tasks. Personally, I would like to explore further in the future this approach and try to add Temporal Convolutional layers or even transformers.

Bibliography

Böck, Sebastian, and Markus Schedl. "Enhanced beat tracking with context aware neural networks." Proc. Int. Conf. Digital Audio Effects. 2011.

MatthewDavies, E. P., and Sebastian Böck. "Temporal convolutional networks for musical audio beat tracking." 2019 27th European Signal Processing Conference (EUSIPCO). IEEE, 2019.

S. Hershey et al., "CNN architectures for large-scale audio classification," 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), New Orleans, LA, USA, 2017, pp. 131-135, doi: 10.1109/ICASSP.2017.7952132.