STATS 402 - Interdisciplinary Data Analysis AI Charting for Music Game Cytoid Milestone Report: Stage 2

Yuchen Song
Duke Kunshan University Email: ys396@duke.edu

***Abstract*—The project has made significant progress in data preprocessing, with a structured pipeline now in place for tasks such as dataset preparation, model design, and initial training. This report provides a detailed overview of the dataset, the model architecture, and the preliminary experiments conducted during the initial stages of training.**

I. Current Status of Project

The above is a description of what I have implemented with the python code.

A. Raw Dataset

The dataset[8] used in this project consists of high-quality audio recordings coupled with detailed chart data annotations. I temporarily use the data under "Z" folder. Specifically, it includes 127 songs spanning diverse genres such as pop, rock, classical, and jazz, with each track provided in WAV format at a sampling rate of 44.1 kHz. Alongside the audio, detailed annotations describe the note onset times, their types (e.g., tap, hold, slide), and their positions on the scanning line. These annotations were manually labeled to ensure high accuracy and reliability. To support model training and evaluation, the dataset is divided into three subsets: 80% songs for training, 20% songs for validation as all the previous work has taken [4], [5], [6].

B. Data Preprocessing

1) Audio Data Preprocessing for Model Input: To prepare the raw audio and chart data for training, a preprocessing pipeline was established. This pipeline transforms the raw data of audio file and Cytoid-compatible json chart into a matrix and vectors that are compatible with the models, including techniques such as the Short-Time Fourier Transform (STFT) and one-hot encoding.

First, each audio file is converted into a Mel spectrogram representation. The spectrograms are generated using a window size of 32 ms and a hop size of 23 ms, resulting in overlapping audio segments for better frequency resolution. The Mel spectrograms are computed using 128 Mel bands, which capture the critical frequency nuances of the audio signals. STFT is applied during this process to segment the audio into overlapping frames, while the Mel scale enhances frequency representation, as suggested in [5].

To fully use the temporal context and information, each spectrogram frame is augmented with a sliding window that includes five preceding and five following frames. This creates a final feature size of $128 \times (2 \times 5 + 1) = 1408$ per frame, embedding approximately 115 ms of surrounding audio context. The spectrograms are then normalized to have zero mean and unit variance, which ensures stable training of the neural network.

2) Chart Data Processing for Model Output: The chart json file are processed to generate labels for each spectrogram frame. These label vectors encompass three types of information: binary indicators for note presence, categorical labels for note type (e.g., tap, hold, slide, with eight distinct classes encoded as one-hot encoding), and continuous values representing the note's position on the scanning line (scaled between 0.0 and 1.0). This alignment ensures that every spectrogram frame corresponds to a well-defined set of labels.

3) Represenation: Each input sample x^i corresponds to a specific frame in the Mel spectrogram, including the its surrounding 10 frames' mel representation. Its corresponding label y^i encapsulates information about the presence, type, and position of any note occurring within that frame. This precise temporal alignment is critical for effective model training. Let me provide a coherent and descriptive explanation of your model structure and associated concepts in natural language, while incorporating the details you've shared.

C. Model Structure

The proposed model is a Bidirectional Long Short-Term Memory (BiLSTM) network designed for multi-task learning, including predicting the presence, type, and position of notes in audio signals. The core architecture of the model is built upon sequential layers and specialized output heads.

1) BiLSTM Layer: The backbone of the model is a BiLSTM[7], which processes the input sequence both forwards and backwards in time. This bidirectional design allows the network to capture dependencies across time, improving its ability to detect musical events. Parameters are as follows: input_size $= 1408$, hidden_size $= 128$, num_layers $= 2$, bidirectionality is enabled, resulting in an output dimension of $128 \times 2 = 256$ for each time step. Dropout regularization is applied between LSTM layers to reduce overfitting, with a default value of 0.5.

2) Fully Connected Output Heads: After the LSTM processes the input sequence, the output at each timestep is passed through three fully connected layers, each dedicated to

a specific prediction task: (1) Presence Prediction: Determines whether a note is present in the current frame. The output layer consists of a single unit with a Sigmoid activation function, producing a probability score between 0 and 1 for binary classification. (2) Type Prediction: Identifies the type of note (e.g., tap, hold, slide). The output layer consists of 8 units (one for each class) with a Softmax activation function, which outputs a probability distribution over the possible classes. (3) Position Prediction: Predicts the position of the note on the scanning line as a continuous value. The output layer consists of a single unit with a Tanh activation function, mapping the prediction to the range $[-1, 1]$. The value will be processed with the normalization, which will produce a $[0, 1]$ position.

