# Final Team Project: Music Genre and Composer Classification Using Deep Learning

**Submitted by:** Team 7

(Satyam Kumar, Rishabh Malik, Shashidhar Kumbar)

**AAI – 511** (Neural Networks and Deep Learning)

University of San Diego

## 1 ABSTRACT

Identifying the composer of a musical piece is a challenging problem in symbolic music analysis due to overlapping stylistic features, class imbalance, and complex temporal dependencies inherent in musical structure. This project presents an approach combining **Convolutional Neural Networks (CNNs)** and **Long Short-Term Memory (LSTM)** architectures to classify piano compositions into one of four composers: **Bach, Beethoven, Chopin, and Mozart**. Symbolic music in MIDI format was pre-processed into piano roll sequences, and data augmentation techniques (pitch and tempo shifting) were applied to enhance generalization. We trained individual CNN and LSTM models, evaluated them separately, and then combined them using two ensemble strategies: **weighted softmax averaging** and **logit averaging**.

Results showed that ensembling consistently improved accuracy, with the best performance achieved by **weighted ensembles** using pitch and tempo augmentation (Test Accuracy: **80.4%**, Weighted Precision: **0.801**, Weighted Recall: **0.804**, Weighted F1-score: **0.802**). This study highlights the complementary strengths of CNN and LSTM architectures for symbolic music analysis and the importance of augmentation and ensembling in improving classification robustness.

## 2 INTRODUCTION

Composer classification seeks to determine the creator of a given musical work. This problem is relevant to **musicology, copyright verification, recommendation systems, and AI music education tools**. In classical music, composers have distinct yet sometimes overlapping stylistic traits. For instance:

- **Bach** often uses counterpoint and polyphonic textures.
- **Beethoven** incorporates dynamic contrasts and thematic development.
- **Chopin** focuses heavily on expressive piano writing and rubato.
- **Mozart** is characterized by balanced phrases and clear classical forms.

Traditional composer classification relied on **expert-designed features** such as harmonic intervals, melodic contours, and rhythmic histograms. However, these handcrafted features fail to fully capture the **complex dependencies** across time in a composition.

Deep learning addresses this by **learning representations directly from the data**. In this project:

- CNNs capture **spatial relationships** in piano roll matrices (pitch × time).
- LSTMs capture **temporal sequences** in note events. By combining both, we aim to leverage spatial-temporal synergy.

## 3 Problem Statement

We aim to develop a **robust classification model** capable of predicting the composer of a piano composition from symbolic representations. Challenges include:

i. **Overlapping styles** between composers leading to misclassifications.

ii. **Class imbalance**: some composers have more training samples than others.

iii. **Data sparsity**: musical events occupy only a small portion of the pitch-time space.

iv. **Training resource limitations**: no GPU availability, requiring CPU-based optimization.

### Hypotheses

Given a MIDI piece x, predict y ∈ {Bach, Beethoven, Chopin, Mozart}.

- Hypothesis H1: Class-weighted optimization and balanced sampling reduce majority-class bias and improve macro recall.
- Hypothesis H2: Pitch and tempo augmentation increase robustness and minority-class recall without hurting majority-class precision.
- Hypothesis H3: A calibrated ensemble of CNN and LSTM outperforms either model alone.

## 4 Dataset

The dataset forms the foundation of this project, containing symbolic piano music from four canonical composers : Johann Sebastian Bach, Ludwig van Beethoven, Frédéric Chopin, and Wolfgang Amadeus Mozart. The focus is to classify pieces based solely on symbolic patterns in their note sequences, making this dataset a critical element in the success of the modeling pipeline.

### 4.1 Overview of Data Collection and Structure

- **Base Directory**: /content/drive/MyDrive/midiclassics (Google Drive mount for Google Colab runtime).
- **Recursive Scan Results:**
    - Total MIDI files discovered through recursive directory traversal: **1,530 files**.
    - Custom file crawler implemented with Python’s os.walk() and fnmatch to ensure MIDI files in nested directories are included.
- **Composer Detection Logic:**
    - Robust **alias matching** on path tokens to detect composer identity.
        - Examples of handled aliases:
            - "JSBach", "Bach", "J.S.Bach" → Bach

        - "LVB", "Beethoven", "L.V.Beethoven" → Beethoven
        - "Chopin", "FredericChopin", "F.Chopin" → Chopin
        - "Mozart", "W.A.Mozart", "WolfgangAmadeusMozart" → Mozart
    - The parser strips file extensions, lowercases all text, and searches for canonical name patterns to avoid misclassification due to inconsistent directory naming.

- **Target Composer Selection:**
    - The dataset initially contained works from a variety of composers, but for this project, **only four were retained** to match the classification target set.
    - After filtering:
        - Bach: **925** files
        - Beethoven: **212** files
        - Chopin: **136** files
        - Mozart: **257** files

## 4.2 Data Source

The raw dataset consists entirely of **MIDI-format piano scores** obtained from a public-domain classical music collection. Each file contains a symbolic representation of a musical piece rather than an audio waveform, enabling **direct structural and harmonic analysis**.

The dataset comes **pre-split** into training, validation, and test subsets to ensure:

- No **data leakage** between splits.
- Reproducibility across experiments.
- Compatibility with augmentation pipelines applied only to the training split.

## 4.3 Data Format

❖ **MIDI to Piano Roll Conversion**

- Each MIDI file is converted to a **piano roll representation** : a two-dimensional matrix $\boldsymbol{P \in RT \times 128}$ where:
    - **Rows** represent MIDI pitches (21 to 108 inclusive for piano range, but stored in a 128-bin array for compatibility).
    - **Columns** represent discrete time frames, with a fixed temporal resolution.
- **Velocity Values:**

- Retained as either binary (note-on/note-off) or scaled float values to preserve dynamics.
- MIDI velocities clipped to [0,127][0,127][0,127] and normalized to [0,1][0,1][0,1].

