SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 of the UGC Act, 1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA
THANJAVUR | KUMBAKONAM | CHENNAI

DEEP LEARNING-BASED AUTOMATIC MUSIC TRANSCRIPTION USING CR-GCN

Project Guide – Dr. Emily Jenifer A

By

126003238 Selvakarthik S
126003241 Shanthosh Kumar R
126003218 Rithvik L

MOTIVATION

- Music is a major stress-buster for billions around the world. While we listen to millions of songs each year, certain tunes stay with us in our minds.
- Musicians often transcribe these melodies into music sheets to aid in composition and performance.
- Our goal is to convert a piece of music into its corresponding notes using a CR-GCN model.
- To make this process accessible to everyone, we plan to develop a web application where users can upload a music file, and our model will generate the corresponding music notes, which will be displayed as a music sheet within the application.

OBJECTIVE

- To develop an approach to convert an audio file (music) into its corresponding musical notes.
- To identify and leverage efficient feature selection algorithms.
- To select and utilise suitable classification algorithms.
- To construct a model with the highest possible accuracy through rigorous training and testing on large datasets.
- To deploy the model as a back-end in an user-friendly web application.

BASE PAPER

- [1] Xiao, Z., Chen, X., & Zhou, L. (2023). Polyphonic piano transcription based on graph convolutional network. Signal Processing, 212, 109134.
- Doi : https://doi.org/10.1016/j.sigpro.2023.109134

PROBLEM STATEMENT

- To design an Automatic Music Transcription (AMT) system that accurately converts complex polyphonic audio signals into symbolic music representations by capturing note interdependencies and temporal dynamics.
- To deploy the model as a back-end in an user-friendly web application.

ABSTRACT

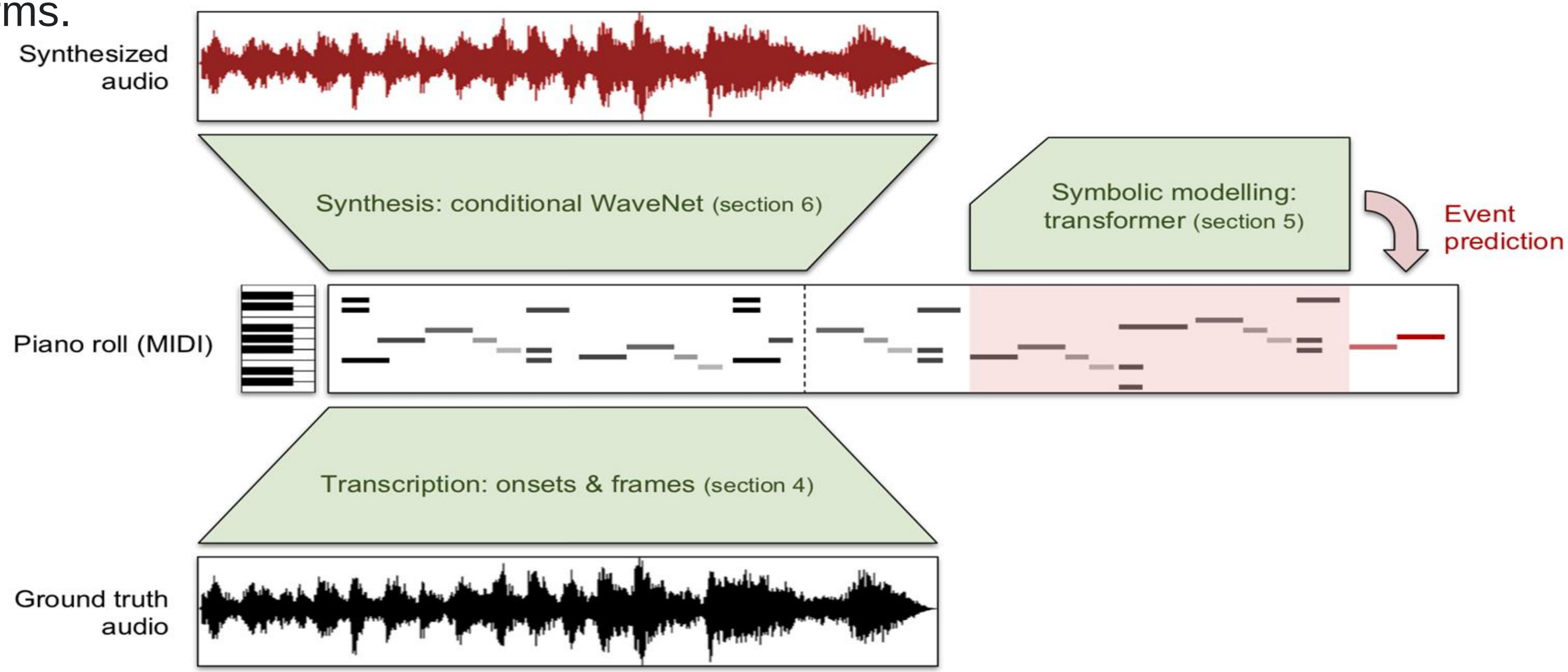
- The task of automatic music transcription (AMT) mainly focuses on converting audio signals to symbolic music representations, facilitating applications such as computational musicology and music analysis.
- One of the biggest problems is when multiple notes are played at the same time, dimension explosion can happen which makes it difficult for accurate music note transcription.
- To overcome this challenge, we have proposed a hybrid deep learning architecture combining Convolutional Neural Network for spatial feature extraction, bidirectional LSTMs or self-attention mechanisms for precise temporal note-level predictions and Graph Convolutional Network for accurate label learning to capture note interdependencies in polyphonic music.
- Experiments on public datasets like MAESTRO, MAPS, GiantMIDI show that the proposed methodology with F1-score of 96.88% is much more superior than existing methodologies like Onset and Frames, Wavenet, Non-Negative Matrix Factorization (NMF).
- The generated music sheets validate the model’s accuracy and practical applicability, providing a valuable tool for musicians and researchers.
- By addressing the limitations of prior methods, the proposed approach CR-GCN (Channel Relationship-Based Graph Convolutional Network) represents a step forward in automated transcription technology, making it feasible for large-scale and real-time applications.

