# STATS 402 - Interdisciplinary Data Analysis AI Charting for Music Game Cytoid Milestone Report: Stage 3

Yuchen Song
Email: ys396@duke.edu

This project aims to develop an model for generating charts for the music game Cytoid. Throughout the development process, I have explored various modeling techniques and made several adjustments to our technical approach based on the performance outcomes observed in previous stages.

## I. Current Status: All the failed Attempts

In this section, I will discuss the current status of my model. After numerous trials and errors, I decided to transfer my basic model from Bidirectional Long Short Term Memory (BiLSTM) to Convolutional Neural Network(CNN), which was primarily determined by the prediction accuracy. I will explore all the attempts to improve, most of them failed and did not have a good effect, in Section 2.

Currently, my entire model operates as a pipeline structure consisting of two CNN models: one for note presence and another for note type prediction, along with a BiLSTM model for note position prediction. Due to time constraints, I have chosen not to consider difficulty levels; instead, all training data used are of difficulty levels 15 and 16. I believe these levels offer greater flexibility and more unified features.

Figure 1 is a brief pipeline structure of the current model.

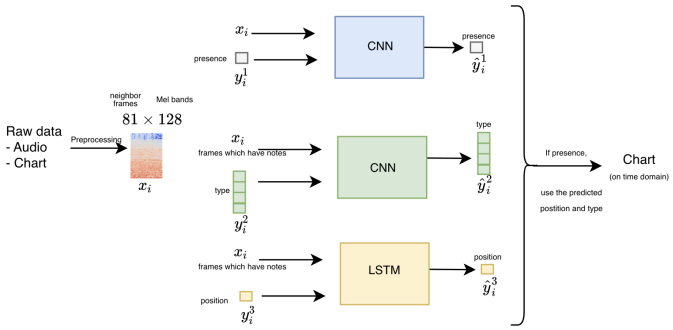


Fig. 1. A Flow Chart of Current pipeline Structure: CNN for Presence applies all frame representation as input and output whether a frame have note presence. The CNN for Type and LSTM for position only use frames which have notes as input to predict note type and position.

### A. CNN model for note presence prediction

In initial experiments, I found that a simple CNN model achieved a final F1 score of approximately 70% on the note presence prediction task. Based on this result, I decided to adopt CNN models for both note presence and type prediction. Below is a detailed description of the CNN model implementation: (1)Input: the audio signal is transformed using Short-Time Fourier Transform (STFT) into a Mel spectrogram. To capture temporal context, the model uses the current frame along with the preceding and following 40 frames of the Mel spectrogram as input, totaling 81 frames. Each frame consists of 128 Mel frequency bands, resulting in an input shape of $(81, 128)$. (2) Output: This value is passed through a sigmoid activation function to produce a probability score between 0 and 1, indicating the likelihood of a note being present at the central frame of the window. The corresponding label is set to 1 if a note is present and 0 otherwise. (3) Model Structure: The basic structure of the CNN contains 3 layers with each layer contains a convolution network, a batch normalization, a ReLU activation function, and a max-pooling layer. (4) Loss funciton: The model uses the Binary Cross-Entropy (BCE) with Logits Loss as the loss function. This loss function is suitable for binary classification tasks and combines a sigmoid layer with the binary cross-entropy loss. (5) Evaluation: For evaluation, the F1 score is primarily used to measure the model's performance.

### B. CNN model for note type presence prediction

Building upon the success of the CNN model for note presence prediction, I extended the architecture to predict the type of each note. This model shares a similar structure with the presence prediction model but is tailored for multi-class classification.(1) Input: Similar to the presence prediction model, the input consists of 81 frames of the Mel spectrogram, each with 128 Mel frequency bands. This temporal context helps the model discern subtle differences in the spectrogram that correspond to different note types. (2) Output: Instead of a single probability score, the model outputs a probability distribution across five different note types. A softmax activation function is applied to produce probabilities that sum to 1, indicating the likelihood of each note type. (3) Model Structure: The CNN architecture also consists of three convolutional layers, each followed by batch normalization, ReLU activation, and max-pooling. After the convolutional layers, the feature maps are flattened and passed through fully connected layers with dropout regularization to prevent overfitting. The final layer uses a softmax activation function to output probabilities for each note type. (4) Loss Function: Cross-Entropy Loss

is used as the loss function for the multi-class classification task. This loss function measures the discrepancy between the predicted probability distribution and the true distribution. (5) Evaluation: The model's performance is evaluated using accuracy, precision, recall, and F1 score.

