# Music Genre and Composer Classification Using Deep Learning

Swathi Subramanyam Pabbathi, Omar Sagoo, Carlos A. Ortiz Montes De Oca

Shiley-Marcos School of Engineering, University of San Diego

AAI-511: Neural Networks and Deep Learning

Dr. Bilgenur Baloglu

## Author Note

This project was developed for the Neural Networks and Deep Learning (AAI-511) graduate course at the University of San Diego by aspiring Master's students of the MSAAI program under the tutelage of Dr. Bilgenur Baloglu. The used dataset "midi_classic_music" (Fedorak, 2019), was extracted from Kaggle.

The work uses Deep Learning models, particularly CNN, LSTM, and Bidirectional LSTMs+Attention architectures to classify music by composer. The project files and code repository can be found at: https://github.com/PSswathi/music-genre-and-composer-classification/ We have no conflicts of interest to disclose.

**Abstract**

This project explores the classification of classical music composers from MIDI files. We implemented and evaluated three deep learning models Long Short-Term Memory (LSTM), a Bidirectional LSTM with an Attention Mechanism , and Convolutional Neural Networks (CNN) to distinguish between composers based on statistical and temporal characteristics of the music. Experiments were conducted using Amazon SageMaker on a GPU-enabled ml.g4dn.xlarge instance with 20 GB of working space. Results indicate that the Bidirectional LSTM models outperformed the LSTM and CNN in classification accuracy and generalization on the test set.

Keywords: Deep Learning, Long Short-Term Memory (LSTM), Bidirectional LSTM(BiLSTM), Convolutional Neural Networks (CNN), Composer Classification, Music Analysis

## Music Genre and Composer Classification Using Deep Learning

Classical music compositions by different composers often carry unique stylistic signatures that allow both humans and ML models to correctly recognize the pieces that belong to a specific composer. The goal of this project was to leverage those patterns to build a deep learning model that can classify composers from their pieces, available in the form of MIDI files. MIDI files were preprocessed to extract features such as tempo, average pitch, pitch variance, chord density, and note durations. After extracting and structuring these features into tabular format, we developed Deep Learning models, LSTMs, BiLSTMs and CNNs to evaluate their effectiveness on this task. LSTMs were chosen because of their ability to capture temporal dependencies from sequentially-structured data (Olah, 2015), and CNNs because of their effectiveness in extracting distinct local features from image-like representations of data, in our case piano sequences in the form of arrays. Given the success of Attention-based architectures in the last couple of years, we decided to implement a model which harnessed the benefits of both attention mechanisms and the LSTMs temporal dependency capturing capabilities. We therefore chose to deploy BiLSTM models with an Attention layer as well. Music, being both sequential in nature and a complex phenomena usually stored in raw audio files or midi, hence benefits from analysis via Deep Learning models. We were able to achieve accuracies of 85.6% , 90.9%, and 77% on the LSTM, BiLSTM, and CNN models respectively. Strategies to improve the model performance, as well as more detailed interpretation of the results are discussed further down below.

## Methodology

The midi_classic_music dataset was used, which contains a collection of classical works by 175 composers in the form of midi files. This project was limited to the work of four specific composers: **Bach, Beethoven, Chopin, and Mozart.** The dataset consists of 1608 midi files spread across the four composers. Strategies were implemented to handle the different class imbalances

that surfaced during preprocessing, and are discussed later in the report. The data was handled differently for CNNs and for the LSTMs. For the CNNs we preprocessed the data by reshaping the midi files to arrays of piano roll sequences (piano roll approach). While for the LSTMs, musical features were extracted from the midi files. The midi files were afterwards split into chunks of 200 notes each to be used as inputs in the corresponding models. Each chunk kept its corresponding set of locally computed features. The Exploratory Data Analysis was conducted on the extracted features dataset before chunking.

**Dataset and Feature Engineering**

We used a collection of MIDI files categorized by composer and organized them into train, val, and test folders. Each MIDI file was parsed using the pretty_midi library, where the library was used differently depending on the data preprocessing approach. The extracted features approach led to a rich set of 18 midi-derived features. A brief description of each of the features can be found on Figure 1. Frequency encoding was applied to the most_common_pitch, and standard scaling was performed on all features. We addressed class imbalance using upsampling and downsampling techniques to ensure balanced training across composer categories. The piano roll approach led to piano roll sequences represented as one dimensional vectors of length 128. We addressed class imbalance by random sampling an arbitrary number of rows (15k per composer). We also ended up encoding the label categories.

**EDA Insights**

We Performed an Exploratory Data Analysis that involved: obtaining the dataset Summary Statistics, Feature Distribution plots (Figure 2-3), and Feature Correlation Matrix (Figure 4) for the extracted features dataset. From our analysis, we were able to establish the following meaningful relationships:

- **Pitch**: All composers use pitch ranges 40–80. Bach tends to favor slightly higher pitches.

- **Velocity:** Bach displays a consistent velocity spike around 90. Chopin's distribution is spread out on the lower range of velocity. Mozart's and Beethoven's distributions are a bit more centered.
- **Interval:** Chopin displays a centered larger spread. Mozart seems to be extremely concentrated, and Bach and Beethoven are concentrated as well. This could be interpreted as the propensity of the composer to display musical contrasts or dynamism when moving along the available octaves.
- **Tempo**: Distinct tempo spreads for each of the composers between 150-250 BPMs. Bach is spread out yet avoids the center, Beethoven and Chopin have large centered spikes, and Mozart is distributed close to the upper center range.
- **Chord and Note Density:** Mozart and Bach show strong chord density, characteristic of the monophonic/harmonic tendencies of their era. We see shifting more dispersed spreads for Beethoven and consequently Chopin, (given the evolution of classical music to incorporate less rigid nuances of harmony and adherence to monophony).

It is worth noting that, given the nature of DL models, the relationships we identified across the dataset features might not have been the same as the ones the DL model captured. They served merely as insights that influenced the data preprocessing and feature selection phase. Also given the imbalanced dataset we augmented and resampled the training data and compared the training data with the augmented data throughout the EDA. The comparisons can be seen on their corresponding figures. We ended up using the original training data to train our models.

**Preprocessing strategies**

We first split the MIDI files into Train, Test, and Validation sets (70/15/15 split). We went on to separately chunk each of these sets, as well as extract the midi related features to be used during

each stage of the ML lifecycle. The strategies that are employed, regarding data preprocessing and feature selection were:

- **Class Imbalance**: The train dataset is imbalanced. We plotted the feature distributions for the train_dataset and identified that Beethoven had a significantly larger amount of chunks than all other composers on the features extracted dataset, and Chopin had a smaller number of datapoints compared to the other composers on the piano roll approach. We therefore proceeded to implement the previously mentioned class imbalancing techniques for each approach.
- **Feature Standardization**: The EDA revealed wide variation in the scale of certain features such as tempo, duration, and note_density. Features were standardized using StandardScaler to ensure that all input variables contribute equally during model training (feature-extracted dataset).
- **Collinearity**: We identified strong collinearity between the Start and End features, and therefore went on to drop the End feature from the dataset (feature-extracted dataset).
- **Frequency/label Encoding**: We went on to encode the Program (n# instruments) feature and replace any NaN values that remained (feature-extracted dataset). Also encoded the label column for both approaches.