LITERATURE

TITLE	MERITS	DEMERITS
A Data-Driven Analysis of Robust Automatic Piano Transcription	The study improved note-onset accuracy to 88.4 F1-score on the MAPS dataset through data augmentation.	Performance on out-of-distribution annotated piano data indicates challenges in generalizing to unseen data.
Automatic Piano Sheet Music Transcription with Machine Learning	BiLSTM architecture is identified as one of the effective for automatic piano music transcription, achieving a top F1-score of 74.80%.	CNNs underperform significantly in music transcription tasks, achieving only an F1-score of 22.85% despite extensive hyper-parameter tuning.
Research on the Recognition of Piano-Playing Notes by a Music Transcription Algorithm	The CRNN algorithm combines CNN and BiLSTM, achieving impressive F1-scores of 84.90%, 92.24%, and 79.27% for frames, notes, and offsets.	The study's results have limited generalizability due to small datatset, with further exploration of different piano note features needed to enhance recognition accuracy.
Multimodal Image and Audio Music Transcription	The multimodal framework combining OMR and AMT improves transcription accuracy by reducing errors up to 40% for more accurate music transcription.	The proposed framework can degrade overall transcription accuracy if either OMR or AMT already performs near-perfectly. Overall Transcription may degrade
Automatic Piano Music Transcription Using Audio-Visual Features	This research uses audio-visual features to improve piano music transcription accuracy by 12.69%, surpassing audio-only systems.	The system's dependency on specialized equipment, like an overhead camera, limits its practicality for general use

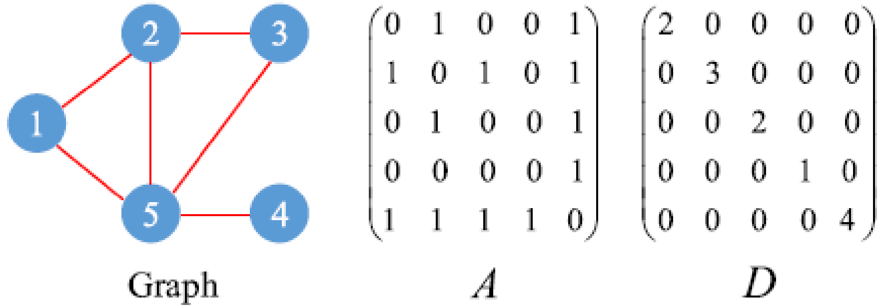
DATASET PREPARATION

- **Datasets Used**:
  - **The MAESTRO Dataset**
    - MAESTRO (MIDI and Audio Edited for Synchronous TRacks and Organization) is a dataset composed of about 200 hours of virtuosic piano performances.
    - The audio files are captured with fine alignment (~3 ms) between note labels and audio waveforms.



EXISTING SYSTEM

- **Key Features:**
  - Combination of CNN, RNN, GCN
  - Two Stage Learning - Feature Learning and Label Learning
  - Graph Representation of Notes
  - Mapped Classifier with Dot Product



- **CR-GCN (Channel Relationship-Based Graph Convolutional Network)**
  - CNN for spatial features, LSTM for temporal dependencies, and GCN for modeling relationships between notes in a graph-based format.
- **Performance:**
  - **Datasets & Accuracies:**
    - MAESTRO (v2) : Precision - 97.38%, Recall - 96.21%, F1-Score - 96.88%
    - MAPS Dataset : Precision - 84.30%, Recall - 84.65%, F1-Score - 84.48%
- **Advantages:**
  - Joint Feature and Label Learning
  - Scalable to Multi-Label Tasks
  - Improved Performance for Complex Music and Generalizable to Other Domains

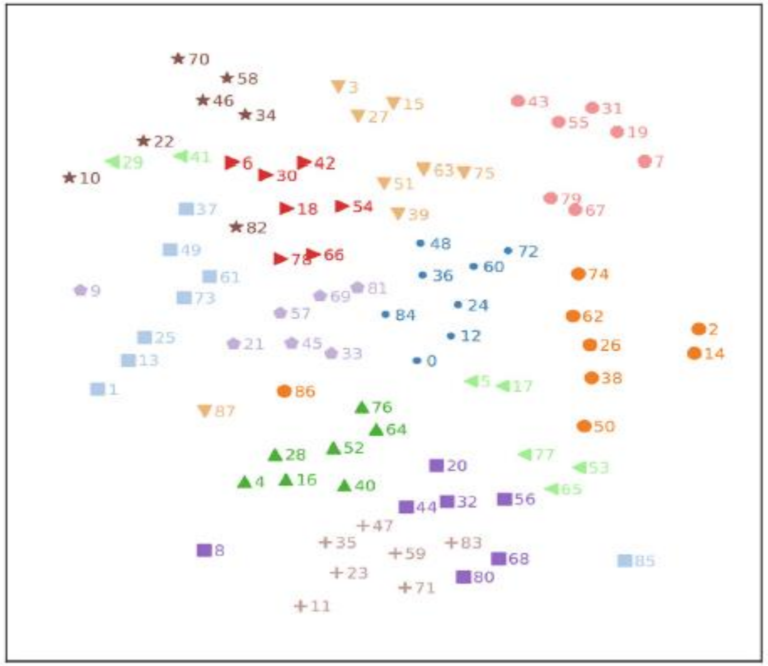
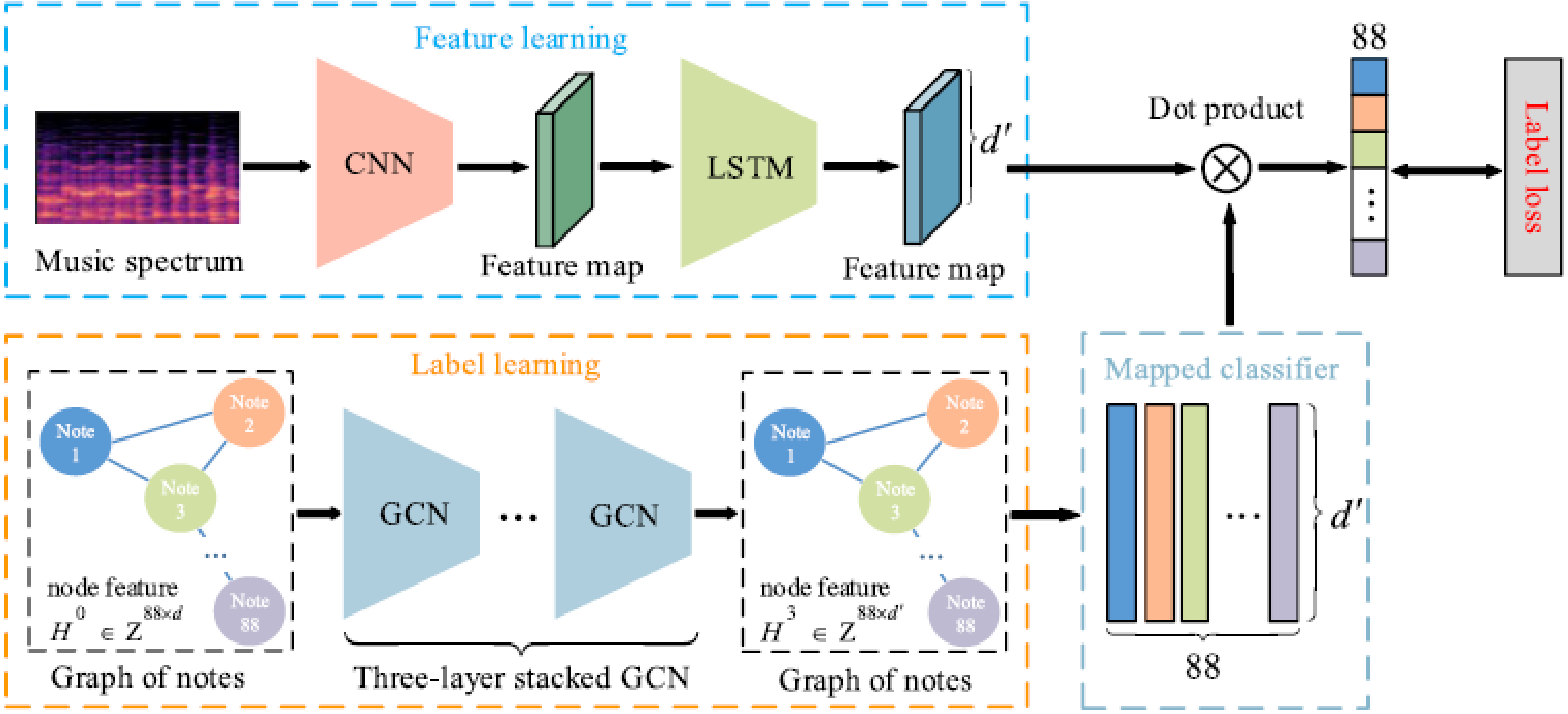