### C. LSTM model for presence

Predicting the precise position of notes within a song requires understanding temporal dependencies. To address this, I integrated an LSTM (Long Short-Term Memory) model into the pipeline for note position prediction. (1) Input: The CNNs process a $81 * 126$ Mel spectrogram to generate feature maps, which are then fed into the LSTM to capture temporal patterns associated with note positions. So the input to the LSTM model consists of features extracted by the CNN models for note presence and type prediction. (2) Output: The LSTM model outputs continuous values representing the precise timing of each note within the song. These predictions aim to accurately place notes in synchronization with the music. (3) Model Structure: The LSTM architecture comprises two layers with a hidden size of 128 units each. Following the LSTM layers, a fully connected layer maps the LSTM outputs to the final position predictions. (4) Loss Function: Mean Squared Error (MSE) Loss is utilized to measure the discrepancy between the predicted note positions and the true positions. This loss function is appropriate for regression tasks where the goal is to minimize the squared differences between predicted and actual values. (5) Evaluation: The model's performance is assessed using MSE and R-squared ($R^2$) metrics.

### D. Current implement status

Currently I have integrated the Note Presence CNN model and Note Type CNN model, and above is the results. The models were evaluated on a validation dataset comprising 81 songs [6]. The performance metrics for both Presence and Note Type prediction tasks are summarized in Table I.

TABLE I
PERFORMANCE METRICS FOR PRESENCE AND NOTE TYPE PREDICTION

| Metric | Accuracy | Precision | Recall | F1 Score |
|---|---|---|---|---|
| **Presence** | 90.39% | 86.28% | 51.66% | 64.63% |
| **Note Type** | 92.19% | 86.13% | 92.19% | 89.06% |

Figure 2 demonstrate the prediction of a song segment, with the red plot indicating the note presence probability. A frame has note presence if the probability is greater than the threshold of 0.5. The green line represents the frames where there is ground truth presence of a note. Different colors of dotted lines indicate different predicted types. Different colors of the straight line indicates predicted notes.

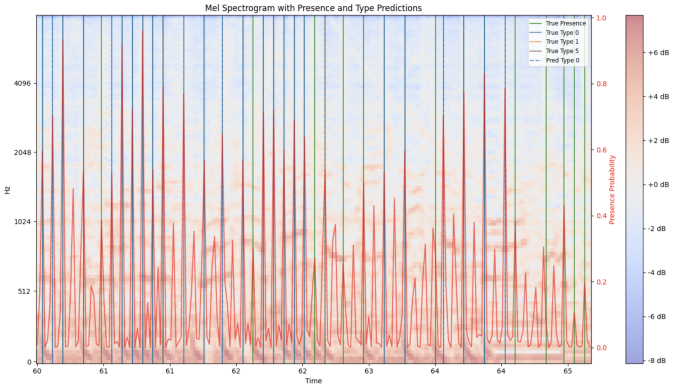


Fig. 2. CNN Note Prediction with Presence and Type

However, it can be seen from the confusion matrix from Figure 3 that the note type prediction is not very accurate. This might be due to class imbalance. I plan to work on improving this in the upcoming days. Additionally, the LSTM model for position prediction has not yet been added to the pipeline.

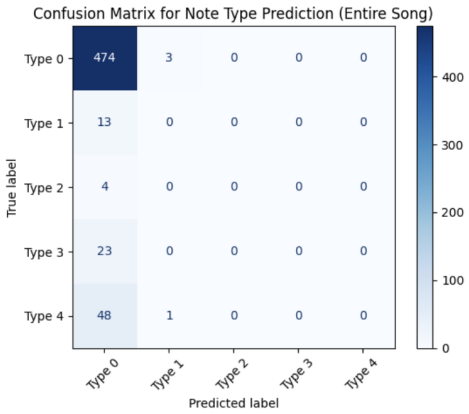


Fig. 3. Confusion matrix of predicted note type of a certain song

## II. ALL THE EXISTING ATTEMPTS

Below is all the attempts, that has been failed to achieve good performance, I have made to improve the Note presence model.