With our upsampled, normalized, segmented datasets of either 200-note chunked out tensors, or one dimensional vectors of length 128, we proceeded to define, train, and evaluate our DL models.

**Model Architectures of LSTM, BiLSTM and CNN Models:**

To optimize model performance, a grid search approach was applied for hyperparameter tuning.

**Purpose:** The goal was to systematically explore different combinations of model settings (hyperparameters) to identify the configuration that yields the highest validation accuracy and balanced performance across classes.

**Selection Criteria:** Choosen the hyperparameter set that provided the **best trade-off** between accuracy, precision, recall, and F1-score. Prioritized settings that improved generalization (small gap between training and validation performance). This tuning process ensured that the LSTM, BiLSTM + Attention, and CNN models were trained with **their respective best-performing configurations**, leading to improved accuracy, reduced overfitting, and better class-level balance compared to default parameters.

| Feature | LSTM | BiLSTM + Attention | CNN (Piano Roll) |
|---|---|---|---|
| **Input Shape** | (200, 14) | (200, 14) | (time_steps, n_features) piano roll representation |
| **Main Layers** | LSTM (64 units) | Bidirectional LSTM (64 units, return_sequences=True) | 6 × [Conv1D → Dropout(0.3) → MaxPooling1D(2)] |
| **Special Components** | L2 regularization on input, recurrent, and output weights | Attention layer after BiLSTM to weight important timesteps | Progressive Conv1D filters: 256 → 256 → 128 →128 → 64 → 64 |
| **Regularization** | Dropout (0.5), L2 regularization (λ=0.01) | Dropout (0.5) | Dropout (0.3) after convolutional and dense layers |
| **Dense Layers** | Dense (softmax, 4 classes) | Dense (softmax, 4 classes) | Dense(64) → Dropout(0.3) → Dense(64) →Dropout(0.3) → Dense(softmax, 4 classes) |
| **Loss Function** | Sparse categorical crossentropy | Sparse categorical crossentropy | Categorical crossentropy |
| **Optimizer** | Adam (lr=0.001) | Adam (lr=0.001) | Adam (lr=0.001) |
| **Batch Size** | 128 (best config) | 32 (best config) | 32 |
| **Best Validation Accuracy** | 0.9288 | 0.9376 | 0.7817 |

| | | | |
|---|---|---|---|
| **Input Shape** | (200, 14) | (200, 14) | (time_steps, n_features) piano roll representation |
| **Main Layers** | LSTM (64 units) | Bidirectional LSTM (64 units, return_sequences=True) | 6 × [Conv1D → Dropout(0.3) → MaxPooling1D(2)] |

The LSTM architecture processes 200×14 feature sequences using a single 64-unit LSTM layer with L2 regularization on all weights, followed by 50% dropout and a softmax output for 4 composers. The BiLSTM + Attention model extends this by processing sequences in both directions and applying an attention mechanism to emphasize important timesteps, leading to more stable validation accuracy. The CNN (piano roll) model applies six stacked Conv1D–Dropout–Pooling blocks to the piano roll representation, progressively reducing filter sizes from 256 to 64. This is followed by dense layers with dropout before the softmax output. While CNN captures strong local feature patterns, its validation accuracy was lower compared to recurrent models, reflecting the advantage of temporal context modeling in LSTM-based approaches. Train/Validation loss and accuracy plots can be found under Figures 6-11.

## Results

Given the success of the chosen architectures on their corresponding validation datasets, we went on to evaluate the LSTM, BiLSTM, and CNN performance on the test datasets. To understand the model's performance we are using as metrics precision, recall, the f1-score, and accuracy. Leveraging the metrics at both a composer level and a dataset level. We also went ahead and plotted their corresponding confusion matrices (Figures 12-14).

Precision measures the number of correct instances divided by all instances that were classified as correct. It helps us understand how much we can rely on our model when it makes a claim (e.g. this piece belongs to Mozart). Recall measures the number of instances classified as correct divided by all instances that are correct (independent of classification). In our case it would

help us understand the ability of the model to correctly identify all pieces belonging to a composer (only 50% of the pieces were correctly identified). The f-1 score is a weighted average between precision and recall that seeks to offer a performance metric that takes both precision and recall into account. Finally, accuracy represents the proportion of correctly classified instances divided by all available instances. Precision, recall, and the f-1 score are computed on an individual basis (composer); where True = belongs to a composer, and False = does not belong to a composer. The model's corresponding precision, recall, and the f-1 scores are then averages of their individual category-based scores.

**Test Results**

The evaluation of our trained models on our test dataset produced the following results:

| Architecture | precision | recall | f1-score | support | Accuracy |
|---|---|---|---|---|---|
| LSTM | 0.86 | 0.83 | 0.82 | 2238 | **0.856** |
| CNN | 0.83 | 0.82 | 0.82 | 2238 | **0.820** |
| BiLSTM | 0.93 | 0.92 | 0.90 | 2238 | **0.902** |

A composer-level results table can be found under Figure 16.

## Discussion

From both composer-averaged metrics and composer-level metrics, we were able to extract some meaningful insights. Overall, the models performed worse on recall when compared to precision, and accuracy tends to be closer to precision. Mozart was challenging for the models to identify. It mistook a lot of pieces belonging to Mozart as belonging to other composers. Based on the feature distributions, this could be due to the fact Mozart had a very distinct, conventional, narrow ranged style. As most of his distributions gravitate towards the center, and we see steep peaks on the distribution plots often (high concentration).

Specifically regarding the better performing BiLSTM and LSTM architectures:

- Chopin had high recall but low precision, meaning the model mistook a lot of pieces belonging to other composers as belonging to Chopin, yet it didn't fail to correctly identify the pieces belonging to Chopin.
- Mozart had significantly low recall and high precision , meaning the model failed to correctly identify a lot of pieces that belonged to Mozart, yet it can be trusted when it predicts a piece belongs to Mozart. Mozart also had an accuracy considerably lower than all other composers, reinforcing the belief that the model failed to correctly classify all of Mozart's pieces.
- Beethoven had high precision and recall, meaning the model correctly mapped the pieces belonging to Beethoven.
- Bach's performance average, somewhat skewed to a lower recall and high precision.

