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

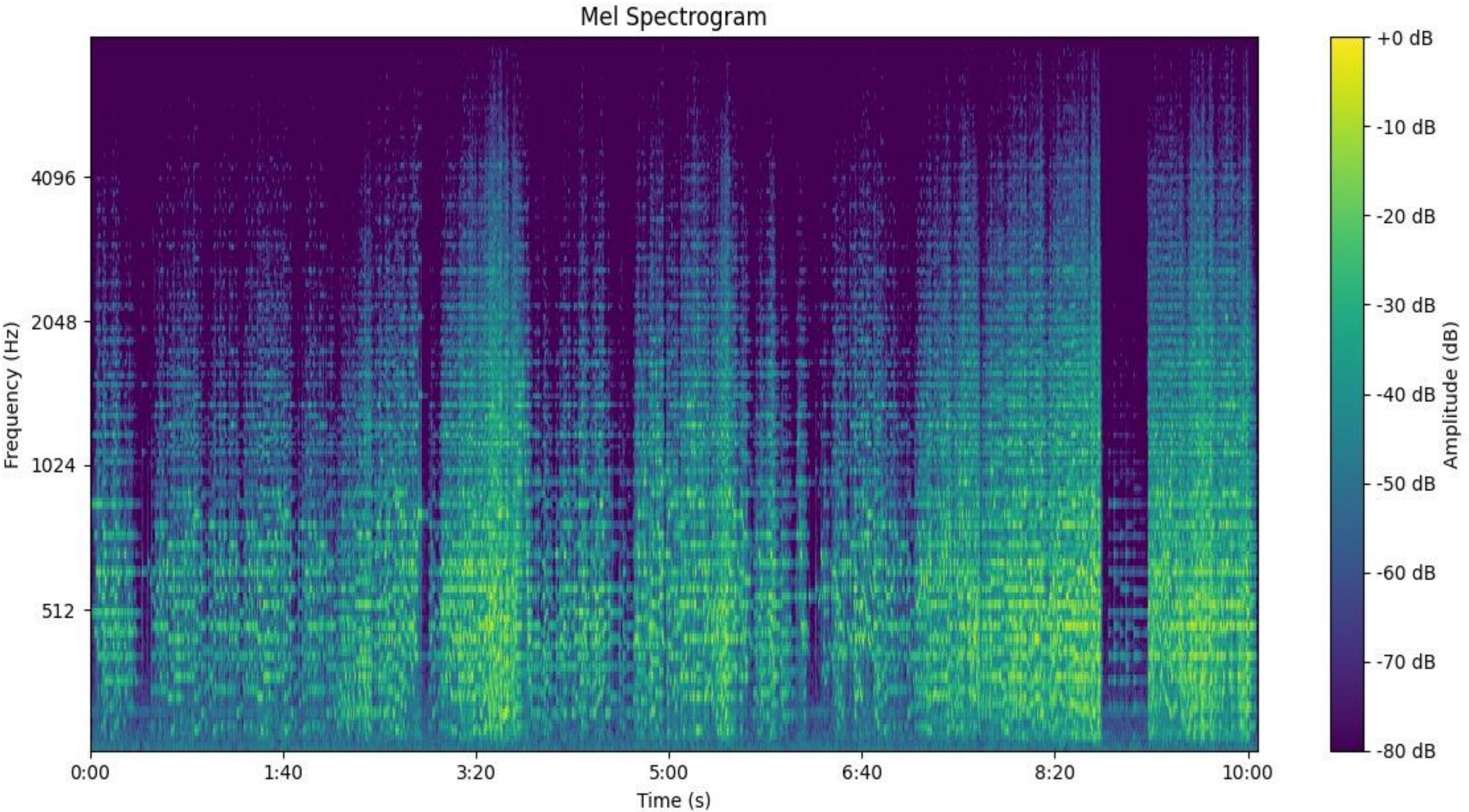
TITLE	MERITS	DEMERITS
A Data-Driven Analysis of Robust Automatic Piano Transcription	The study improved note-onset accuracy to 88.4 F1-score on the MAPS dataset through data augmentation.	Performance on out-of-distribution annotated piano data indicates challenges in generalizing to unseen data.
Automatic 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