### A. Presence Models Attempts

In our Stage 2 report, I proposed an integrated model utilizing Bidirectional Long Short-Term Memory (BiLSTM) networks to predict note presence, type, and placement simultaneously. The model shared the same set of parameters for these predictions but employed different output layers for each task. Unfortunately, the performance of this approach was under expectation as stated those in other paper [5], [4], [3], [2]. The presence prediction component, in particular, struggled to achieve accurate results Because of a pipeline structure, the type prediction is also hard to improve. The model was designed to take inputs consisting of 10 frames surrounding the target frame. The output layer applied a

sigmoid activation function, producing probability values between 0 and 1 for note presence at each frame. Despite fine-tuning the hyperparameters, the predicted probabilities were predominantly clustered between 0.2 and 0.4, lacking significant distinction. Initially, I observed local peaks that suggested potential good performance, but after applying thresholding, the results were disappointing. Even when using an optimal threshold determined by enumerating all possible values to maximize the F1 score, the performance metrics remained low, with F1 scores hovering around 0.1 to 0.3. Figure 4 is a visual example of the predicted note presence (red plotting line) in contrast with the ground truth presence (blue dots). Although there is local peak when there is ground truth notes, the difference between notes and non-notes are not distinct that it cannot be used to predict note presence by applying a threshold.

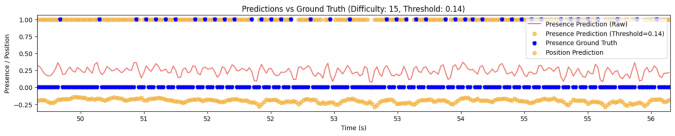


Fig. 4. BiLSTM model for note presence prediction

To address these issues, I seperately tried several modifications to the model architecture and data processing pipeline:

*a) Single Frame Input:* The first adjustment involved changing the input from a window of 11 frames to a single frame. The rationale behind this was that the sliding window mechanism introduced significant variability, as each slide involved shifting the window by one frame. Although adjacent frames are similar, the input data differed entirely, which likely confused the model. By using only the current frame as input, the model could focus on more stable and consistent features. However, this modification did not yield significant improvements, with F1 scores still capped around 30%.

*b) Data Limitation by Difficulty Level:* My second adjustment was to limit the dataset to songs with a difficulty level of 15 or higher. My hypothesis was that higher difficulty levels have more flexible and denser note placements, along with a higher frequency of rare note types beyond simple clicks. By focusing on these challenging tracks, the model could potentially learn more nuanced patterns, leading to better performance. This filtering aimed to enhance the quality of the training data by ensuring that the model was trained on more complex and varied note distributions. Figure 5 is a visual example of the predicted note presence (red plotting line) in contrast with the ground truth presence (blue plotting line). The yellow dot indicates the predicted note presence while the blue dot is the ground truth presence. It can be derived that the predict performance is bad with no distinct flutuation in predicted probability. It is nearly impossible to choose the threshold.

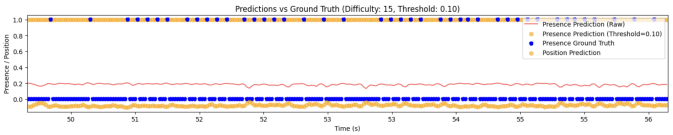


Fig. 5. BiLSTM model for note presence prediction with input of diffculty level no smaller than 15

*c) Threshold Selection Methods:* There are two main approaches to adjusting the threshold for different difficulty levels. (1) The first method involves calculating the average threshold for each difficulty level across all songs and then applying these thresholds to individual songs. This approach ensures that each difficulty level has a tailored sensitivity, which can help the model better handle the variations in complexity between different tracks. (2) The second method focuses on the distribution of notes within each difficulty level. For each difficulty level $i$, we calculate the average number of notes $n_i$. Then, we sort the note presence probabilities and select the top $n_i$ notes as the predicted points. This method aims to match the expected number of notes for each difficulty, potentially improving the accuracy of note detection.