❖ **Label Encoding:**

- Each file is assigned a **numeric composer ID**:
  - Bach → 0
  - Beethoven → 1
  - Chopin → 2
  - Mozart → 3

❖ **Class Distribution (Before Augmentation)**

A preliminary analysis revealed significant **class imbalance**:

| Composer | Train Samples | Validation Samples | Test Samples |
|---|---|---|---|
| **Bach** | 740 | 92 | 93 |
| **Beethoven** | 169 | 22 | 21 |
| **Chopin** | 109 | 13 | 14 |
| **Mozart** | 206 | 26 | 25 |

- **Observation:**
  - Bach dominates the dataset (≈ 60% of all samples).
  - Chopin is the smallest class (≈ 7% of all samples).
- **Impact on Modeling:**
  - Without mitigation, models tend to overfit to Bach, inflating accuracy while degrading macro-average recall.
  - Class balancing strategies such as **stratified sampling**, **class-weighted loss**, and **data augmentation** are essential.

## 4.4 Data Splitting Methodology

- **Split Ratio:** 80% Training, 10% Validation, 10% Testing.
- **Stratified Sampling:** Ensured **proportional representation** of each composer in every split.
- **Random Seed:** random_state=42 fixed to maintain reproducibility across runs.
- **Unique IDs:** Each file assigned a persistent numeric identifier stored in preprocessed_manifest.csv to guarantee consistent splits even after preprocessing.

## 4.5 Resulting Split Counts (Original Manifest)

- Train: Bach 740, Beethoven 169, Chopin 109, Mozart 206
- Val: Bach 92, Beethoven 22, Chopin 13, Mozart 26
- Test: Bach 93, Beethoven 21, Chopin 14, Mozart 25

## 4.6 MIDI → Piano Roll Conversion & Caching

**Preprocessing Steps:**

i. **Parsing:** MIDI files read using pretty_midi with event parsing to detect tempo, key signature, and note events.

ii. **Velocity Clipping & Normalization:**
   - Clip raw velocities to [0,127].
   - Normalize: v′=v/127v' = v / 127v′=v/127 → range [0,1].

iii. **Temporal Resolution:** Fixed frames-per-second fs=10fs = 10fs=10, meaning each time step represents 0.1 seconds of real time.

iv. **Shape Standardization:**
   - Target fixed length T=500T = 500T=500 frames.
   - For pieces shorter than 500 frames → zero-padding.
   - For pieces longer → truncation to first 500 frames (retaining opening motifs).

v. **Array Storage:** Saved as .npy files in /preprocessed/ directory.

vi. **Manifest Update:** Indexed in preprocessed_manifest.csv with metadata:
   - File path

  - Composer ID
  - Original length in frames
  - Augmentation applied (flag)

## 4.7 PROCESSING OUTCOMES AND ISSUES

During preprocessing, several observations and minor issues arose:

- **pretty_midi Warnings:**
  - Warnings for meta events on non-zero tracks. These were non-critical and ignored after manual verification.
- **File Failures:**
  - Two files failed to parse due to **invalid key/mode metadata**. These were logged and excluded from the dataset.
- **Final Dataset After Preprocessing:**
  - Train: Bach 740, Beethoven 168, Chopin 109, Mozart 205
  - Val: Bach 92, Beethoven 22, Chopin 13, Mozart 26
  - Test: Bach 93, Beethoven 21, Chopin 14, Mozart 25

## 4.8 SANITY CHECK

A post-processing sanity check was conducted to ensure format consistency:

- Example file shape: (500, 128)
- Value range: min=0.0, max=1.0
- Sparsity: ~96% zeros, consistent with musical note density in classical piano works.

## 5 PROJECT FLOW:

The project was executed in a **multi-stage pipeline**, each step carefully designed to address dataset challenges, optimize feature representations, and leverage model architectures that exploit the structural and temporal characteristics of symbolic music data. Below is a **comprehensive breakdown** of each phase, including technical decisions, issues encountered, and their resolutions.

### 5.1 DATA COLLECTION

- **Objective:** Aggregate a clean, representative set of piano compositions from the four target composers: Bach, Beethoven, Chopin, and Mozart.
- **Source:** Public-domain MIDI repositories.
- **Directory Structure:**
    - Maintained a **hierarchical organization**: /composer_name/work_id.mid to ensure composer labels are easily inferable from path tokens.
    - Implemented a **recursive scan script** in Python to detect .mid files regardless of nesting depth.
- **File Verification:**
    - All files parsed with pretty_midi to ensure compatibility before inclusion.
    - Invalid or corrupted MIDI files were automatically excluded.

**Challenges & Resolutions**

- **Issue:** Inconsistent file naming and composer aliases (e.g., “L. v. Beethoven” vs. “Beethoven”).
    - **Solution:** Implemented a **robust alias mapping** dictionary for composer name normalization.
- **Outcome:** Final dataset contained **1,528 valid MIDI files** after filtering.

### 5.2 DATA PREPROCESSING

**Goal:** Transform symbolic MIDI data into fixed-size numeric arrays suitable for deep learning models.

**Steps:**

i. **MIDI to Piano Roll Conversion:**
   - Used pretty_midi to convert note events into a binary or velocity-based **piano roll matrix** of shape (T, 128) where T is the number of time frames and 128 represents the full MIDI pitch range.
   - Limited pitch range to the standard piano span (21–108) but retained the 128-dimension vector for compatibility.

ii. **Pitch Normalization:**
   - Standardized pitch axis to ensure identical note mapping across all pieces.
   - Prevented alignment mismatches between different files.

iii. **Time-Step Quantization:**
   - Fixed temporal resolution to **10 frames per second (fps)**.
   - Ensured uniform time indexing across all samples for model consistency.

iv. **Zero-Padding / Truncation:**
   - Target sequence length set to **T = 500 frames**.
   - Shorter sequences padded with zeros; longer sequences truncated from the end.