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

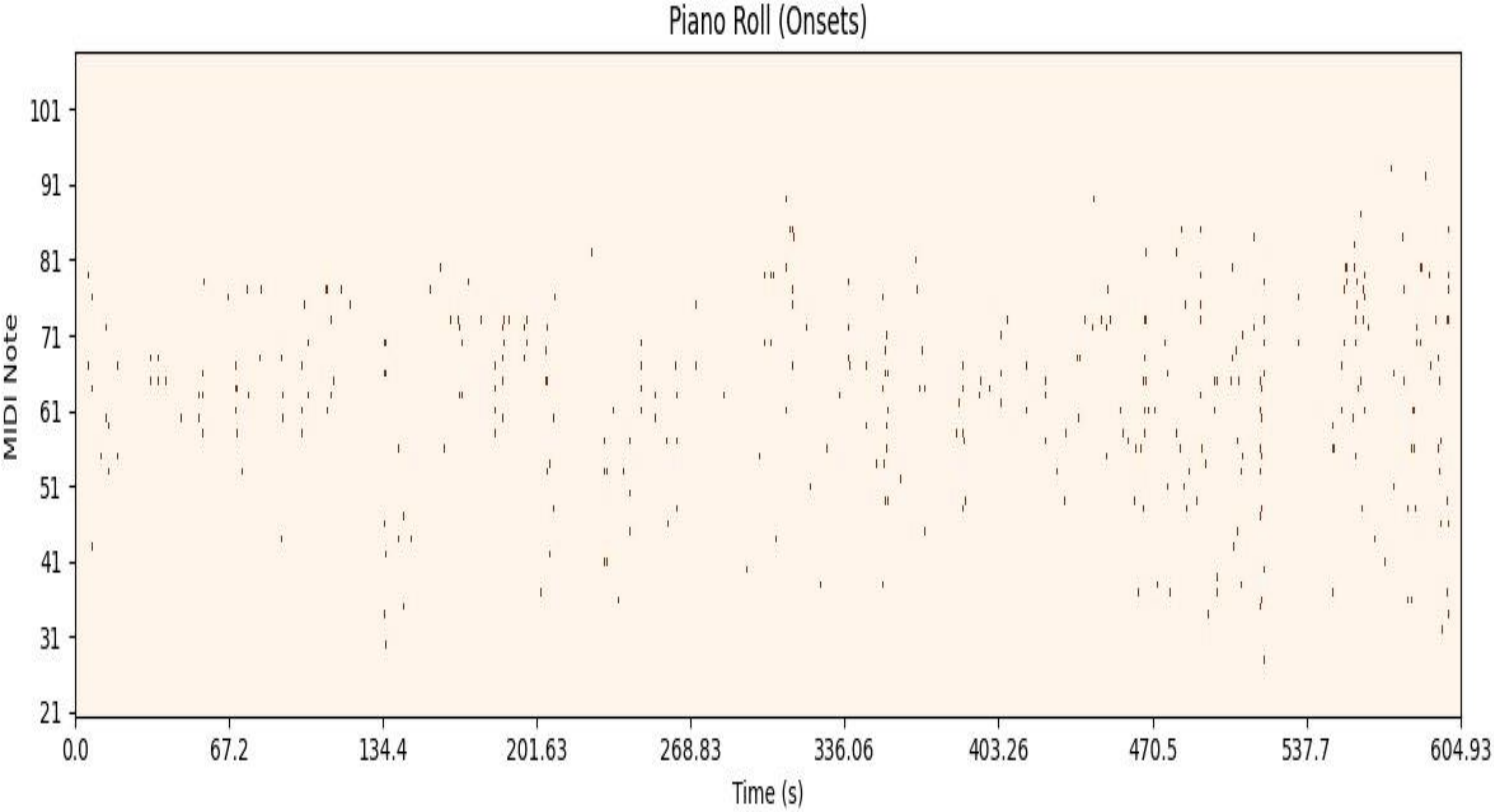
MODULE 1: DATA PREPROCESSING

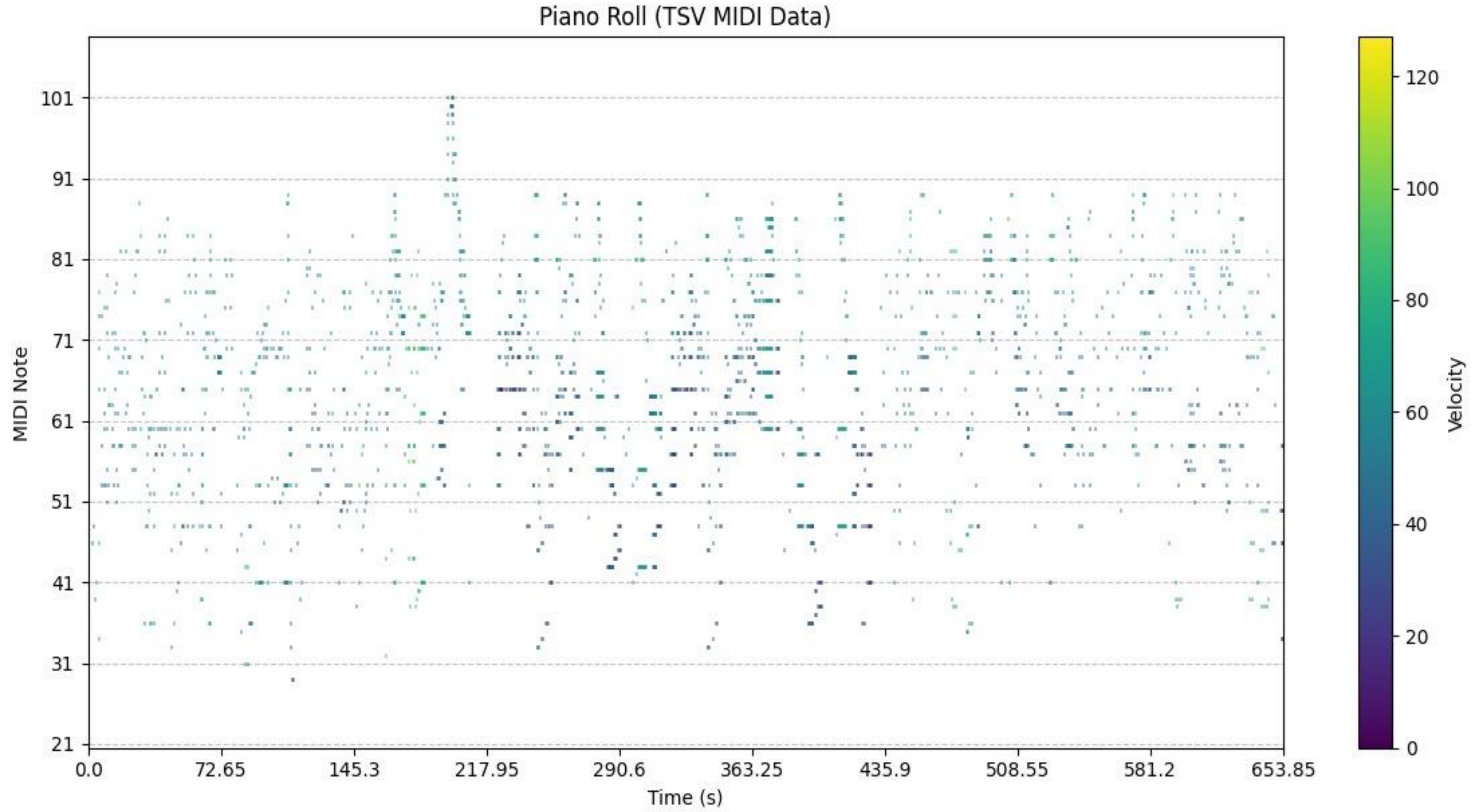
- As per the dataset we have incorporated into our project (Maestro v3), the categorical X attribute consists of .wav audio files.
- To enhance performance of our model, it is required to convert the .wav files into its corresponding mel-spectrograms which is achieved using the librosa module of python.
- 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.


Mel Spectrogram
Frequency (Hz)
4096
2048
1024
512
Time (s)
0:00
1:40
3:20
5:00
6:40
8:20
10:00
Amplitude (dB)
+0 dB
-10 dB
-20 dB
-30 dB
-40 dB
-50 dB
-60 dB
-70 dB
-80 dB