**Infrastructure and Tools Used**

MIDI files were preprocessed to extract features such as tempo, average pitch, pitch variance, chord density, and note durations using the pretty_midi python library. To train the models we tested locally on our machines smaller versions of the datasets and moved on to the cloud for the more computationally expensive tasks. A more detailed disclosure of what we used is as follows:

• Platform: Amazon SageMaker and local machines

• Instance Type: ml.g4dn.2xlarge (Amazon)

• Storage: 20 GB

• Main Libraries: TensorFlow, Scikit-learn, PrettyMIDI, Matplotlib, Seaborn, Pandas, NumPy

• Visualizations: Classification reports, feature distribution plots, confusion matrices, PCA, Train/Test/Validation Accuracy and Loss plots, ROC curves.

**Deployment -** https://music-genre-and-composer-classification.onrender.com/

We deployed the finalized trained BiLSTM model as a simple and lightweight, Dockerized Flask app. The service runs on a server with an interactive web page that submits a midi file to the API and displays the model's predictions in real time. We accept user uploaded MIDI files and preprocess them by parsing tracks, normalizing tempo, and encoding note sequences before passing them into the model. While we only support predictions for Bach, Beethoven, Chopin, and Mozart, the model returns a probability for each of them. The web app also lets you preview your uploaded MIDI file and export the rendered audio to a WAV file.

**Conclusion**

Training Deep Learning architectures on MIDI files proved to be feasible and effective. Leveraging the ability of DL to process data representations in the form of tensors, we were able to achieve composer prediction accuracies ranging from 0.82 to 0.90. The LSTM, BiLSTM, CNN models demonstrated strong capability in composer classification using MIDI-derived features. The BiLSTM achieved the highest accuracy and stability , followed by the LSTM. Making them both more reliable choices for this task than our trained CNN model. The models effectively identified local features for each composer from both our set of midi derived features and the piano_roll sequence data representations. Training on a larger number of composers, further increasing the applicability of these models, is both achievable and recommended given the improved performance of DL models with larger (quality) datasets. It is worth noting that training is computationally expensive, and bigger datasets would require suitable hardware. Our models did tend to have a higher precision and lower recall, which could have been the product of the identified class imbalances on the dataset. Further exploration of improved class balancing techniques, and feature extraction would be recommended. Future work could also incorporate raw MIDI audio embeddings or ensemble

methods for improved performance. A set of possible improvement recommendations can be found down below.

**Future Recommendations:**

- Ensemble Learning: Combine predictions from multiple architectures (e.g., LSTM, BiLSTM, Transformer-based models) to improve robustness and reduce bias toward certain classes.
- Transformer-Based Models: Experiment with Music Transformer or Performer architectures, which can capture long-term dependencies better than RNN-based models.
- Advanced Data Augmentation: Introduce tempo variation, pitch shifting, and rhythmic perturbations in a controlled manner to improve generalization.
- Feature Enrichment: Add higher-level musical features such as harmonic intervals, key signatures, rhythmic complexity scores, and motif repetition counts.
- Class Imbalance Handling: Use focal loss or cost-sensitive learning to improve performance for underrepresented classes without oversampling or undersampling.
- Regularization Enhancements: Apply techniques like layer normalization, weight decay (L2), or stochastic depth to further control overfitting.
- Hyperparameter Optimization: Run Bayesian optimization or genetic algorithms for fine-tuning parameters like units, dropout rates, learning rate schedules, and attention dimensions.
- Multi-Modal Integration: Combine symbolic features (MIDI) with audio spectrograms for richer representation learning.
- Model Interpretability: Use SHAP or Integrated Gradients to visualize which notes or patterns contribute most to classification decisions.
- Deployment & Real-Time Prediction: Optimize the best model for low-latency inference to run in real-time composer identification applications.

## References

Bohdan, F. (2019). midi_classic_music. Kaggle. https://www.kaggle.com/datasets/blanderbuss/midi-classic-music/code

Olah, C. (2015, August 27). Understanding LSTM networks. Colah's blog. https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Dalianis, H. (2018). Evaluation metrics and evaluation. In Clinical text mining (pp. 45–53). Springer. https://doi.org/10.1007/978-3-319-78503-5_6

PyTorch. (2025, July 11). Understanding and implementing BiLSTM with attention in PyTorch. CodeGenes.net. https://www.codegenes.net/blog/bilstm-attention-pytorch/

Craffel, E., Raffel, C., & Pardo, B. (n.d.). pretty_midi — pretty_midi 0.2.10 documentation. https://craffel.github.io/pretty-midi/

Géré, R., Olivetti, E., & Donin, N. (2024). Improved symbolic drum style classification with grammar-based hierarchical representations. arXiv. https://doi.org/10.48550/arXiv.2407.17536

Zhang, Y., & Li, T. (2025). Music genre classification with parallel convolutional neural networks and capuchin search algorithm. Scientific Reports, 15(1). https://doi.org/10.1038/s41598-025-90619-7

Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. Deeplearningbook.org; MIT Press. https://www.deeplearningbook.org/

J, R. T. J. (2021, September 10). LSTMs Explained: A Complete, Technically Accurate, Conceptual Guide with Keras. Analytics Vidhya. https://medium.com/analytics-vidhya/lstms-explained-a-complete-technically-accurate-conceptual-guide-with-keras-2a650327e8f2

## Annex

**Figure 1 Extracted features from MIDI files for LSTMs.**

| Feature | Description |
|---|---|
| Pitch | The MIDI pitch number (0–127) of the note. |
| Duration | The time the note is held (end time - start time). |
| Velocity | The intensity/loudness of the note (0–127). |
| DeltaTime | Time difference between the current and previous note's start time. |
| Start | Start time of the note in seconds. |
| End | End time of the note in seconds. |
| Interval | Difference in pitch from the previous note. |
| Program | MIDI program number representing the instrument used. |
| Accented | Binary flag indicating whether the note is played loudly (velocity > 80). |
| Composer | The name of the composer associated with the MIDI file. |
| ComposerID | Numeric class ID for the composer (used for classification). |
| Tempo | Estimated global tempo (beats per minute) of the MIDI file. |
| PitchRange | Difference between the highest and lowest pitch in the note chunk. |
| NoteDensity | Number of notes per second within the chunk. |
| RepetitionRate | Proportion of notes that have the same pitch as their immediate predecessor. |
| AvgInterval | Average pitch interval between consecutive notes in the chunk. |
| RhythmicVariety | Standard deviation of note durations in the chunk (indicates rhythm complexity). |
| ChordDensity | Fraction of notes that overlap with the previous note (indicates harmony/chords). |

