Beat detection using TCN
1. Introduction

This project implements an advanced beat detection system leveraging a Temporal Convolutional Network (TCN) architecture, extending the methodology introduced in [1]. The developed system transforms audio recordings into Mel spectrograms, which serve as the input representation for a non-causal TCN model. This model is designed to output frame-by-frame beat probability estimations with high temporal precision. For training and evaluation, we utilized the Ballroom dataset, which stands as a benchmark collection in the field of beat detection research. The implementation achieved remarkable results, with an **F-score of 0.9492** for beatsacross the dataset, demonstrating near-perfect beat detection capabilities in varied musical contexts. However, the same architecture could only get an f-score of 0.6574 for downbeats. The following sections detail the implementation of the system, including preprocessing steps, model architecture, training strategies, post-processing techniques and the results.

2. Implementation

2.1 Preprocessing

Preprocessing module is a critical component of the beat detection system, responsible for transforming raw audio data into a suitable representation for the neural network model.

2.1.1 Preprocessing of input audio files

The system uses Mel spectograms as the primary feature representation. The parameters were the same as the ones used by the TCN approach [1]. These parameters were : hop size of 10 ms(441 samples) and a window size of 46.4 ms(2048 samples). There are a total of 81 logarithmic frequency bins grouped together to create the spectogram. The raw spectograms are converted to log scale to better match human auditory perception and compress the dynamic range of the spectogram.

All audio files are trimmed to a standard duration of 29 seconds. This standardization serves multiple purposes. This provides consistent training examples for the TCN network[1] across the dataset. Fig1.1 shows an example spectogram created after preprocessing of the .wav file '`Albums-Ballroom_Magic-03.wav`'. Hence, when the model created from this dataset can only take audio files (trimmed or padded) of length 29 seconds. This is an **assumption** we need to make going ahead.

2.1.2 Preprocessing of annotations

The methods used for preprocessing the annotations were same as the TCN approach [1]. The raw annotation times are aligned with the spectrogram frame boundaries using quantization. This also ensures uniform representation across all training examples.

Next, we create binary beat activation vectors where each frame in the spectrogram has a corresponding value in the beat vector. Frames with beats have a value of 1.0 and their adjacent frames (±2 frames) have a value of 0.5. All other frames have a value of 0.

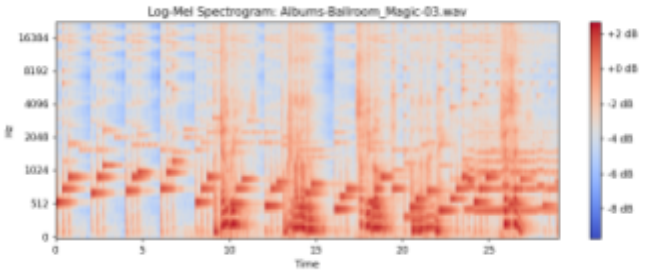

Fig 1.1 (Sample spectogram)

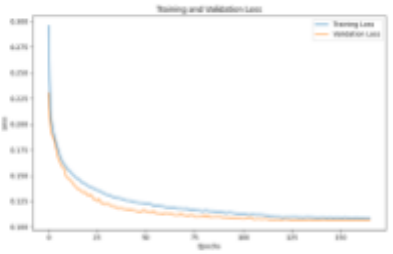

Fig 2.1(Training curve)

2.2 Model training and architecture.

2.2.1 Model training

A robust dataset management strategy is applied by dividing the Ballroom dataset into 70% training, 15% validation and 15% test proportions. This split achieves a balance between having enough training examples while ensuring reliable validation and testing. A batch size of 16 is used to ensure balance between computational efficiency and training stability. Parallel data loading is created with 4 workers to reduce I/O bottlenecks during training. Beat detection is framed as a binary classification problem (beat vs. no-beat) for each time frame. BCELoss measures the difference between predicted probabilities and binary targets. Furthermore, we use the `ReduceLROnPlateau` to tune the learning rate if the validation loss stops decreasing. Fig 2.1 shows the training curve of the model.