Fig. 8. Visualization of the CR-GCN model based on t-SNE.

EXISTING SYSTEM



LIMITATIONS OF EXISTING SYSTEM

1. Graph Convolutional Networks (GCN)

- **Complexity** : Requires a well-defined graph structure, which can be challenging to construct for audio data like polyphonic music.
- **Scalability Issues** : As the number of notes or features increases, the graph size grows, leading to computational inefficiency.
- **Limited Temporal Modeling** : GCNs are not inherently designed to capture temporal dependencies, which are crucial in music transcription.

2. Convolutional Neural Networks (CNN)

- **Lack of Temporal Context**: CNNs excel at spatial feature extraction but struggle with capturing temporal dynamics in music, such as note transitions.
- **Overfitting Risk**: With limited training data, CNNs can overfit, especially when dealing with complex polyphonic music.
- **Fixed Receptive Field**: The fixed kernel size may miss long-range dependencies in music sequences.

LIMITATIONS OF EXISTING SYSTEM

3. Long Short-Term Memory Networks (LSTM)

- **Vanishing Gradient Problem**: Despite being designed to handle long-term dependencies, LSTMs can still face challenges with very long sequences.
- **High Computational Cost**: LSTMs are resource-intensive, making them slower to train compared to other models.
- **Sensitivity to Input Representation**: The performance of LSTMs heavily depends on how the input (e.g., spectrograms or MIDI data) is preprocessed.

4. Some other Limitations

- Data Dependence and Annotation Challenges
- Model Interpretability and Debugging
- Real-Time Processing and Computational Overhead

SPECIFICATIONS AND REQUIREMENTS

Hardware Requirements

1. **Processor**: A multi-core processor (e.g., Intel i5/i7 or AMD Ryzen 5/7) for efficient computation.
2. **RAM**: At least 8 GB (16 GB recommended) to handle large datasets and model training.
3. **Storage**: SSD with at least 256 GB for faster data access and storage of audio files, models, and results.
4. **GPU**: A dedicated GPU (e.g., NVIDIA GTX 1650 or higher) for training deep learning models.
5. **Audio Interface**: Optional, for high-quality audio input/output if you plan to work with live recordings.

- **Processor Used** : Intel(R) Xeon(R) Silver 4310 CPU @ 2.10GHz
- **GPU Used** : NVIDIA A100 Tensor Core GPU

SPECIFICATIONS AND REQUIREMENTS

Software Requirements

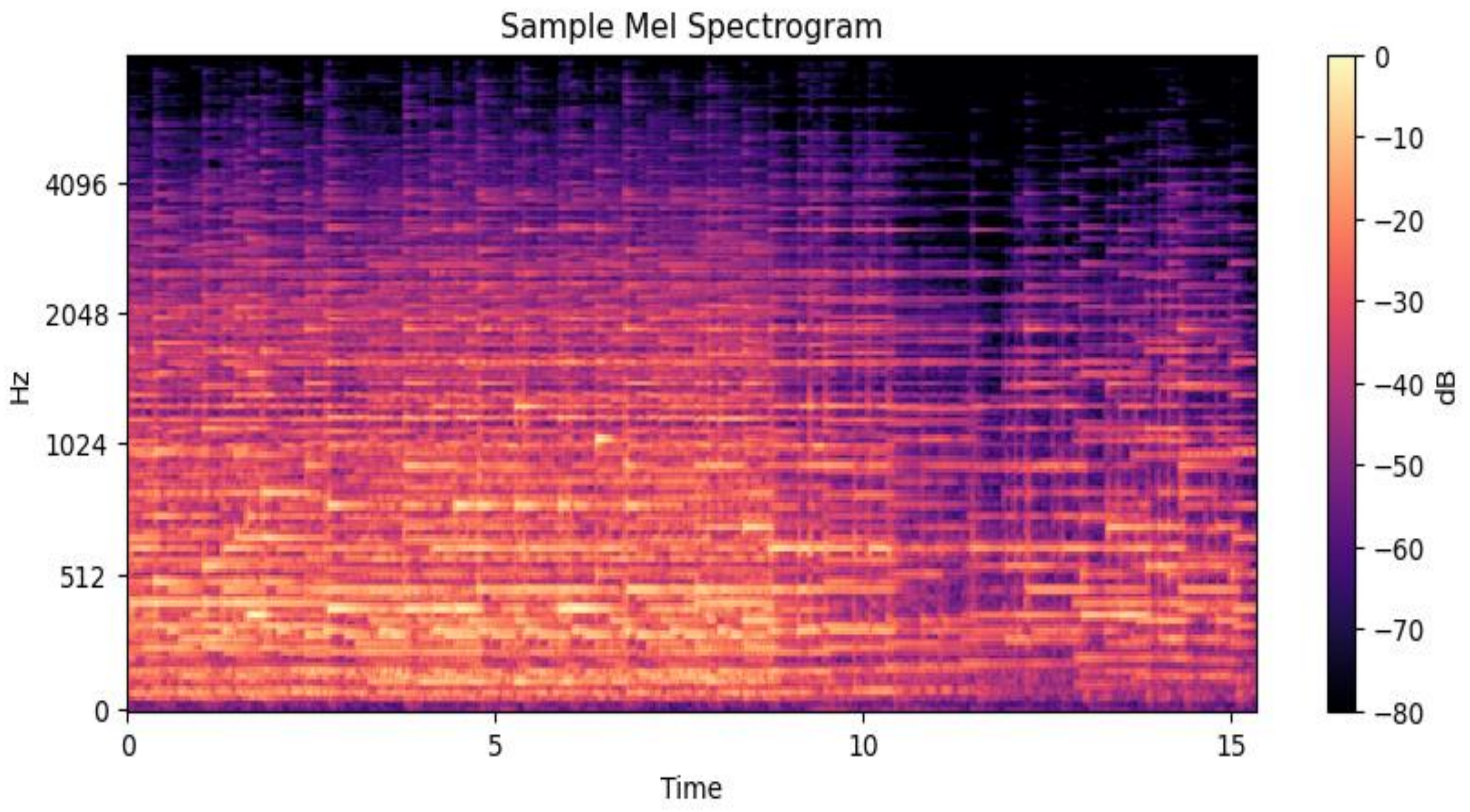
1. **Programming Language**: Python (widely used for machine learning and audio processing).
2. **Libraries/Frameworks**:
    - **TensorFlow** or **PyTorch**: For building and training neural networks.
    - **Librosa**: For audio analysis and feature extraction.
    - **Music21**: For music theory and symbolic music processing.
    - **MIDI Libraries**: Such as mido or pretty_midi for handling MIDI files.
    - **pdf2image**: To make images of Music Sheets from pdf format.
    - **lilypond**: To make pdf of Music Sheets from the MIDI notes.
    - **numpy**: Used for scientific and numerical computing like arrays , matrices , data analysis.
3. **Audio Tools**:
    - **Audacity**: For audio editing and preprocessing.
    - **FFmpeg**: For audio format conversion.
4. **Operating System**: Windows, macOS, or Linux (Linux is often preferred for machine learning projects).
5. **IDE/Code Editor**: Visual Studio Code, Jupyter Notebook, or PyCharm for coding and debugging.