However, when integrating these threshold selection methods with the LSTM models, the results were not as good as expected, with many notes appearing consecutively. To address this issue, I experimented with adding some constraints based on my experience with rhythm games. For example, I implemented a rule that allows at most one note to be selected within any three consecutive frames. Despite these efforts, the performance did not improve significantly because the LSTM model itself was not performing optimally. The poor training results of the LSTM meant that the additional restrictions did not have the desired effect. Due to time limit I decided to drop the consideration of difficulty level.

The two tables below summarize the results when using a 2-layer BiLSTM model with single-frame input for note prediction. This setup aimed to control the total number of predicted notes according to their respective difficulty levels. These results suggest that the model does not adequately address the underlying issues in the model's predictive capability.

TABLE II
THRESHOLD AND PREDICTION STATISTICS

| Difficulty | Threshold | Pred. Count | Target Count |
|---|---|---|---|
| 16 | 0.27 | 1536 | 1533.60 |
| 15 | 0.26 | 1150 | 1145.58 |

TABLE III
EVALUATION METRICS

| Difficulty | Accuracy | Precision | Recall | F1 |
|---|---|---|---|---|
| 16 | 0.8036 | 0.1739 | 0.0694 | 0.0992 |
| 15 | 0.8271 | 0.1616 | 0.0376 | 0.0610 |

*d) Exploring Alternative Models: Transformer and CNN:* The third significant change was exploring different model architectures beyond LSTM-based networks. Specifically, I experimented with Transformer and CNN models. The Transformer model, despite careful hyperparameter tuning, produced very smooth probability curves, indicating poor discrimination between note presence and absence, resulting in an F1 score around 10%. On the other hand, the CNN model demonstrated a surprising improvement, achieving F1 scores up to 70%. This stark contrast highlighted the potential of CNNs for this task, likely due to their ability to capture local temporal patterns more effectively than LSTMs or Transformers in this specific application.

*e) Transformation model detail:* Since I wasn't very familiar with the details of Transformer models, I decided to use an existing Transformer implementation I found online for predicting note presence. To optimize the model, I employed Optuna to tune 20 hyperparameters based on the loss function observed after training. Despite these efforts, the predictions were overly uniform and consistently low, making it impossible to set a meaningful threshold to distinguish when a note was present. Consequently, the F1 score hovered around nearly 0.

Specifically, I configured the Transformer model with the following parameters: an input size of 128 Mel bands, a hidden size of 448, 5 Transformer layers, 4 attention heads, a dropout rate of 0.2, trained for 46 epochs, a batch size of 1, and a very low learning rate of approximately 4.96e-5. Despite these settings, the results were disappointing. The model struggled to differentiate between frames with and without notes, leading to uniformly distributed predictions that failed to capture the presence of notes accurately.

Figure 6 is a visual example of the predicted note presence (red plotting line) in contrast with the ground truth presence (blue plotting line). It can be derived that the predict performance is bad with no distinct flutuation in predicted probability.

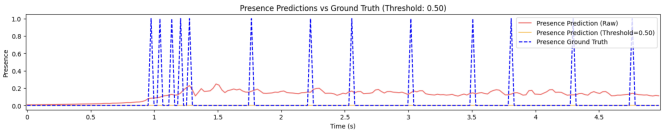


Fig. 6. Note Presence Prediction made by Transformer

## III. Plan for Finalizing Final Report

To complete the final report successfully, I will focus on several key areas over the next few weeks.

First, I plan to enhance the note type prediction by addressing the current class imbalance. This will involve techniques such as oversampling underrepresented classes.

Next, I will integrate the LSTM model for precise note position prediction into the existing pipeline. I will also validate the position predictions using appropriate metrics to ensure their reliability.

Another crucial task is converting the generated charts into JSON format compatible with Cytoid. I will carefully review Cytoid's JSON schema and develop a conversion pipeline that accurately maps the model's outputs to the required structure.

## References


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

[2] 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.

[3] 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.

[4] 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.

[5] 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.

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