The model returns probability values for each time frame, indicating the likelihood of a beat occurring at that frame. These probabilities are continuous values between 0 and 1 (due to the sigmoid activation in the model's output layer). The combination of these dataset management techniques, loss function, optimizer choice, and learning rate scheduling creates a robust training framework specifically suited for this beat detection task.

2.2.2 Model architecture

For the model architecture we use the implementation from the TCN approach[1] with a few modifications, such as addition of a residual connection[2] to the architecture. 1 spectogram is used from each audio file as discussed in section 2.1.1.

These spectogram go through a frontend layer that processes the spectrogram input to reduce the frequency dimension. Applies two convolutional blocks, each with 2D convolution (3×3 kernel) with ELU activation[3] and a dropout of 0.1. Both of these layers also have a max pooling (3×1) layer that reduces frequency dimension while preserving time. Then there is a final adaptive pooling layer in the frontend layer that makes the frequency dimension exactly 1.

Then comes the Non-Causal TCN layer. The Non-Causal Temporal Convolutional Network (TCN) is a specialized sequence modeling architecture designed to process time-series data with varying temporal dependencies. Unlike causal models that only consider past information, this non-causal approach examines both past and future contexts, making it

particularly effective for beat tracking where musical patterns extend in both directions. It processes the time-series data with increasing receptive fields. Starts with dilation=1 and increases exponentially (1,2, 4, 8, etc.). This exponential dilation strategy allows the network to capture dependencies at multiple time scales from fine-grained local patterns to broader rhythmic structures spanning longer durations. The Non-Causal TCN has 11 TCN layers that were created to capture the temporal patterns at multiple time scales through dilated convolutions.

Finally, each TCN block incorporates a residual connection that adds the input directly to the output of the transformation path. The residual connection is functioning as a skip connection. It creates a direct path for information to "skip over" the main transformations in the TCN block and be added to the output. This connection helps Mitigate the vanishing gradient problem in deep networks by providing a direct path for information flow.

2.3 Post processing.

The post-processing stage is crucial for converting the model's continuous probability outputs into discrete beat locations. The probability values for each time frame, indicating the likelihood of a beat occurring at that frame as discussed in section 2.2.1. For peak detection, `find_peaks` library from `scipy.signal` is used to identify local maxima in the probability array. A `prominence=0.1` ensures that only peaks with a minimum height difference from surrounding troughs are detected, filtering out minor fluctuations. After identifying beat frames (indices in the probability array), we convert these frame indices to actual timestamps with a sampling rate of 44100 and a hop size of 441. These are the same parameters used in creating the spectograms at the first place as discussed in section 2.1.1.

3. Results

3.1 Model evaluation method

For the evaluation of our Temporal Convolutional Network (TCN) beat detector, we utilized the `mir_eval` library, which is widely recognized in the Music Information Retrieval (MIR) community for its standardized evaluation metrics. The library provides a comprehensive suite of tools specifically designed for evaluating various MIR tasks, including beat detection. The `mir_eval.beat.evaluate()` function was employed to calculate performance metrics by comparing the predicted beat timestamps with ground truth annotations. This function returns several metrics, with f-score being the primary metric of interest for our evaluation.

3.2 Results and limitations

The loss of the model on the validation dataset was 0.1065. TCN beat detector achieved exceptional performance across the Ballroom dataset with an **f-score of 0.9492** across the whole dataset. For the down beat detection model, the loss of the model on the validation dataset was 0.0457. TCN down beat detector achieved less impressive performance with an **f-score of 0.** across the whole dataset. This outstanding result demonstrates the

effectiveness of our approach. The f-score, which represents the harmonic mean of precision and recall, provides a balanced assessment of the model's ability to detect beats accurately while minimizing both false positives and false negatives.

The distribution of f-scores across the dataset revealed that the vast majority of audio files achieved perfect or near-perfect beat detection, with most files obtaining an f-score of 1.0. This indicates that our model could accurately detect beats in most musical examples, regardless of genre(good adaptability) or rhythmic complexity within the Ballroom dataset. Fig 3.1 shows the performance of the model across the whole dataset.