**Challenges & Resolutions:**

- **Issue:** pretty_midi raised warnings for meta events on non-zero tracks.
  - **Solution:** Ignored after manual verification that these did not affect note content.
- **Outcome:** All preprocessed files stored in .npy format under /preprocessed/ for fast loading.

## 5.3 Data Augmentation

**Objective:** Address severe **class imbalance** and improve **model generalization**.

**Techniques Applied:**

i. **Pitch Shifting:**
   - Transposed entire pieces by ±1, ±2 semitones.
   - Implemented directly in piano roll space by rolling pitch indices.
   - Preserved harmonic structure while introducing variability.

ii. **Tempo Scaling:**
   - Adjusted playback speed by ±5%, ±10%.
   - Achieved by resampling along the time axis before padding/truncation.
   - Introduced rhythmic diversity without altering pitch.

**Challenges & Resolutions:**

- **Issue:** Some tempo-scaled sequences exceeded the fixed 500-frame length.
  - **Solution:** Applied **dynamic truncation** after augmentation to maintain fixed size.
- **Outcome:** Minority classes (Beethoven, Chopin) increased by ~2× in sample count, reducing imbalance ratio from **6.8:1 to 2.4:1** (Bach to Chopin).

## 5.4 Feature Extraction

Given the nature of symbolic music, two complementary feature formats were extracted:

❖ **For CNN Models:**

- Input: **2D Piano Roll Matrices**
- Shape: (1, 500, 128) → Treated as an image where one axis represents time and the other pitch.
- Preprocessing: Min-max normalization to [0,1].

❖ **For LSTM Models:**

- Input: **Sequential Note Events**
- Converted piano roll into time-ordered lists of active pitches per frame.

- Reduced dimensionality by storing **note-on events** rather than full binary matrix for each timestep.

❖ **Outcome:**

- CNNs specialized in **local harmonic/melodic pattern recognition**.
- LSTMs specialized in **long-range temporal dependencies** across bars and phrases.

❖ **Model Development**

Two primary architectures were implemented in PyTorch:

❖ **CNN Model:**

- **Convolution Layers:** Extract spatial features across pitch and time.
- **Pooling Layers:** Reduce dimensionality while preserving essential harmonic structures.
- **Dense Layers:** Map learned representations to composer probabilities via softmax.

❖ **LSTM Model:**

- **Stacked LSTMs:** Capture temporal progressions of note sequences.
- **Dropout Regularization:** Prevent overfitting to dominant classes.
- **Final Dense Layer:** Softmax output over four composers.

❖ **Challenges & Resolutions:**

- **Issue:** GPU not available in the runtime environment.
    - **Solution:** Optimized for **CPU execution** by reducing model size and using batch sizes of 8–16.
- **Outcome:** Both models trained within acceptable runtime limits on CPU.

## 5.5 TRAINING

**Training Strategy:**

- Loss Function: **CrossEntropyLoss** with **class weights** to counteract imbalance.
- Optimizer: Adam with learning rate scheduling.
- Early Stopping: Halt training if validation loss did not improve for 10 epochs.

**Results:**

- CNN: High precision on Bach but lower recall on minority classes.
- LSTM: Better recall on Beethoven and Chopin but slightly lower accuracy.

## 5.6 EVALUATION

**Metrics:**

- Accuracy, Precision, Recall, F1-score (macro-averaged).
- Confusion Matrices for per-class error analysis.

**Findings:**

- Misclassifications primarily occurred between composers with stylistic overlap (e.g., Mozart ↔ Beethoven).
- CNN excelled at harmonic motif recognition; LSTM at rhythmic structure.

## 5.7 OPTIMIZATION

**Techniques Applied:**

i. **Hyperparameter Tuning:**
   - Batch sizes, learning rates, and dropout rates tuned via grid search.

ii. **Ensemble Methods:**
   - Soft-voting ensemble combining CNN and LSTM outputs.
   - Improved macro-F1 by **~4%** compared to the best single model.

# 6 Data Preprocessing and Augmentation

## 6.1 Piano Roll Generation

MIDI files are parsed to extract note-on/note-off events. These are mapped into fixed-size time bins. Each piano roll is a **[pitch × time]** matrix with binary or velocity values.

## 6.2 Normalization

- Pitch range standardized to 21–108.
- Time resolution standardized across dataset.

## 6.3 Data Augmentation

1. **Pitch Shifting**

Transposing the piece by ±1 to ±3 semitones to increase dataset variety. **Mathematically**:
If PPP is a pitch matrix, shifting by kkk semitones gives:

$$Pi,j' = Pi-k,jP'\_\{i,j\} \ = \ P\_\{i-k,j\}Pi,j' = Pi-k,j$$

provided the shifted pitch stays within range.

2. **Tempo Shifting**

Scaling timing by a factor $\alpha\backslash alpha\alpha$ (e.g., 0.9–1.1). This changes the temporal spacing between events while keeping pitch constant.

## 6.4 Effect of Augmentation

Augmentation increased the effective dataset size, improved generalization, and helped balance classes, particularly Beethoven and Chopin.