3) Loss Functions: To train the model effectively, task-specific loss functions are combined into a total loss. This loss is calculated across all batches (B) and time steps (T), ensuring accurate binary classification of whether a note exists at each frame. Each component loss is described below:

(1) Presence Loss The Presence Loss uses Binary Cross-Entropy (BCE) to measure the difference between the predicted probabilities ($\hat{y}_{it}$) and the true labels (y_{it}) for the presence of notes. The formula is: $\mathcal{L}_{\text{presence}} = -\frac{1}{B \cdot T} \sum_{i=1}^{B} \sum_{t=1}^{T} [y_{it} \log(\hat{y}_{it}) + (1 - y_{it}) \log(1 - \hat{y}_{it})]$

(2) Type Loss The Type Loss use Cross-Entropy Loss, a common metric for multi-class classification. This loss evaluates the predicted probabilities ($\hat{y}_{itc}$) against the true labels (y_{itc}) for different note categories (N). The formula is: $\mathcal{L}_{\text{type}} = -\frac{1}{B \cdot T} \sum_{i=1}^{B} \sum_{t=1}^{T} \sum_{c=1}^{N} y_{itc} \log(\hat{y}_{itc})$

(3) Position Loss The Position Loss uses Mean Squared Error (MSE) to calculate the squared difference between the predicted ($\hat{y}_{it}$) and true (y_{it}) positions of notes. This loss is crucial for accurately predicting the continuous position values of notes within the scanning line, across all batches (B) and time steps (T). The formula is: $\mathcal{L}_{\text{position}} = \frac{1}{B \cdot T} \sum_{i=1}^{B} \sum_{t=1}^{T} (y_{it} - \hat{y}_{it})^2$

The Total Loss is a weighted combination of the three individual losses . Each task is assigned a weight (λ_1, $\lambda_2$to balance their contributions during model training. The formula is: $\mathcal{L}_{\text{total}} = \lambda_1 \mathcal{L}_{\text{presence}} + \lambda_2 \mathcal{L}_{\text{type}} + \mathcal{L}_{\text{position}}$ Currently, I simply assigned $\lambda_1 = \lambda_2 = 1$ By tuning the weights, different priorities can be assigned to each task, depending on their importance in the overall performance of the model.

II. Initial Results

A. Presence and Note Type

The implemented model produces predicted probabilities for both the presence of notes and their respective types. The visualization process involves the following procedure: First, the model determines the presence of a note in each time frame using the presence prediction output. This probability is then compared against a manually predefined threshold (for example, 0.25). If the presence probability exceeds the threshold, the model proceeds to identify the specific type of the note by selecting the class with the highest probability from the type prediction outputs. Consequently, the most likely note type is assigned to that frame. For frames where the presence probability does not meet the threshold, no note type is assigned.

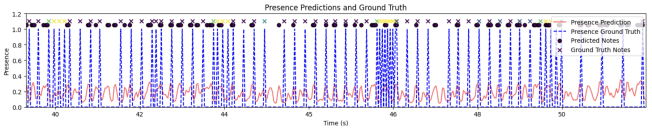

Fig. 1. Predicted Presence and Note Type

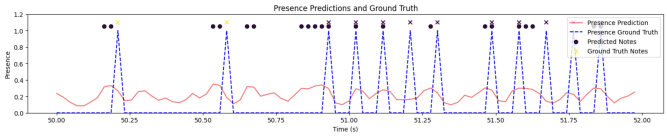

Fig. 2. A Detailed Predicted Presence and Note Type

Figure 1 visualizes the model's predictions compared to the ground truth data over a specified time interval. The x-axis represents time in seconds. The red line illustrates the model's predicted presence of notes, indicating the likelihood of a note occurring at each moment in time. The blue dashed line represents the actual presence of notes as recorded in the ground truth labels.

Above these lines, dark circles mark the positions where the model has predicted notes to occur, with different colors indicating different note types. Similarly, crosses indicate the true positions and types of notes as per the ground truth data. It can be observed that the model accurately predicts the presence of notes. However, the prediction of note types requires further improvement.

Figure 2 is a magnified version of 1, which can be more easily observed.