**Figure 2 Feature distribution of the extracted features.**

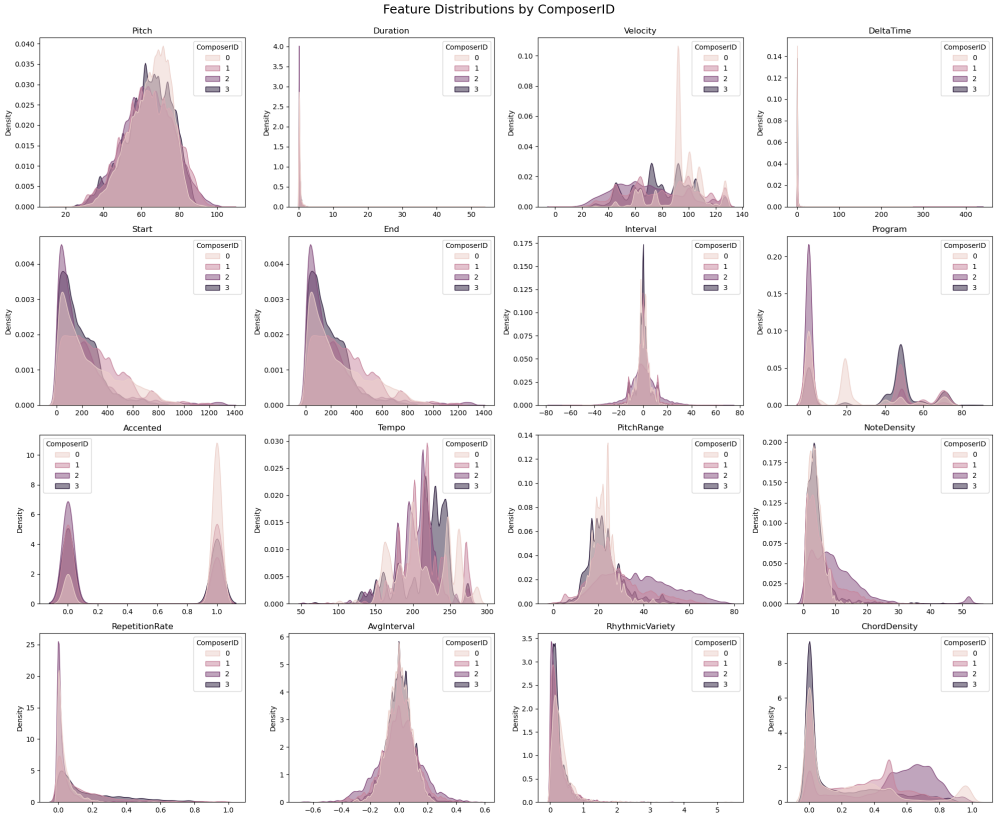


**Figure 3 Feature distribution of the compared original and augmented features.**

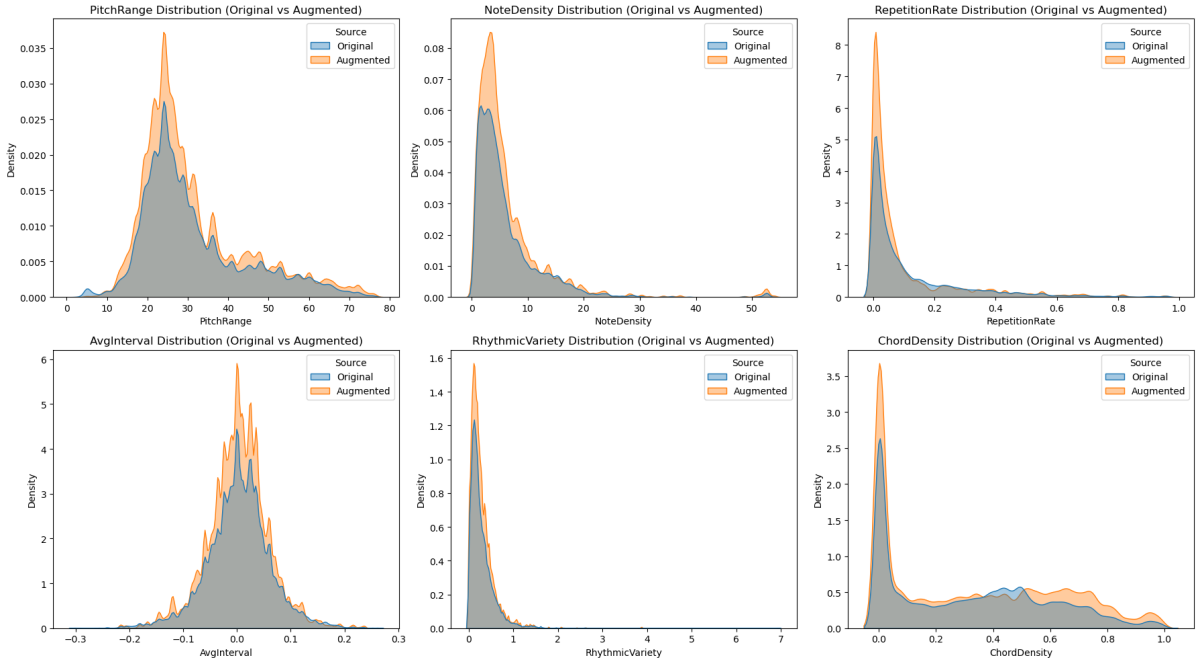


**Figure 4 Feature Correlation Matrix of the extracted features.**

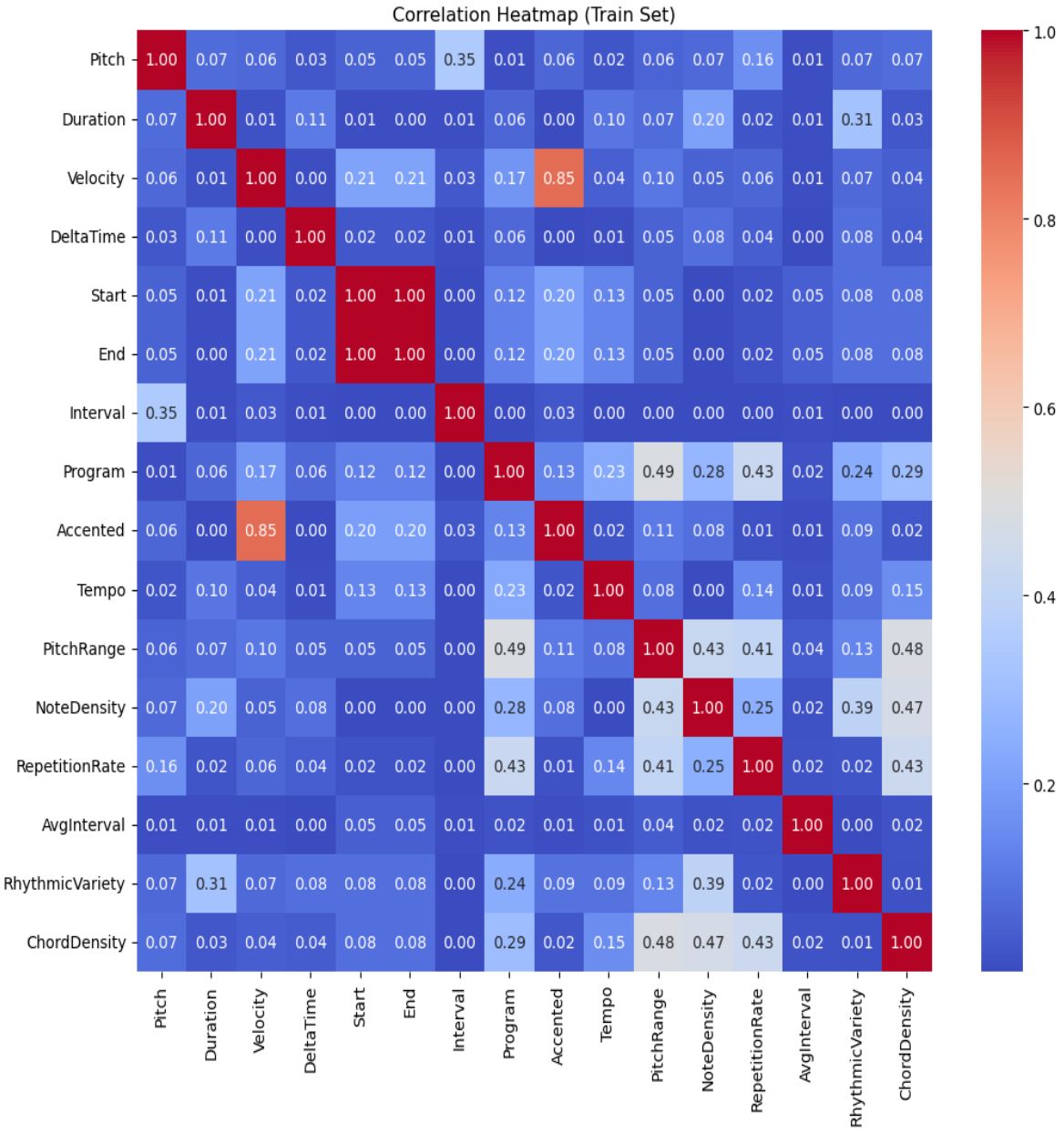
Correlation Heatmap (Train Set)
Pitch 1.00 0.07 0.06 0.03 0.05 0.05 0.35 0.01 0.06 0.02 0.06 0.07 0.16 0.01 0.07 0.07
Duration 0.07 1.00 0.01 0.11 0.01 0.00 0.01 0.06 0.00 0.10 0.07 0.20 0.02 0.01 0.31 0.03
Velocity 0.06 0.01 1.00 0.00 0.21 0.21 0.03 0.17 0.85 0.04 0.10 0.05 0.06 0.01 0.07 0.04
DeltaTime 0.03 0.11 0.00 1.00 0.02 0.02 0.01 0.06 0.00 0.01 0.05 0.08 0.04 0.00 0.08 0.04
Start 0.05 0.01 0.21 0.02 1.00 1.00 0.00 0.12 0.20 0.13 0.05 0.00 0.02 0.05 0.08 0.08
End 0.05 0.00 0.21 0.02 1.00 1.00 0.00 0.12 0.20 0.13 0.05 0.00 0.02 0.05 0.08 0.08
Interval 0.35 0.01 0.03 0.01 0.00 0.00 1.00 0.00 0.03 0.00 0.00 0.00 0.00 0.01 0.00 0.00