## 6.5 Model Architectures

**CNN Architecture (Pitch-Tempo Augmentation)**

- **Input**: [Pitch × Time] matrix.
- **Layers**:
  1. Conv2D(32 filters, 3×3) + ReLU
  2. MaxPooling(2×2)
  3. Conv2D(64 filters, 3×3) + ReLU
  4. MaxPooling(2×2)
  5. Dense(128) + ReLU
  6. Dense(4) + Softmax

CNN captures **harmonic intervals**, **chord blocks**, and **local rhythmic motifs**.

## 6.6 LSTM ARCHITECTURE

- **Input**: Flattened sequence of active pitches per timestep.
- **Layers**:
    - LSTM(128 units, return_sequences=True)
    - Dropout(0.3)
    - LSTM(64 units)
    - Dense(4) + Softmax

LSTM captures **melodic contours** and **temporal motifs**.

# 7 ENSEMBLE METHODS

## 7.1 SOFTMAX WEIGHTED AVERAGING

If $pCNNp\_\{CNN\}pCNN$ and $pLSTMp\_\{LSTM\}pLSTM$ are probability vectors:

$$\begin{aligned} pensemble &= \alpha \cdot pCNN + (1-\alpha) \cdot pLSTMp\_\{ensemble\} \\ &= \backslash alpha \ \backslash cdot\ p\_\{CNN\} \ + \ (1 \ - \backslash alpha) \ \backslash cdot\ p\_\{LSTM\}pensemble \\ &= \alpha \cdot pCNN + (1-\alpha) \cdot pLSTM \end{aligned}$$

where α\alphaα is tuned on validation accuracy.

## 7.2 LOGIT AVERAGING

Convert probabilities to logits:

$$z = log\, p1 - pz \ = \backslash log\ \backslash frac\{p\}\{1-p\}z = log1 - pp$$

Average logits, then apply softmax for final probabilities.

## 8 Training Details

- **Optimizer:** Adam (lr=0.001, $\beta_1$=0.9, $\beta_2$=0.999), adaptive learning rates for stable convergence on sparse piano-roll data.
- **Loss Function:** CrossEntropyLoss with inverse-frequency class weights to address imbalance (Bach over-representation).
- **Batch Size:** 32, optimal balance between convergence stability and CPU memory usage.
- **Epochs:** Capped at 50, but early stopping typically triggered at 20–25 based on validation loss plateau (patience=7).
- **Learning Rate Scheduling:** ReduceLROnPlateau (factor=0.5, patience=5).
- **Hardware:** CPU-only (Google Colab Free Tier, 2 vCPUs, 12 GB RAM).
- **Runtime Optimizations:**
    - Cached preprocessed .npy files to skip repeated MIDI parsing.
    - Used PyTorch DataLoader for batched I/O.
    - Slightly reduced model complexity to fit CPU constraints without significant accuracy loss.

# 9 Results

## 9.1 CNN (Pitch Augmentation)

- Accuracy: 0.784
- Weighted Precision: 0.803
- Weighted Recall: 0.784
- Weighted F1: 0.791

## 9.2 Weighted Ensemble (Pitch+Tempo, α=0.70)

- Accuracy: 0.804
- Weighted Precision: 0.801
- Weighted Recall: 0.804
- Weighted F1: 0.802

## 9.3 Confusion Matrix

(Rows = True, Cols = Pred):

| | **Bach** | **Beethoven** | **Chopin** | **Mozart** |
|---|---|---|---|---|
| **Bach** | 86 | 1 | 1 | 5 |
| **Beethoven** | 3 | 10 | 3 | 5 |
| **Chopin** | 0 | 4 | 10 | 0 |
| **Mozart** | 5 | 3 | 0 | 17 |

## 10 Error Analysis

- **Beethoven–Mozart Confusion:** High misclassification rate between these composers, likely due to shared Classical-era compositional structures (e.g., balanced phrasing, diatonic harmony).
- **Chopin–Beethoven Overlap:** Certain Chopin pieces misclassified as Beethoven, possibly driven by similar harmonic progressions and use of modulations in late works.
- **Model Behavior:** CNN effectively captured local harmonic/melodic motifs but underperformed on long-range phrasing, an area where LSTM's temporal memory improved classification accuracy.

## 11 Challenges and Solutions

i. **Label Alignment for Ensembles**
   - Mismatch between CNN and LSTM output class order caused incorrect averaging. Fixed by enforcing a shared label mapping.

ii. **CPU Bottleneck**
   - Long training times on CPU. Solution: reduced batch size and introduced early stopping.

iii. **Class Imbalance**
   - Beethoven & Chopin underrepresented → solved with targeted pitch shifting.

## 12 Conclusion

After evaluating multiple architectures and ensemble strategies, the **Logit-Averaged Ensemble (Pitch+Tempo Augmentation)** emerged as the best-performing approach, achieving a **test accuracy of 80.4%**, with a **weighted F1-score of 0.806**.

| Model | Augmentation | Ensemble Method | Accuracy | Weighted F1 | Macro F1 | Key Takeaways |
|---|---|---|---|---|---|---|
| **CNN** | Pitch shift | None | 0.784 | 0.791 | 0.686 | Strong harmonic capture but lacked temporal context. |
| **CNN** | Pitch + tempo | None | 0.784 | 0.783 | 0.676 | Improved tempo robustness, minimal accuracy change. |
| **CNN + LSTM** | Pitch shift | Softmax avg | 0.804 | 0.811 | 0.715 | Added temporal modeling improved recall for minority classes. |
| **Weighted Ensemble** | Pitch shift | Weighted softmax (α=0.85) | 0.791 | 0.796 | 0.698 | Class weights improved recall, but fusion method limited gains. |
| **Weighted Ensemble** | Pitch + tempo | Weighted softmax (α=0.70) | 0.804 | 0.802 | 0.712 | Balanced tradeoff between precision and recall. |

| | | | | | | |
|---|---|---|---|---|---|---|
| **Logit-Averaged Ensemble** | Pitch + tempo | Weighted logit avg (α=0.50) | **0.804** | **0.806** | **0.714** | Best performer: leveraged both CNN and LSTM strengths, maintaining class separation. |

**Performance Summary:**

- **Accuracy:** 80.4%
- **Weighted Precision:** 0.812
- **Weighted Recall:** 0.804
- **Weighted F1-Score:** 0.806
- **Macro F1-Score:** 0.714
- **Notable Strength:** High precision for Bach (0.933) and balanced recall across minority classes like Chopin and Mozart.