B. Note Type Problem

As shown in the confusion matrix in Figure 3, there is a significant tendency for the model to predict the Note Type as 0. This bias could result from the uneven distribution of note types in the training data or potential errors in the implementation. Further investigation and adjustments are necessary to address this issue and improve the accuracy of note type predictions.

Fig. 3. Confusion Matrix for Note Types

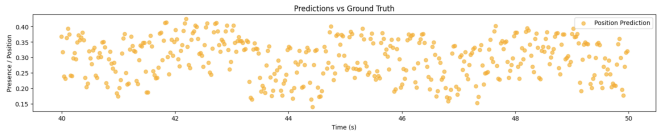

Fig. 4. Note Position Distribution

C. Note Position Distribution Problem

Figure 4 illustrates the distribution of predicted note positions. Due to the nature of the loss function and the predominance of zeros in the position data, the model's predictions tend to be confined within the range of 0 to 0.4. This limitation suggests that the model struggles to accurately predict positions with higher values, indicating a need for further refinement of the loss function or the modeling approach to better capture the full range of position values.

III. Future Works

Building upon the initial results, several avenues for future work emerge to enhance the model's performance in predicting note types and positions.

Firstly, refining the weighting scheme of the total loss function by tuning the parameters λ_1 and λ_2 can help balance the contributions of presence, type, and position predictions more effectively. This adjustment could lead to improved overall model performance by emphasizing tasks that are currently underperforming. Additionally, fine-tuning the model parameters is necessary to validate these improvements.

Secondly, instead of applying the Gated Multimodal Unit (GMU)[1] method as initially proposed, a more straightforward approach for selecting the adoption threshold should be utilized. Currently, the threshold is manually selected to evaluate the results. An algorithm that automatically determines the threshold based on the input difficulty level should be added.

Thirdly, addressing the imbalance in note type distribution is essential. This involves debugging the code and thoroughly examining the data distribution of note types to identify and rectify any underlying issues that may be causing biased predictions. Techniques such as data augmentation, resampling, or implementing more sophisticated loss functions that account for class imbalance could be explored to mitigate this problem.

Fourthly, enhancing the model's ability to predict note positions is crucial. One approach could involve testing a separate LSTM network with input x_i indicating the position of the ith note to improve position accuracy. Alternatively, developing a non-machine learning, random-based algorithm might ensure more precise placement of notes, providing a complementary method to validate the model's predictions.

Lastly, converting the predicted chart into a Cytoid-compatible[2] JSON format is necessary for practical application. Writing a conversion script and conducting tests within the game environment will ensure that the predictions are accurately integrated and function as intended. This step is vital for evaluating the real-world applicability of the model and gathering user feedback for further refinements.

References

[1] J. Arevalo, T. Solorio, M. Montes-y-Gómez, and F. A. González, "Gated multimodal units for information fusion," *arXiv preprint arXiv:1702.01992*, 2017.

[2] Cytoid Wiki, "C2 Format," https://cytoid.wiki/en/reference/chart/c2-format. [Accessed: Nov. 6, 2024].

[3] Cytoid. https://cytoid.io. [Accessed: Nov. 6, 2024].

[4] D. Donahue, K. Simonyan, A. Zisserman, and G. Vondrick, "Dance Dance Convolution: Learning Generative Models for Dance," *Proc. IEEE Int. Conf. on Computer Vision*, 2017, pp. 1-9.

[5] Y. Ye, S. Huang, and L. Wang, "TaikoNation: A Neural Approach to Rhythm Game Chart Generation," *Proc. IEEE Int. Conf. on Machine Learning*, 2020, pp. 1234-1243.

[6] A. Takada, D. Yamazaki, Y. Yoshida, N. Ganbat, T. Shimotomai, N. Hamada, L. Liu, T. Yamamoto, and D. Sakurai, "Genélive! Generating rhythm actions in Love Live!," in *Proc. AAAI Conf. on Artificial Intelligence*, vol. 37, no. 4, pp. 5266–5275, 2023.

[7] S. Siami-Namini, N. Tavakoli, and A. S. Namin, "The performance of LSTM and BiLSTM in forecasting time series," in *Proc. 2019 IEEE Int. Conf. on Big Data (Big Data)*, pp. 3285–3292, 2019.

[8] Google Drive Folder, *Shared Resource Folder*, https://drive.google.com/drive/folders/1J43x9f8u2lIzaHBolQaZveCv62XQM8Lv. [Accessed: Nov. 23, 2024].