Program 0.01 0.06 0.17 0.06 0.12 0.12 0.00 1.00 0.13 0.23 0.49 0.28 0.43 0.02 0.24 0.29
Accented 0.06 0.00 0.85 0.00 0.20 0.20 0.03 0.13 1.00 0.02 0.11 0.08 0.01 0.01 0.09 0.02
Tempo 0.02 0.10 0.04 0.01 0.13 0.13 0.00 0.23 0.02 1.00 0.08 0.00 0.14 0.01 0.09 0.15
PitchRange 0.06 0.07 0.10 0.05 0.05 0.05 0.00 0.49 0.11 0.08 1.00 0.43 0.41 0.04 0.13 0.48
NoteDensity 0.07 0.20 0.05 0.08 0.00 0.00 0.00 0.28 0.08 0.00 0.43 1.00 0.25 0.02 0.39 0.47
RepetitionRate 0.16 0.02 0.06 0.04 0.02 0.02 0.00 0.43 0.01 0.14 0.41 0.25 1.00 0.02 0.02 0.43
AvgInterval 0.01 0.01 0.01 0.00 0.05 0.05 0.01 0.02 0.01 0.01 0.04 0.02 0.02 1.00 0.00 0.02
RhythmicVariety 0.07 0.31 0.07 0.08 0.08 0.08 0.00 0.24 0.09 0.09 0.13 0.39 0.02 0.00 1.00 0.01
ChordDensity 0.07 0.03 0.04 0.04 0.08 0.08 0.00 0.29 0.02 0.15 0.48 0.47 0.43 0.02 0.01 1.00
Pitch
Duration
Velocity
DeltaTime
Start
End
Interval
Program
Accented
Tempo
PitchRange
NoteDensity
RepetitionRate
AvgInterval
RhythmicVariety
ChordDensity
1.0
0.8
0.6
0.4
0.2