MODULE

- ➢ **MODULE 1:** DATA PREPROCESSING AND DATA SPLITTING
- ➢ **MODULE 2:** FEATURE LEARNING USING CNN + LSTM
- ➢ **MODULE 3:** LABEL LEARNING USING GCN
- ➢ **MODULE 4:** RESULTS AND OBSERVATION (so far)
- ➢ **MODULE 5:** DEPLOYMENT

MODULE 1: DATA PREPROCESSING

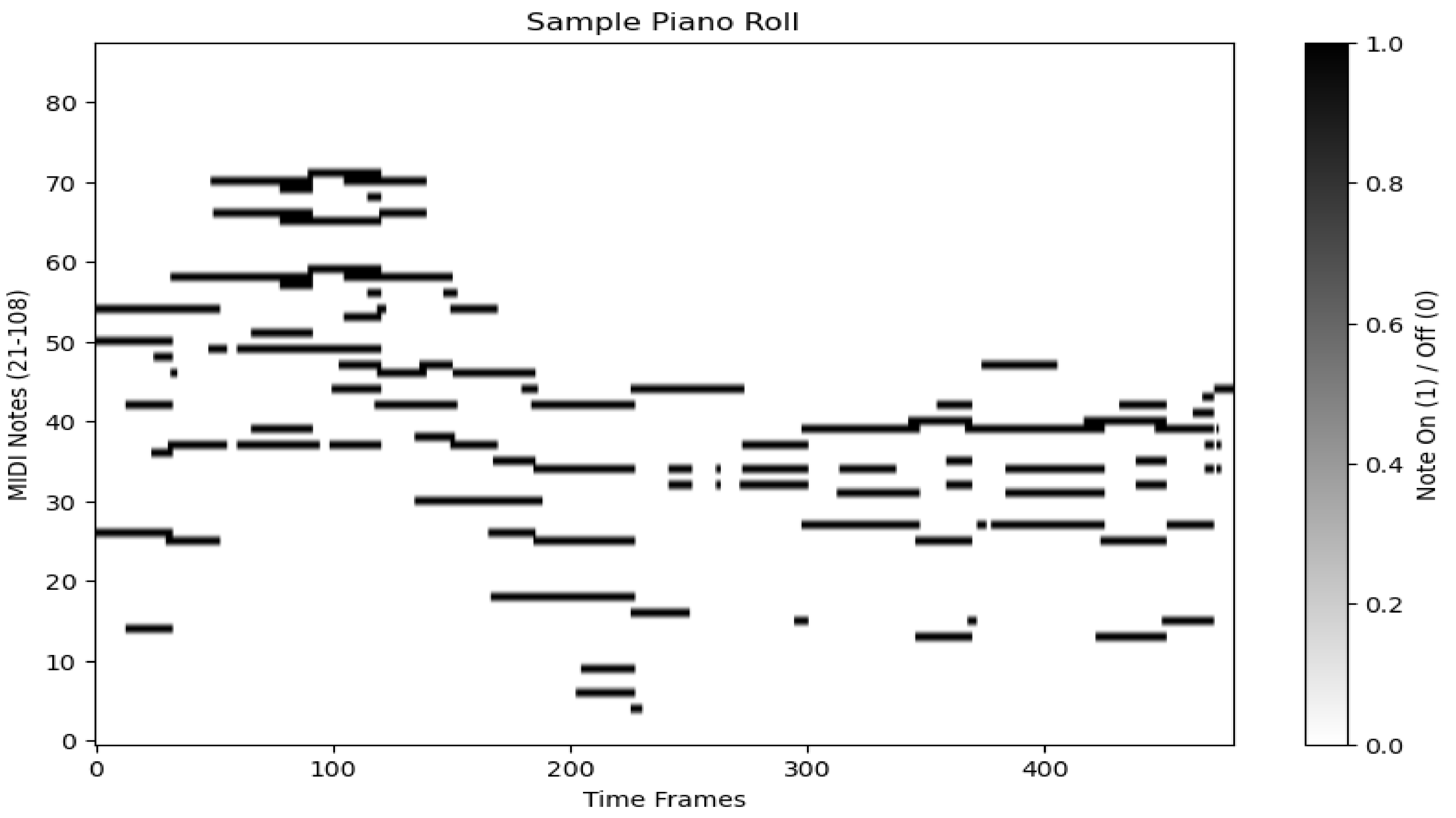
- As per the dataset we have incorporated into our project (Maestro v3), the categorical X attribute consists of .wav audio files.
- To enhance performance of our model, it is required to convert the .wav files into its corresponding mel-spectrograms which is achieved using the librosa module of python.
- A mel spectrogram is a visual representation of an audio signal's frequency content over time, using the mel scale on the y-axis and decibel scale for amplitude, making it more aligned with human auditory perception.
- The mel-spectrogram consists of the frequency (in hz) in y-axis and time in x-axis (in s) and also the sound-level (in db) is depicted as a third parameter with variation in colors.
- Mel spectrograms provide an estimate of the short-term, time-localized frequency content of an audio signal, making them useful for analyzing how audio patterns change over short intervals of time.