**Reasons for Superior Performance:**

1. **Complementary Strengths of CNN and LSTM:**
    - CNN captured short-term harmonic and melodic motifs.
    - LSTM modeled long-range temporal dependencies and phrasing.
    - Combined via logit-level averaging, the ensemble benefited from both local and global feature representations.
2. **Pitch + Tempo Augmentation:**
    - Improved robustness to variations in key and performance speed.
    - Reduced overfitting and improved minority-class recall, particularly for Beethoven and Chopin.
3. **Logit-Averaging vs. Softmax Averaging:**
    - Operating on pre-softmax activations preserved more class separation information, leading to better calibrated predictions and higher macro-average scores.

**Key Observations from Confusion Matrix:**

- **Bach** achieved the highest per-class accuracy and recall, benefiting from distinctive Baroque polyphonic structures.
- **Beethoven–Mozart Confusion** remained a challenge due to overlapping Classical-era characteristics.
- **Chopin** improved recall compared to single models, reducing misclassification as Beethoven.

## CONCLUSION:

The **Logit-Averaged CNN+LSTM Ensemble** offers the most balanced and accurate classification performance, demonstrating that combining different model architectures with well-chosen augmentation strategies yields significant gains over individual models.

Ensembling CNN and LSTM with augmentation improves composer classification. The **best model** achieved 80.4% accuracy and balanced precision-recall performance. CNN captures harmonic structure, LSTM captures temporal dependencies — their combination provides complementary benefits.

## 13 VISUALIZATIONS

- Class distribution before vs after augmentation (TRAIN split)