**Figure 5 LSTM cell.**

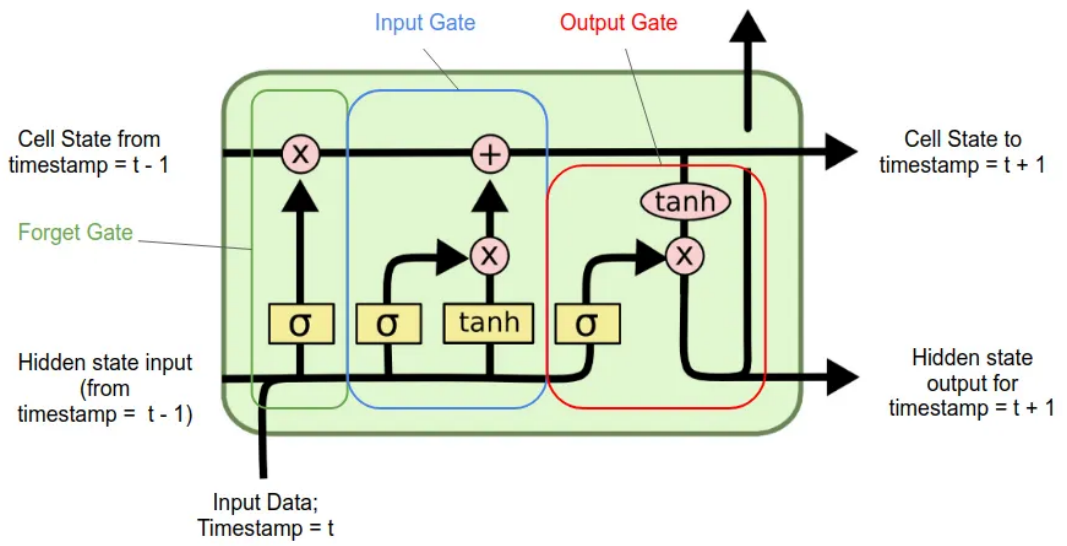


**Figure 6 Train/Validation Accuracy plots for each of the 5 best LSTM hyperparameter combinations.**

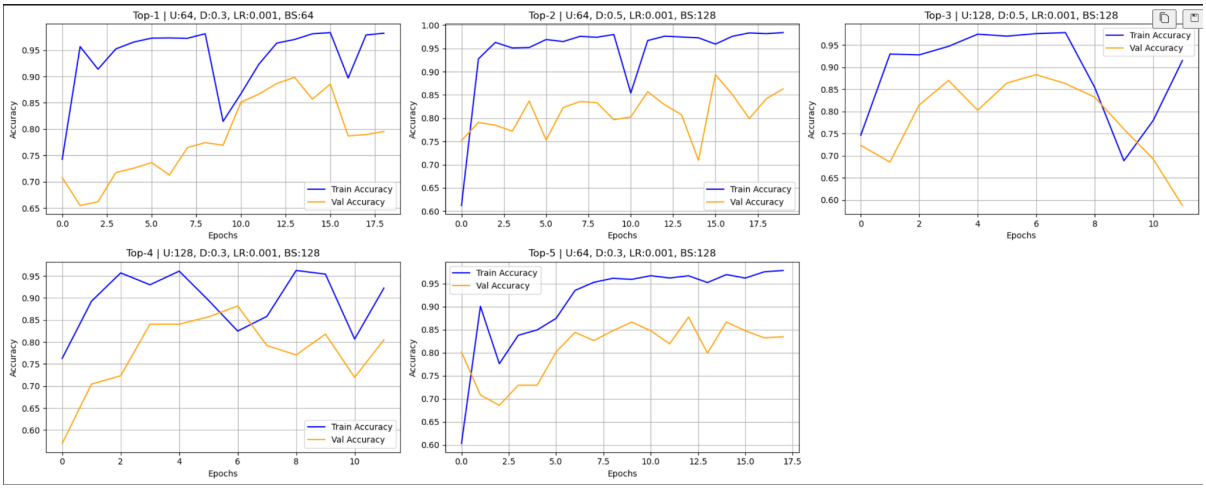

**Figure 7 Train/Validation loss plots for each of the 5 best LSTM hyperparameter combinations.**

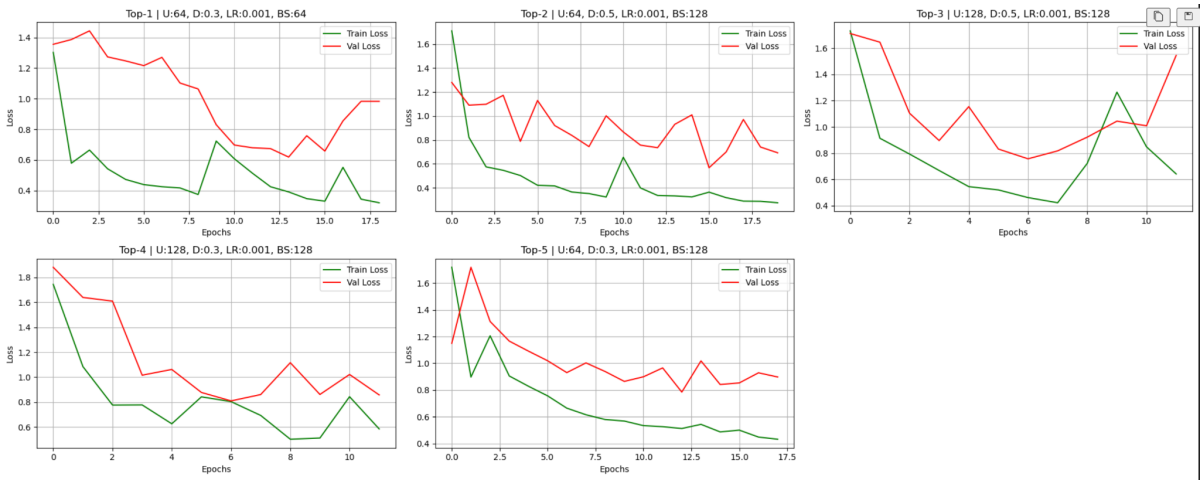


**Figure 8 Train/Validation accuracy and loss plots for the best 3 BiLSTM hyperparameter combinations.**

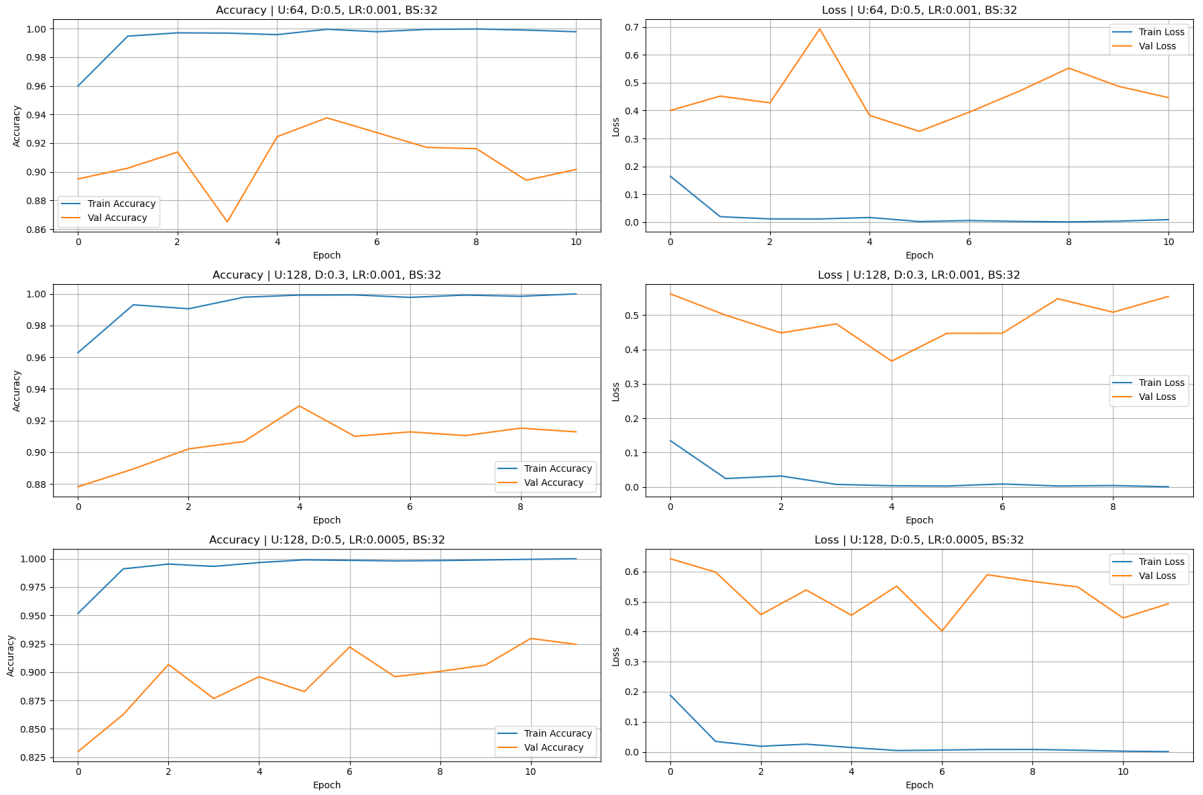

**Figure 9 Train/Validation accuracy and loss plots for the trained BiLSTM.**

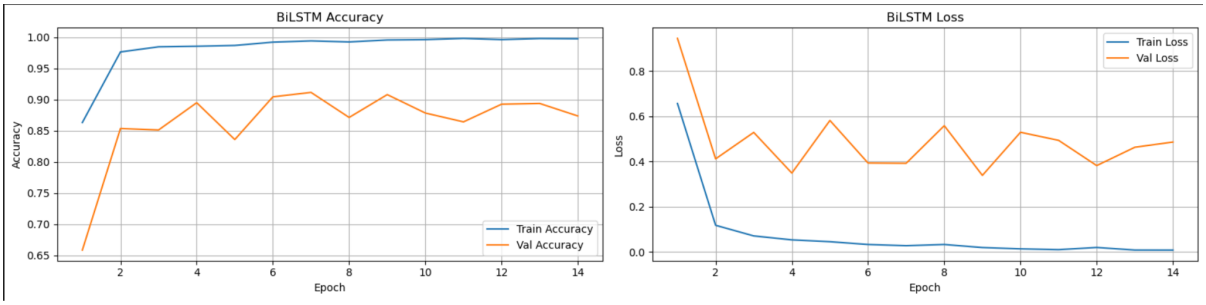