Sample Mel Spectrogram
Hz
4096
2048
1024
512
0
Time
0
5
10
15
dB
0
−10
−20
−30
−40
−50
−60
−70
−80

MODULE 1: DATA PREPROCESSING

- As per the dataset we have incorporated into our project (Maestro v3), the categorical Y attribute consists of .midi files.
- To enhance performance of our model, it is required to convert the .midi files into its corresponding piano rolls which is achieved using the pretty_midi module of python.
- A piano roll is a visual representation of musical notes over time, where notes are depicted as horizontal lines or bars on a grid, with the vertical axis representing pitch and the horizontal axis representing time.
- The piano roll consists of the midi notes (from 21-108) in y-axis and time in x-axis (in s) and also whether the note is pressed or not is depicted as a third parameter with variation in hues.
- Piano rolls can also convey additional information such as note duration and velocity, which can be represented through variations in line length and color intensity, respectively, providing a comprehensive view of musical performance dynamics.



Sample Piano Roll
MIDI Notes (21-108)
Time Frames
Note On (1) / Off (0)
0
10
20
30
40
50
60
70
80
0
100
200
300
400
0.0
0.2
0.4
0.6
0.8
1.0

MODULE 1: DATA PREPROCESSING

```python
# Preprocessing functions
def preprocess_wav_to_mel(wav_path, midi_duration):
    audio, _ = librosa.load(str(wav_path), sr=sr)
    # Trim audio to MIDI duration
    max_samples = int(midi_duration * sr)
    audio = audio[:max_samples] if len(audio) > max_samples else audio
    S = librosa.feature.melspectrogram(y=audio, sr=sr, n_fft=n_fft, hop_length=hop_length, n_mels=n_mels)
    S_db = librosa.power_to_db(S, ref=np.max)
    expected_T = (len(audio) - n_fft) // hop_length + 1
    if S_db.shape[1] > expected_T:
        S_db = S_db[:, :expected_T]
    return S_db

def preprocess_midi_to_piano_roll(midi_path, T_target, frame_rate):
    midi = pretty_midi.PrettyMIDI(str(midi_path))
    piano_roll = midi.get_piano_roll(fs=frame_rate)
    piano_roll = piano_roll[21:109, :]  # 88 pitches
    if piano_roll.shape[1] < T_target:
        padding = np.zeros((n_pitches, T_target - piano_roll.shape[1]), dtype=np.uint8)
        piano_roll = np.hstack((piano_roll, padding))
    elif piano_roll.shape[1] > T_target:
        piano_roll = piano_roll[:, :T_target]
    piano_roll = (piano_roll > 0).astype(np.uint8)
    return piano_roll

# Segment data into fixed-length chunks
def segment_data(data, segment_length, axis=1):
    T = data.shape[axis]
    if T < segment_length:
        pad_width = [(0, 0)] * len(data.shape)
        pad_width[axis] = (0, segment_length - T)
        return [np.pad(data, pad_width, mode='constant', constant_values=0)]
    segments = []
    for start in range(0, T, segment_length):
        end = min(start + segment_length, T)
        segment = data[:, start:end] if axis == 1 else data[start:end, :]
```

Output:

```
Processing file 427/1276: MIDI-Unprocessed_07_R2_2009_01_ORIG_MID--AUDIO_07_R2_2009_07_R2_2009_01_WAV.wav (from 2009)
  Saved 32 segments
Processing file 428/1276: MIDI-Unprocessed_07_R2_2009_01_ORIG_MID--AUDIO_07_R2_2009_07_R2_2009_02_WAV.wav (from 2009)
  Saved 31 segments
Processing file 429/1276: MIDI-Unprocessed_07_R2_2009_01_ORIG_MID--AUDIO_07_R2_2009_07_R2_2009_03_WAV.wav (from 2009)
  Saved 11 segments
Processing file 430/1276: MIDI-Unprocessed_07_R2_2009_01_ORIG_MID--AUDIO_07_R2_2009_07_R2_2009_04_WAV.wav (from 2009)
  Saved 34 segments
Processing file 431/1276: MIDI-Unprocessed_08_R1_2009_01-04_ORIG_MID--AUDIO_08_R1_2009_08_R1_2009_01_WAV.wav (from 2009)
  Saved 44 segments
Processing file 432/1276: MIDI-Unprocessed_08_R1_2009_01-04_ORIG_MID--AUDIO_08_R1_2009_08_R1_2009_02_WAV.wav (from 2009)
  Saved 22 segments
Processing file 433/1276: MIDI-Unprocessed_08_R1_2009_01-04_ORIG_MID--AUDIO_08_R1_2009_08_R1_2009_03_WAV.wav (from 2009)
  Saved 16 segments
Processing file 434/1276: MIDI-Unprocessed_08_R1_2009_01-04_ORIG_MID--AUDIO_08_R1_2009_08_R1_2009_04_WAV.wav (from 2009)
  Saved 18 segments
Processing file 435/1276: MIDI-Unprocessed_08_R1_2009_05-06_ORIG_MID--AUDIO_08_R1_2009_08_R1_2009_06_WAV.wav (from 2009)
  Saved 80 segments
Processing file 436/1276: MIDI-Unprocessed_08_R2_2009_01_ORIG_MID--AUDIO_08_R2_2009_08_R2_2009_01_WAV.wav (from 2009)
  Saved 51 segments
Processing file 437/1276: MIDI-Unprocessed_08_R2_2009_01_ORIG_MID--AUDIO_08_R2_2009_08_R2_2009_02_WAV.wav (from 2009)
  Saved 27 segments
Processing file 438/1276: MIDI-Unprocessed_08_R2_2009_01_ORIG_MID--AUDIO_08_R2_2009_08_R2_2009_03_WAV.wav (from 2009)
  Saved 15 segments
...
  Saved 61 segments
Processing file 484/1276: MIDI-Unprocessed_15_R1_2009_03-06_ORIG_MID--AUDIO_15_R1_2009_15_R1_2009_05_WAV.wav (from 2009)
  Saved 25 segments
Processing file 485/1276: MIDI-Unprocessed_15_R1_2009_03-06_ORIG_MID--AUDIO_15_R1_2009_15_R1_2009_06_WAV.wav (from 2009)
```