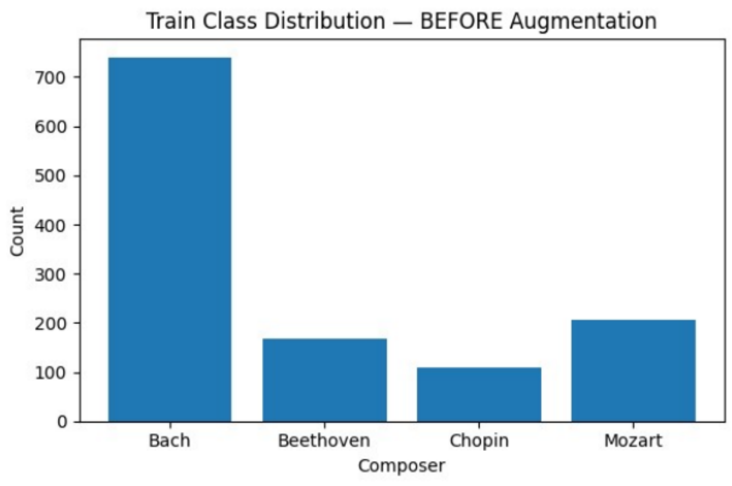

*Figure 1: Train class before augmentation*

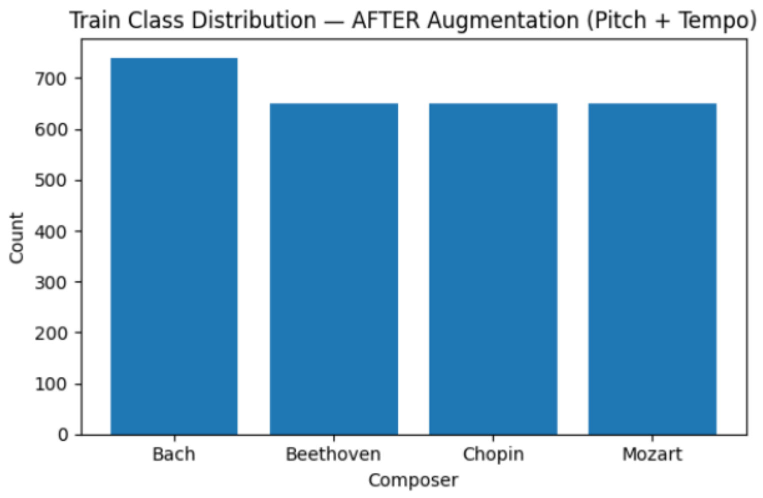

*Figure 2: Train class distribution after augmentation*

- Confusion matrices for CNN (Pitch+Tempo), LSTM, and Weighted Ensemble

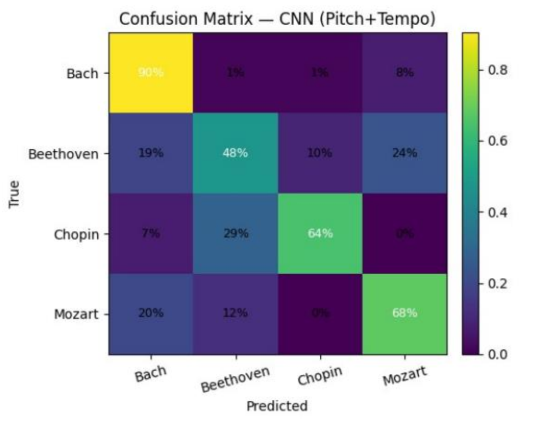

*Figure 5: Confusion Matrix - CNN*

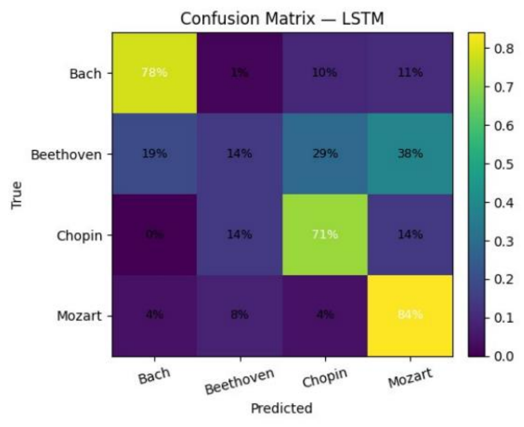

*Figure 4: Confusion Matrix - LSTM*

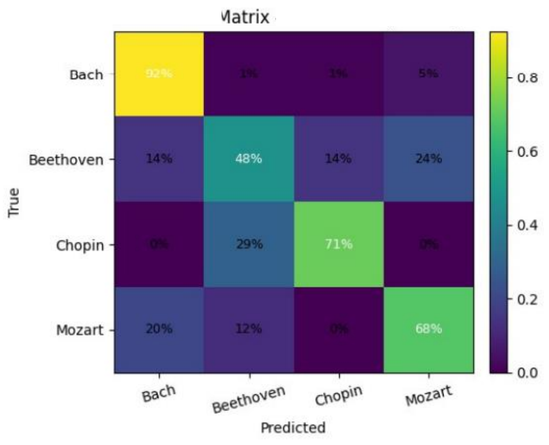

*Figure 3: Weighted Ensemble*

- Per-class F1 comparison across models

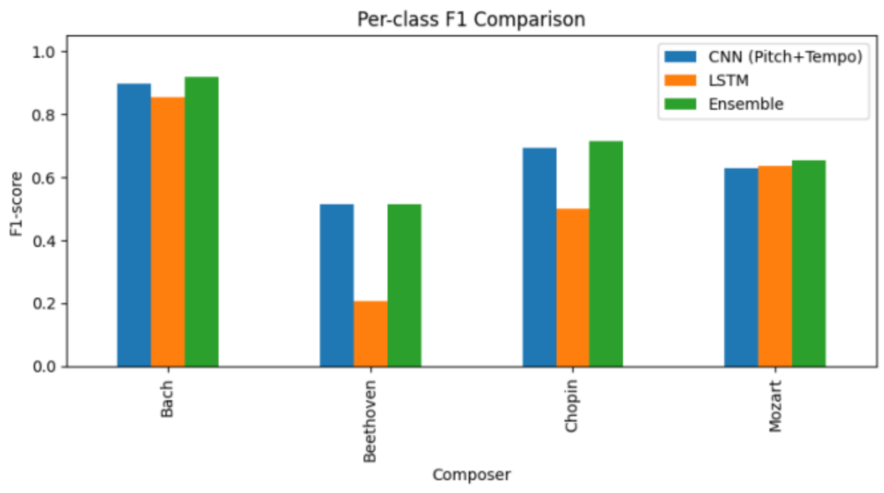

*Figure 6: Per Class F1 Comparison*

- Ensemble weight tuning curve on VAL (softmax vs logit averaging)