**Figure 10 Train/Validation loss and accuracy plots for the trained LSTM model.**

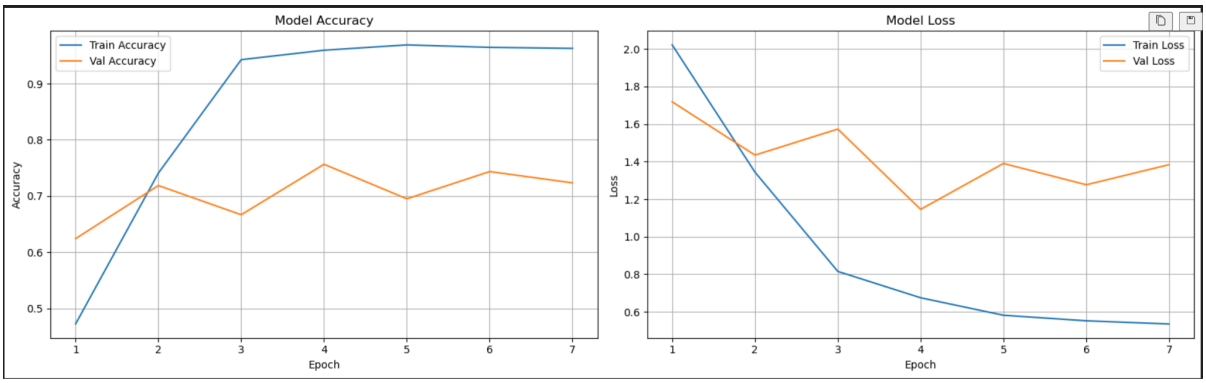


**Figure 11 Train/Validation loss and accuracy plots for the trained CNN model.**

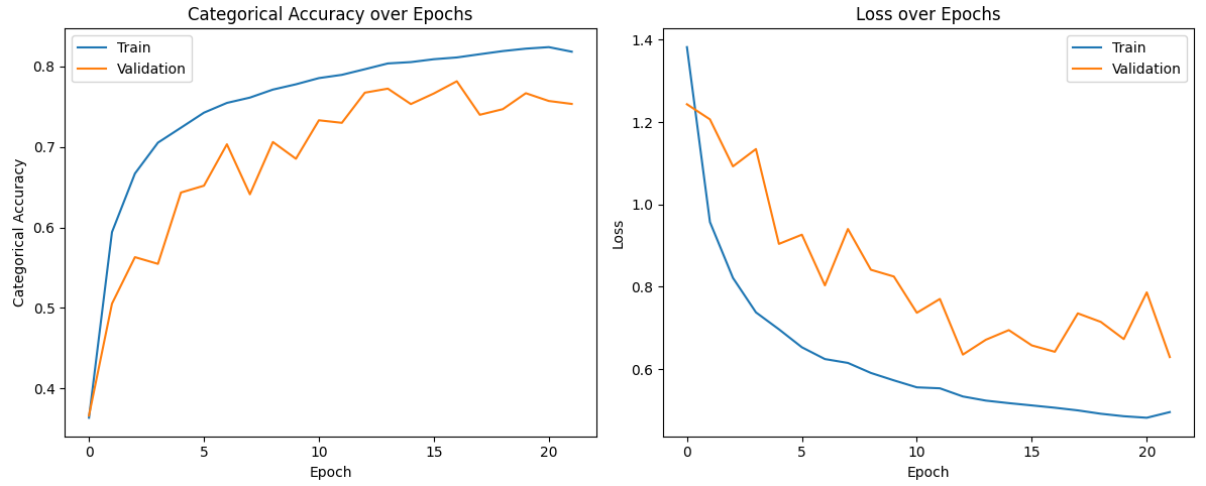

**Figure 12 LSTM Confusion Matrix.**

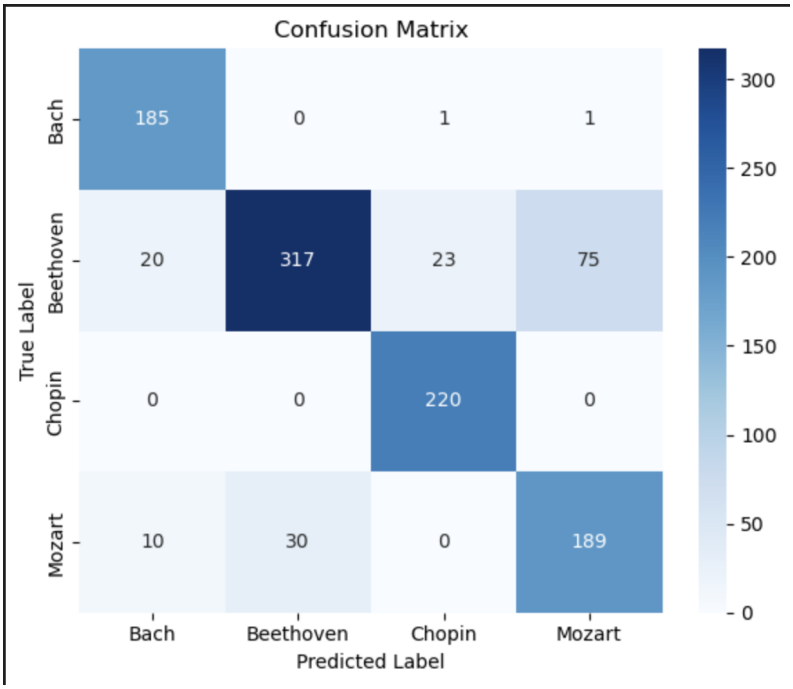


**Figure 13 CNN Confusion Matrix.**

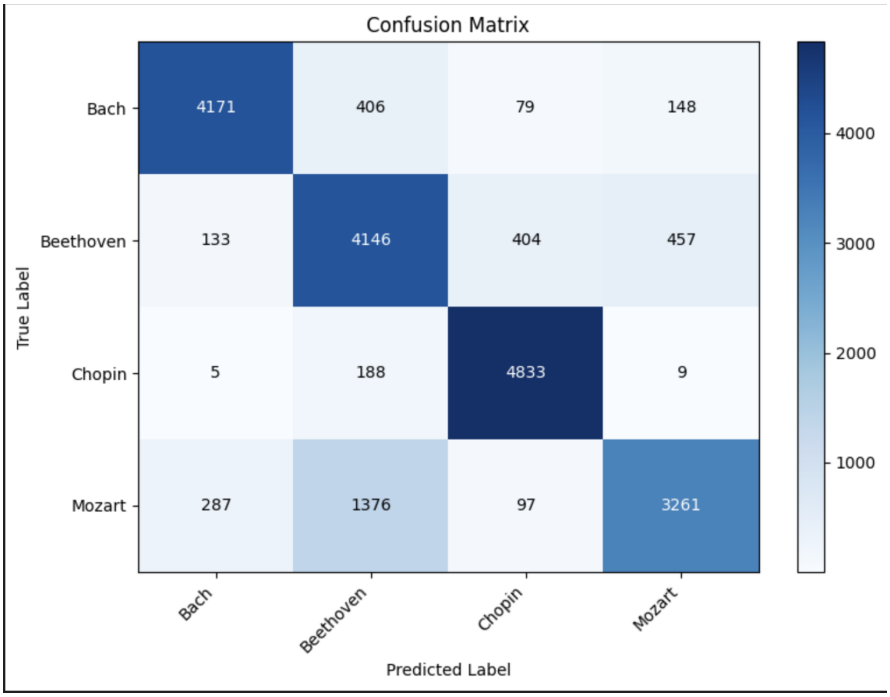

**Figure 14 BiLSTM Confusion Matrix.**

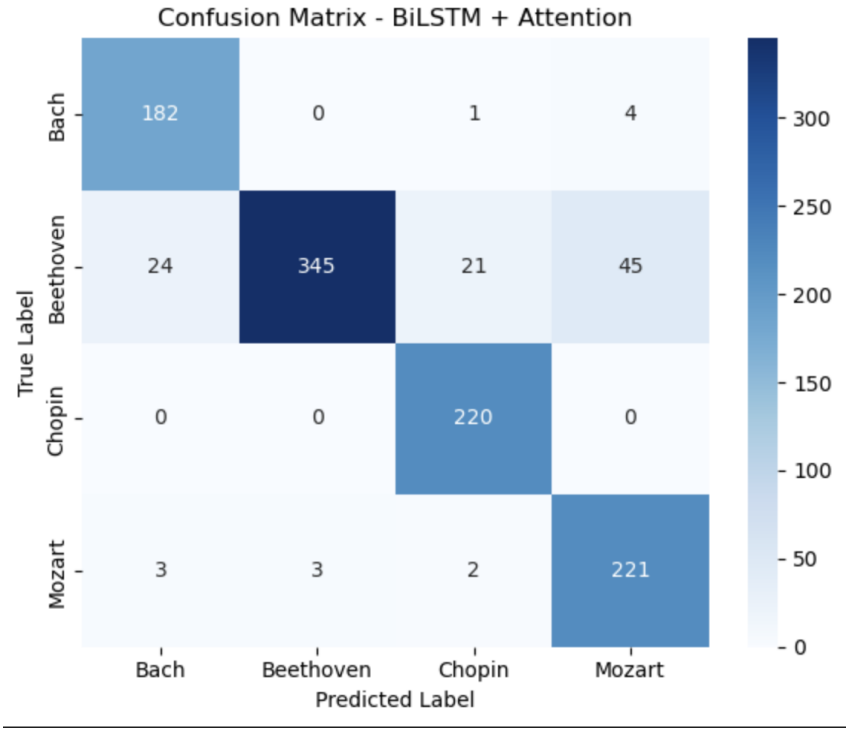


**Figure 15 Composer-level results of the final models**

**Overall Performance Comparison Table:**

| Metric | LSTM Model | BiLSTM + Attention Model | CNN Model |
|---|---|---|---|
| Test Accuracy | 85.06% | 90.38% | 82.06% |
| Test Loss | 0.7187 | 0.2460 | 0.56 |
| Macro Avg Precision | 0.85 | 0.90 | 0.83 |
| Macro Avg Recall | 0.89 | 0.93 | 0.82 |
| Macro Avg F1-Score | 0.86 | 0.91 | 0.82 |
| Weighted Avg F1-Score | 0.85 | 0.90 | 0.82 |

**Confusion matrix comparision:**

| Class | LSTM | BiLSTM + Attention | CNN |
|---|---|---|---|
| Bach | 185 / 187 (99%) | 181 / 187 (97%) | 4171 / 4804 ≈ 86.8% |
| Beethoven | 317 / 435 (73%) | 344 / 435 (79%) | 4146 / 5140 ≈ 80.7% |
| Chopin | 220 / 220 (100%) | 220 / 220 (100%) | 4833 / 5035 ≈ 96.0% |
| Mozart | 189 / 229 (83%) | 222 / 229 (97%) | 3261 / 5021 ≈ 64.9% |