MODULE 1: DATA SPLITTING

```python
# Step 1: Collect all Mel spectrogram files and group by original file
mel_files = sorted(list(mel_input_dir.glob("*_mel.npy")))
print(f"Total Mel spectrogram segments: {len(mel_files)}")

# Group segments by original file (based on filename before "_segXXXX_mel.npy")
file_groups = defaultdict(list)
for mel_file in mel_files:
    # Extract the base filename (e.g., "MIDI-Unprocessed_SMF_02_R1_2004_01-05_ORIG_MID--AUDIO
    base_name = mel_file.stem.split("_seg")[0]
    file_groups[base_name].append(mel_file)

# Step 2: Split files into train, val, test (80-10-10)
file_names = list(file_groups.keys())
np.random.seed(42)  # For reproducibility
np.random.shuffle(file_names)

total_files = len(file_names)
train_split = int(0.8 * total_files)  # 80%
val_split = int(0.1 * total_files)    # 10%
test_split = total_files - train_split - val_split  # Remaining 10%

train_files = file_names[:train_split]
val_files = file_names[train_split:train_split + val_split]
test_files = file_names[train_split + val_split:]

print(f"Total files: {total_files}")
print(f"Train files: {len(train_files)} ({len(train_files)/total_files*100:.1f}%)")
print(f"Val files: {len(val_files)} ({len(val_files)/total_files*100:.1f}%)")
print(f"Test files: {len(test_files)} ({len(test_files)/total_files*100:.1f}%)")
```

Output:

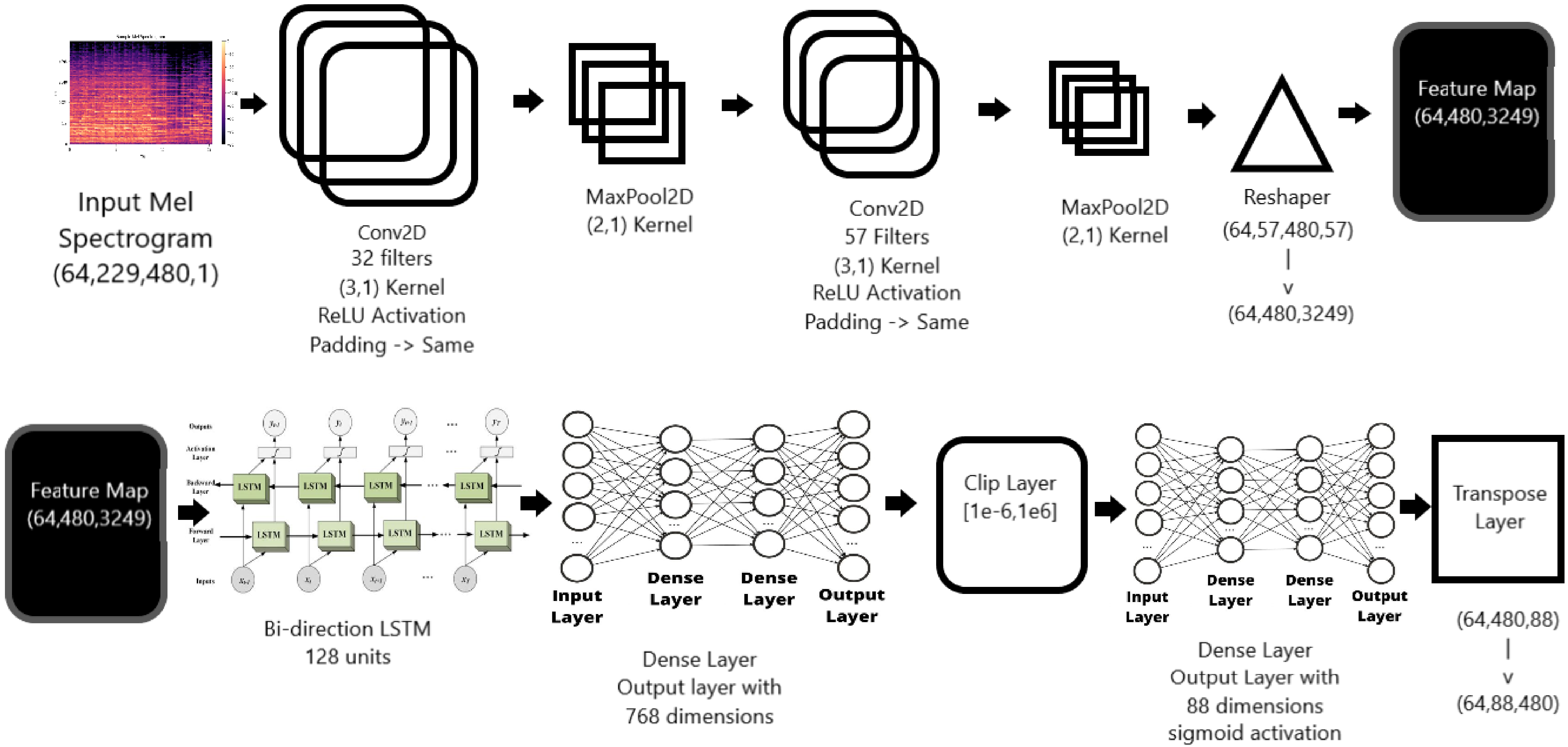
```
Total Mel spectrogram segments: 47268
Total files: 1276
Train files: 1020 (79.9%)
Val files: 127 (10.0%)
Test files: 129 (10.1%)

Moved to train:
  Mel spectrograms: 37993 segments
  Piano-rolls: 37993 segments

Moved to val:
  Mel spectrograms: 4113 segments
  Piano-rolls: 4113 segments

Moved to test:
  Mel spectrograms: 5162 segments
  Piano-rolls: 5162 segments
```

MODULE 2: FEATURE LEARNING CNN + LSTM



MODULE 2: FEATURE LEARNING CNN + LSTM

```python
# Build CNN+LSTM branch
def build_cnn_lstm():
    inputs = layers.Input(shape=(n_mels, frames_per_segment, 1))
    x = layers.Conv2D(32, (3, 1), activation='relu', padding='same')(inputs)
    x = layers.MaxPooling2D((2, 1))(x)
    x = layers.Conv2D(57, (3, 1), activation='relu', padding='same')(x)
    x = layers.MaxPooling2D((2, 1))(x)
    x = layers.Reshape((frames_per_segment, -1))(x)
    x = layers.Bidirectional(layers.LSTM(128, return_sequences=True))(x)
    x = layers.Dropout(0.5)(x)
    features = layers.Dense(d_prime, activation=None)(x)
    features = ClipLayer(min_value=-1e6, max_value=1e6)(features)
    pred = layers.Dense(n_pitches, activation='sigmoid')(features)
    pred = TransposeLayer(perm=[0, 2, 1])(pred)
    return Model(inputs=inputs, outputs={'cnn_lstm_features': features, 'cnn_lstm_pred': pred}, name="cnn_lstm")
```

Total params: 3,730,133 (14.23 MB)

Trainable params: 3,729,955 (14.23 MB)