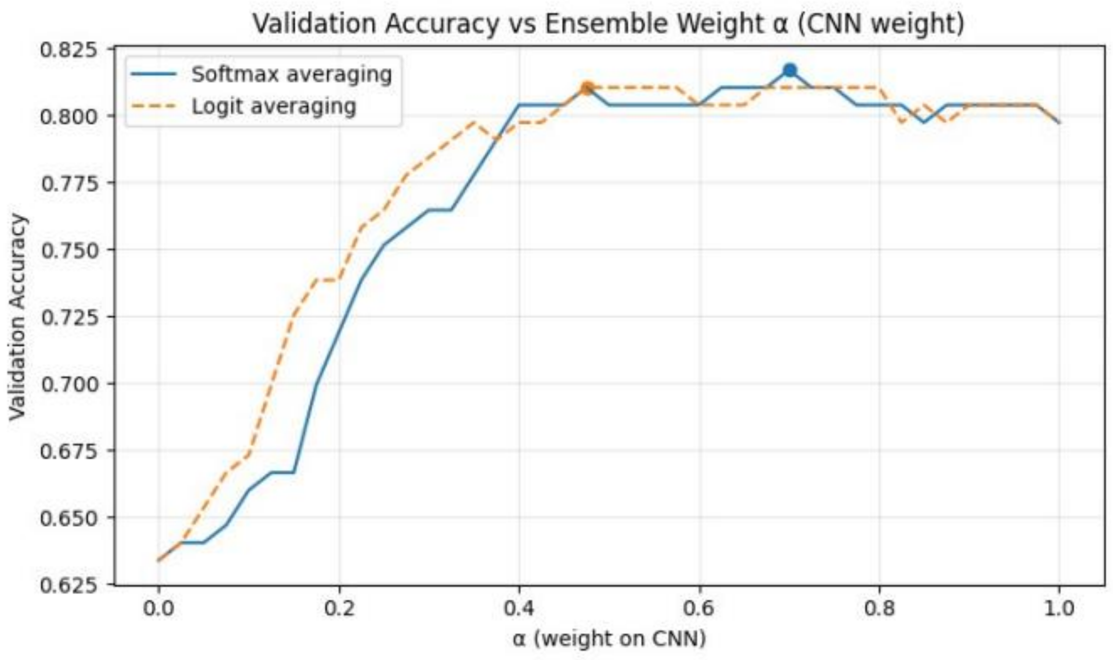

*Figure 7: Sofmax vs logit averaging*

- t-SNE of CNN penultimate features on TEST set

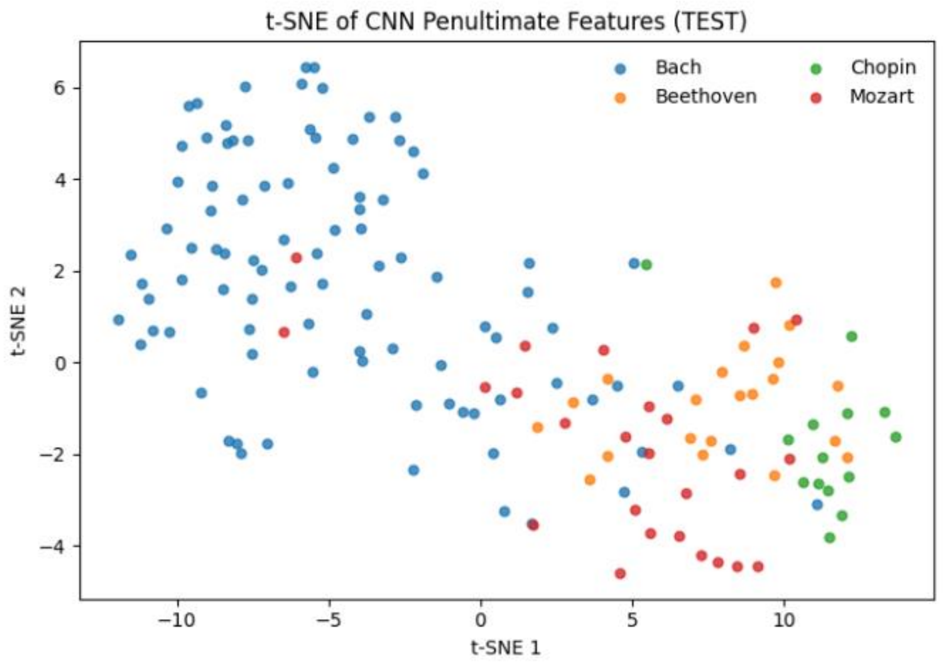

*Figure 8: t-SNE of CNN penultimate features*

- Results summary — test accuracy across models

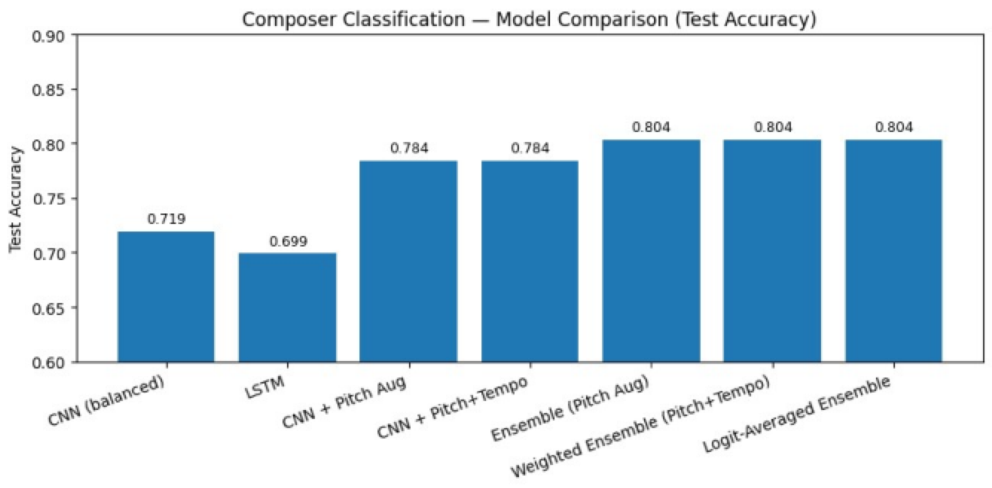

*Figure 9: Composer classification - model comparison*

- Precision / Recall / F1 (weighted) across key models

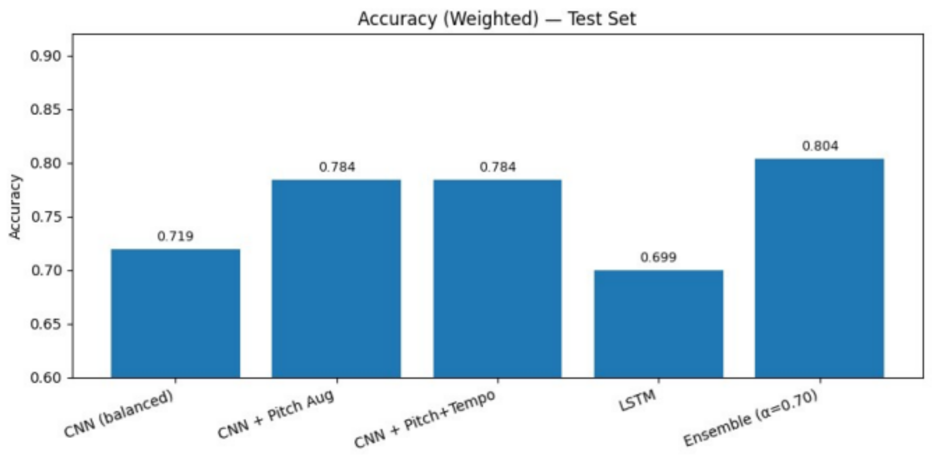

*Figure 10: Accuracy weighted - Test set*

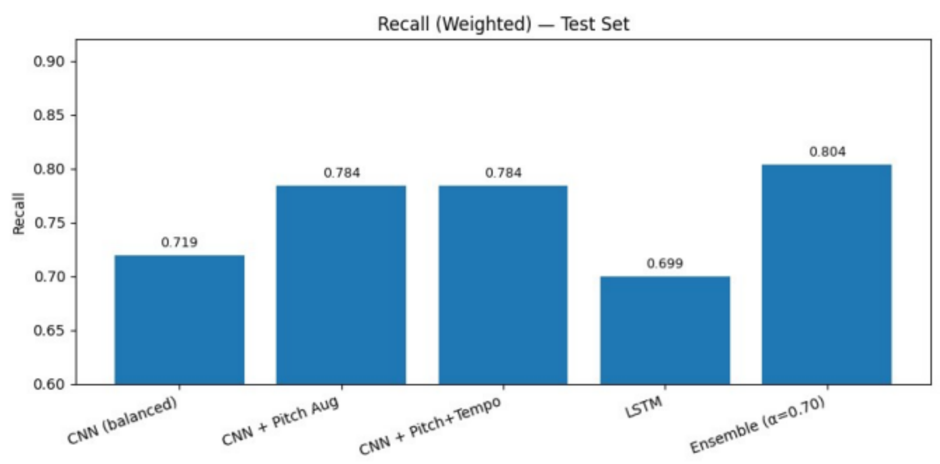

*Figure 11: Recall - Test set*

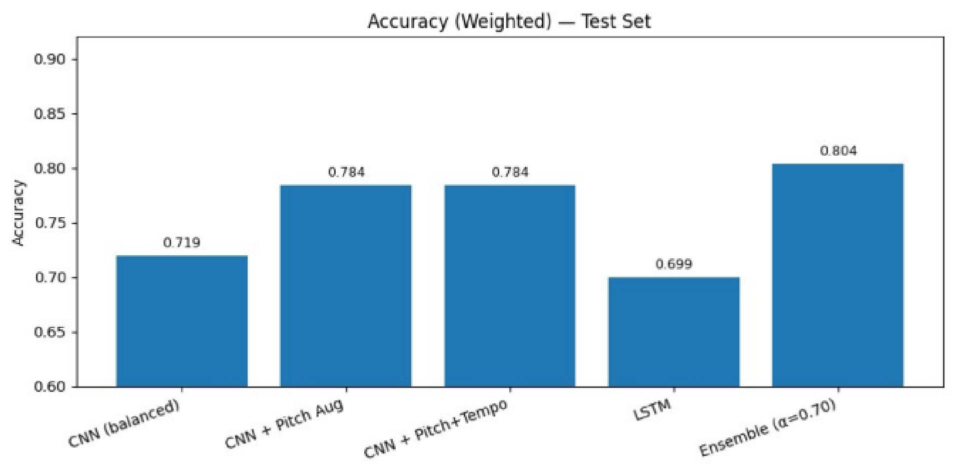

*Figure 12: Accuracy - Test set*

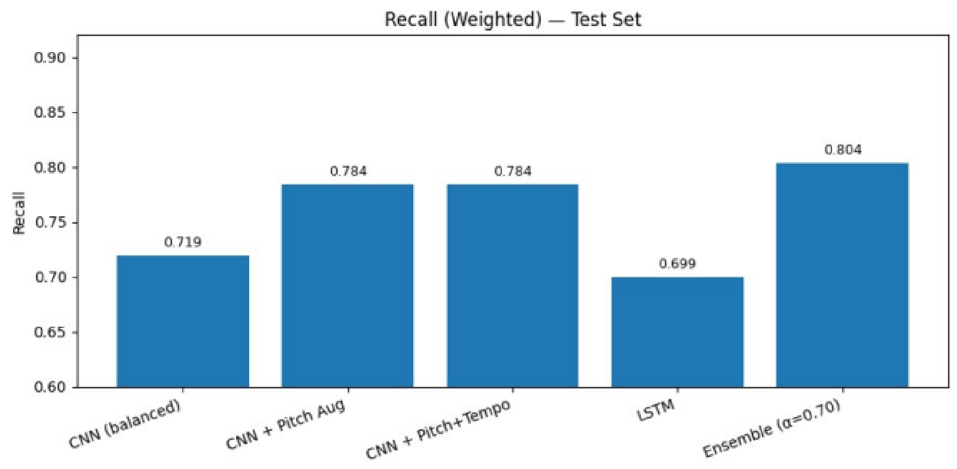

*Figure 13: Recall - Test set*

## 14 Future Directions

- Transformer-based symbolic music models.
- Multi-modal learning combining MIDI and audio.
- Larger datasets across more composers.

## 15 References

- Chollet, F. (2017). *Deep Learning with Python*. Manning Publications.
- Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.
- Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(8), 1735–1780.
- LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, 521(7553), 436–444. https://doi.org/10.1038/nature14539
- Kingma, D. P., & Ba, J. (2015). Adam: A method for stochastic optimization. *International Conference on Learning Representations (ICLR)*. https://arxiv.org/abs/1412.6980
- Raffel, C., & Ellis, D. P. W. (2016). Intuitive analysis, creation, and manipulation of MIDI data with pretty_midi. *ISMIR*, 84–90.
- Humphrey, E. J., Bello, J. P., & LeCun, Y. (2013). Feature learning and deep architectures: New directions for music informatics. *Journal of Intelligent Information Systems*, 41(3), 461–481. https://doi.org/10.1007/s10844-013-0248-5
- Müller, M. (2015). *Fundamentals of Music Processing: Audio, Analysis, Algorithms, Applications*. Springer. https://doi.org/10.1007/978-3-319-21945-5
- OpenAI. (2025). *ChatGPT*. Retrieved from https://openai.com/chatgpt