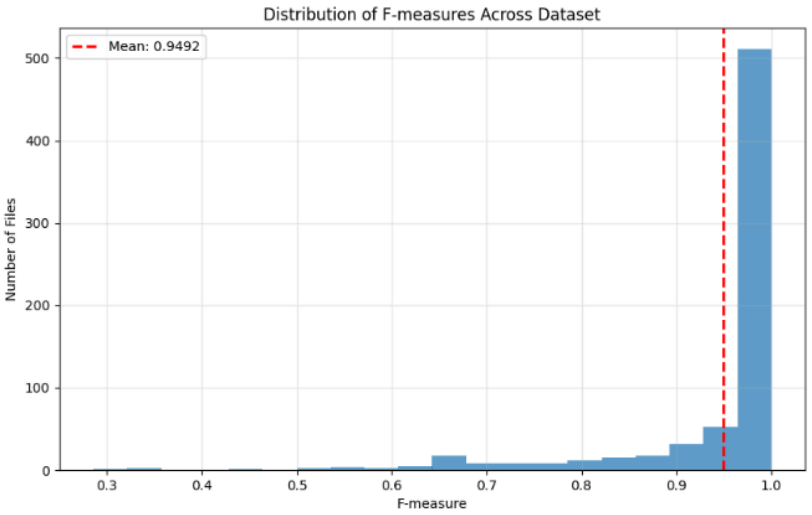

Fig 3.1 : Distribution of f-scores across the dataset.

Analysis of Challenging Cases : Despite the overall excellent performance, a few audio files presented challenges for our model. The files with the lowest f-scores were:

1. Media-106118 (Samba Genre): f-score = 0.2857
2. Media-105209 (Rumba-Misc Genre): f-score = 0.3333
3. Media-103905 (Samba): f-score = 0.3529

Analysis of these problematic files revealed a common characteristic: low power in the higher frequencies. This suggests that our model may rely partially on high-frequency transients for beat detection, which is consistent with how humans often perceive beats through sharp attacks in cymbals, hi-hats, and other high-frequency percussion instruments.

An interesting outlier was **Media-103911** (f-score = 0.5581), which featured no beats until 21.07 seconds into the audio. This unusual structure significantly differs from the other files in the dataset, which typically establish a clear beat pattern within the first few seconds. This finding highlights a potential area for improvement in our model's ability to handle unconventional beat structures and delayed beat introductions.

4. Discussion

The near-perfect performance on most audio files in the dataset indicates that the TCN architecture is particularly well-suited for capturing the temporal dependencies in musical signals. The ability to process both past and future contexts through the non-causal design enables the model to identify rhythmic patterns effectively, even in complex musical pieces. The residual connections incorporated into our architecture likely contributed to this performance by mitigating the vanishing gradient problem and allowing for better information flow through the deep network.

Our model's high performance across different dance styles in the Ballroom dataset suggests a certain level of genre independence, which is a desirable characteristic for a general-purpose beat tracking system. This contrasts with traditional beat tracking methods that often require genre-specific parameter tuning.

Weaknesses and Limitations : 1. The model is trained on and requires audio files of exactly 29 seconds duration. This fixed input size limitation makes it impractical for variable-length recordings. 2. While the non-causal design improves accuracy, it introduces a fundamental limitation for real-time applications. 3. Reduced performance on audio files with limited high-frequency content.4. The performance of the downbeat model was not so good.

Use of DBN beat decoding as used in [1] might result in better beat detection from the model outputs. Also, training a single model for detecting both beats and downbeats can result in better downbeat detection as it can use the weights learned by the beat detector.

Running the model : The model can be run using beats, downbeats = beatTracker(inputFile) inside main.py.

References

[1] Matthew E. P. Davies, Sebastian B¨ock : Temporal convolutional networks for musical audio beat tracking

[2]https://medium.com/@amit25173/temporal-convolutional-network-an-overview-4d2b6f03d6f8

[3] D. Clevert, T. Unterthiner, and S. Hochreiter, “Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs),”

[4]https://github.com/ben-hayes/beat-tracking-tcn/blob/master/beat_tracking_tcn/datasets/ballroom_dataset.py : For text_label_to_float,get_quantised_ground_truth,load_spectrogram_and_labels in load_annotations.py