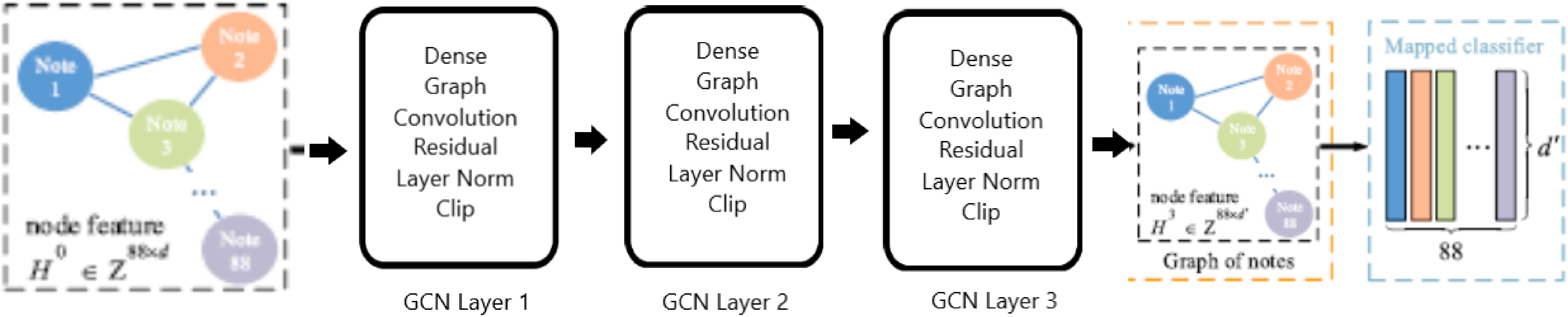
Non-trainable params: 178 (712.00 B)

Output:

Model: "cnn_lstm"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 229, 480, 1)	0
conv2d (Conv2D)	(None, 229, 480, 32)	128
batch_normalization (BatchNormalization)	(None, 229, 480, 32)	128
max_pooling2d (MaxPooling2D)	(None, 114, 480, 32)	0
conv2d_1 (Conv2D)	(None, 114, 480, 57)	5,529
batch_normalization_1 (BatchNormalization)	(None, 114, 480, 57)	228
max_pooling2d_1 (MaxPooling2D)	(None, 57, 480, 57)	0
reshape (Reshape)	(None, 480, 3249)	0
bidirectional (Bidirectional)	(None, 480, 256)	3,459,072
dropout (Dropout)	(None, 480, 256)	0
dense (Dense)	(None, 480, 768)	197,376
clip_layer (ClipLayer)	(None, 480, 768)	0
dense_1 (Dense)	(None, 480, 88)	67,672
transpose_layer (TransposeLayer)	(None, 88, 480)	0

MODULE 3: LABEL LEARNING USING GCN



1) Initialize Node features Ho
2) Pearson Correlation Matrix
3) Creation of Adjacency Matrix

(64,88,769)
Mapped Classifier

Feature Concatenation:
Dot product of feature map from CNN + LSTM and Mapped Classifier from GCN to compute Label loss and train our model

MODULE 3: LABEL LEARNING USING GCN

```python
# Custom GCN Layer
class GCNLayer(layers.Layer):
    def __init__(self, units, activation="relu", **kwargs):
        super(GCNLayer, self).__init__(**kwargs)
        self.units = units
        self.activation = tf.keras.activations.get(activation)
        self.dense = layers.Dense(units, activation=activation, use_bias=True)
        self.layer_norm = layers.LayerNormalization()

    def call(self, inputs):
        node_features, adjacency = inputs
        x = self.dense(node_features)
        x_gcn = tf.matmul(adjacency, x)
        x = x + x_gcn
        x = self.layer_norm(x)
        x = ClipLayer(min_value=-1e6, max_value=1e6)(x)
        return x
```

```
Total params: 1,844,736 (7.04 MB)

Trainable params: 1,844,736 (7.04 MB)

Non-trainable params: 0 (0.00 B)
```

Output:

Layer (type)	Output Shape	Param #	Connected to
node_features (InputLayer)	(None, 88, 88)	0	-
adjacency (InputLayer)	(None, 88, 88)	0	-
gcn_layer (GCNLayer)	(None, 88, 768)	69,888	node_features[0]… adjacency[0][0]
gcn_layer_1 (GCNLayer)	(None, 88, 768)	592,128	gcn_layer[0][0], adjacency[0][0]
gcn_layer_2 (GCNLayer)	(None, 88, 768)	592,128	gcn_layer_1[0][0… adjacency[0][0]
dense_5 (Dense)	(None, 88, 768)	590,592	gcn_layer_2[0][0]
multiply (Multiply)	(None, 88, 768)	0	dense_5[0][0]

MODULE 4: RESULTS

The training journey for the CR-GCN model involved multiple iterations, starting with data preprocessing, facing errors like GCN gradient issues, and refining the model to achieve a final frame-level F1 score of 0.19.

Initial challenges included GCN gradients not flowing, addressed by adjusting the adjacency matrix and adding auxiliary loss, and numerical instabilities like NaN losses, fixed by clipping and normalization.

Training for fewer epochs initially helped identify errors, leading to adjustments in learning rate schedules, focal loss parameters, and model architecture, culminating in a 100-epoch run with early stopping.

The final model used a batch size of 64, focal loss with class weighting, and custom metrics, achieving balanced precision and recall, though note-level metrics remained zero, suggesting post-processing needs.

Suggestions for future work include exploring deeper architectures, data augmentation, and advanced post-processing for note detection, potentially improving F1 scores and addressing zero note-level metrics.

MODULE 4: WHAT WENT WRONG ?

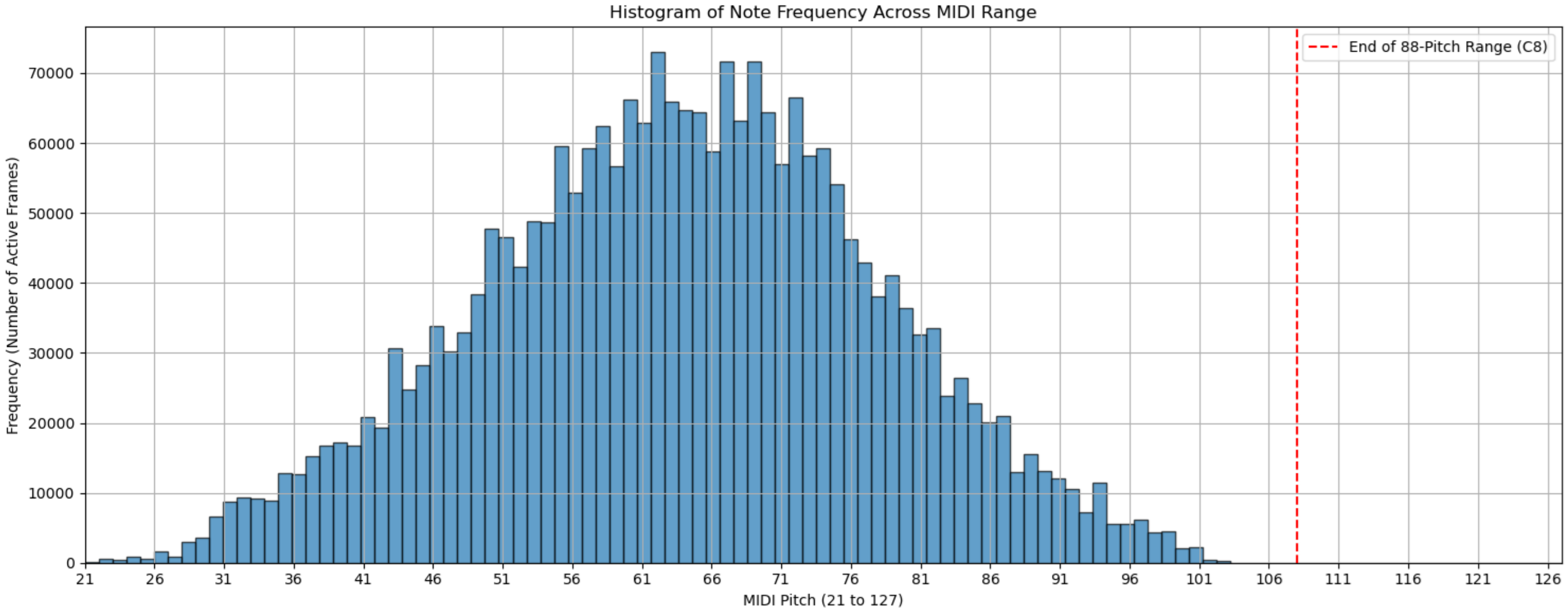
➔*Class Imbalance*

The dataset showed a significant imbalance, with many more negative instances (note inactive) than positive instances (note active). This was addressed using focal loss with gamma=1.5, alpha=0.8, and pos_weight=2.0, aiming to focus on hard-to-classify positive examples. However, precision remained low (0.0759 at threshold 0.2), while recall was high (0.5903), leading to an F1 score of 0.1345, indicating many false positives.

No. of Positive Labels : 2.8M

No. of Negative Labels: 39M

Ratio : 1.14



MODULE 4: WHAT WENT WRONG ?

➔*Adjacency Matrix Threshold Issue*

Early training attempts with the GCN branch encountered issues, as gradients were absent (indicated by GCN Gradients Present: False), impeding effective learning. To address this, we preprocessed the adjacency matrix by adding self-loops, applying row normalization, and handling NaN values to ensure a valid graph structure. However, this introduced a trade-off: aiming for high sparsity (approximately 0.95) reduced the number of edges, limiting the graph's connectivity, while increasing the number of edges to enhance connectivity conversely reduced sparsity, making it difficult to strike a balance. Unlike the base paper's suggested 0.7 threshold for edge selection, we found it challenging to adopt this cutoff due to the conflicting demands of sparsity and edge density. Despite these adjustments, GCN gradients remained absent in later runs, suggesting deeper integration issues within the model.

Current Threshold : 0.18
Total No. of Nodes: 88
Total no. of Edges: 223
Sparsity : 0.9424

Base Paper Threshold : 0.7
Total No. fo Nodes: 88
Total No. of Edges: 0
Sparsity: 1.0

MODULE 4: WHAT WENT WRONG ?

➔ *High Recall and Low Precision*

The model consistently showed high recall (0.5903 at threshold 0.2) but low precision (0.0759), suggesting over-sensitivity and many false positives. This was addressed by optimizing the threshold (found optimal at 0.2) and exploring post-processing techniques, though note-level metrics remained zero, indicating challenges in converting frame-level predictions to note events.

MODULE 4: WHAT WENT WRONG ?

➔*GCN Gradient Flow Absence*

GCN gradient flow is not enabled, resulting in negligible or absent gradients during training.

Vanishing Gradients: Deep GCN layers with normalization (e.g., LayerNorm) may scale gradients too small.
Small GCN Outputs: Initial GCN output scale (W) is too low, diminishing its contribution to the loss.
Residual Connection Issues: Improper residual connections may suppress GCN updates.
Potential Solutions:
Adjust LayerNorm with a higher epsilon or skip normalization for early layers.
Increase GCN output scale (e.g., higher initial scale value or trainable scaling factor).
Enhance residual connections with skip weights or remove for better gradient propagation.
Impact:
GCN fails to learn meaningful graph representations (e.g., pitch relationships).
Learning shifts to CNN-LSTM, underutilizing GCN’s graph-based insights.
Leads to suboptimal model performance.

MODULE 5: DEPLOYMENT

Web Hosting

Framework

- A django project was created to serve as the backend for this project.
- HTML5/CSS and JavaScript were utilised to implement the required front-end aspects of the web-application.

File Transfer & Authentication

- To facilitate the ease in processing the files, a sessions token has been implemented for each incoming client session in django duly serving the purpose of authentication.
- Basic HTTP is utilised to transfer audio files from client to server and the output file from server to client.

Hosting

- For development and testing, the project was hosted locally to facilitate quick responses and instant error corrections.
- For production, the Django project was tunneled through an ngrok server to make it publicly accessible across all networks via the provided URL.

MODULE 5: DEPLOYMENT

Lilypond

Text-Based Input:

- Uses a plain-text language to describe musical sheet scores.

Automatic Engraving:

- Converts input into beautifully formatted, publication-quality sheet music with sophisticated typesetting algorithms.

Customization and Precision:

- Offers fine control over spacing, layout, and stylistic choices, allowing specific musical traditions.

REFERENCES

1 Xiao, Z., Chen, X., & Zhou, L. (2023). Polyphonic piano transcription based on graph convolutional network. *Signal Processing*, *212*, 109134.

2 Edwards, D., Dixon, S., Benetos, E., Maezawa, A., & Kusaka, Y. (2024). A Data-Driven Analysis of Robust Automatic Piano Transcription. *IEEE Signal Processing Letters*.

3 Saputra, F., Namyu, U. G., Vincent, D. S., & Gema, A. P. (2021). Automatic piano sheet music transcription with machine learning. *Journal of Computer Science*, *17*(3), 178-187.

4 Guo, R., & Zhu, Y. (2025). Research on the Recognition of Piano-Playing Notes by a Music Transcription Algorithm. *Journal of Advanced Computational Intelligence and Intelligent Informatics*, *29*(1), 152-157.

5 de la Fuente, C., Valero-Mas, J. J., Castellanos, F. J., & Calvo-Zaragoza, J. (2022). Multimodal image and audio music transcription. *International Journal of Multimedia Information Retrieval*, *11*(1), 77-84.

6 Wan, Y., Wang, X., Zhou, R., & Yan, Y. (2015). Automatic piano music transcription using audio-visual features. *Chinese Journal of Electronics*, *24*(3), 596-603.


Thank
you