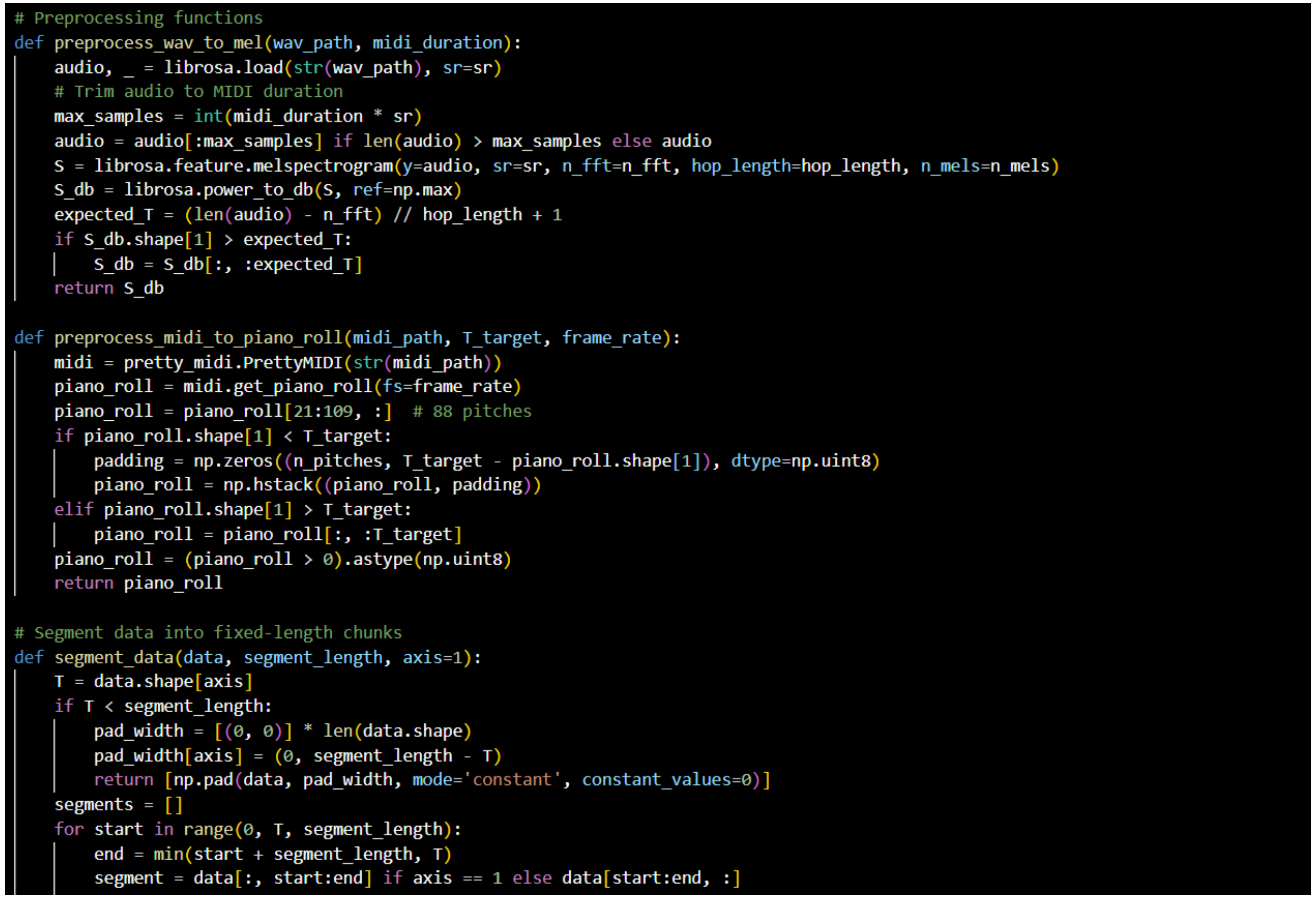
MODULE 1: DATA PREPROCESSING

- 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 (tsv files) 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.


Piano Roll (Onsets)
MIDI Note
101
91
81
71
61
51
41
31
21
Time (s)
0.0
67.2
134.4
201.63
268.83
336.06
403.26
470.5
537.7
604.93

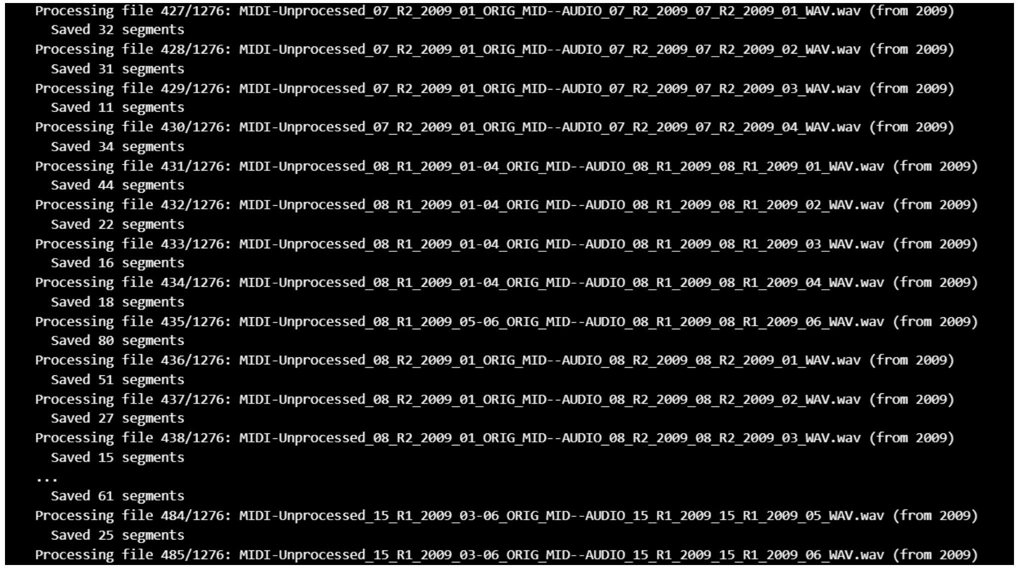

Piano Roll (TSV MIDI Data)
MIDI Note
101
91
81
71
61
51
41
31
21
Time (s)
0.0
72.65
145.3
217.95
290.6
363.25
435.9
508.55
581.2
653.85
Velocity
120
100
80
60
40
20
0

MODULE 1: DATA PREPROCESSING



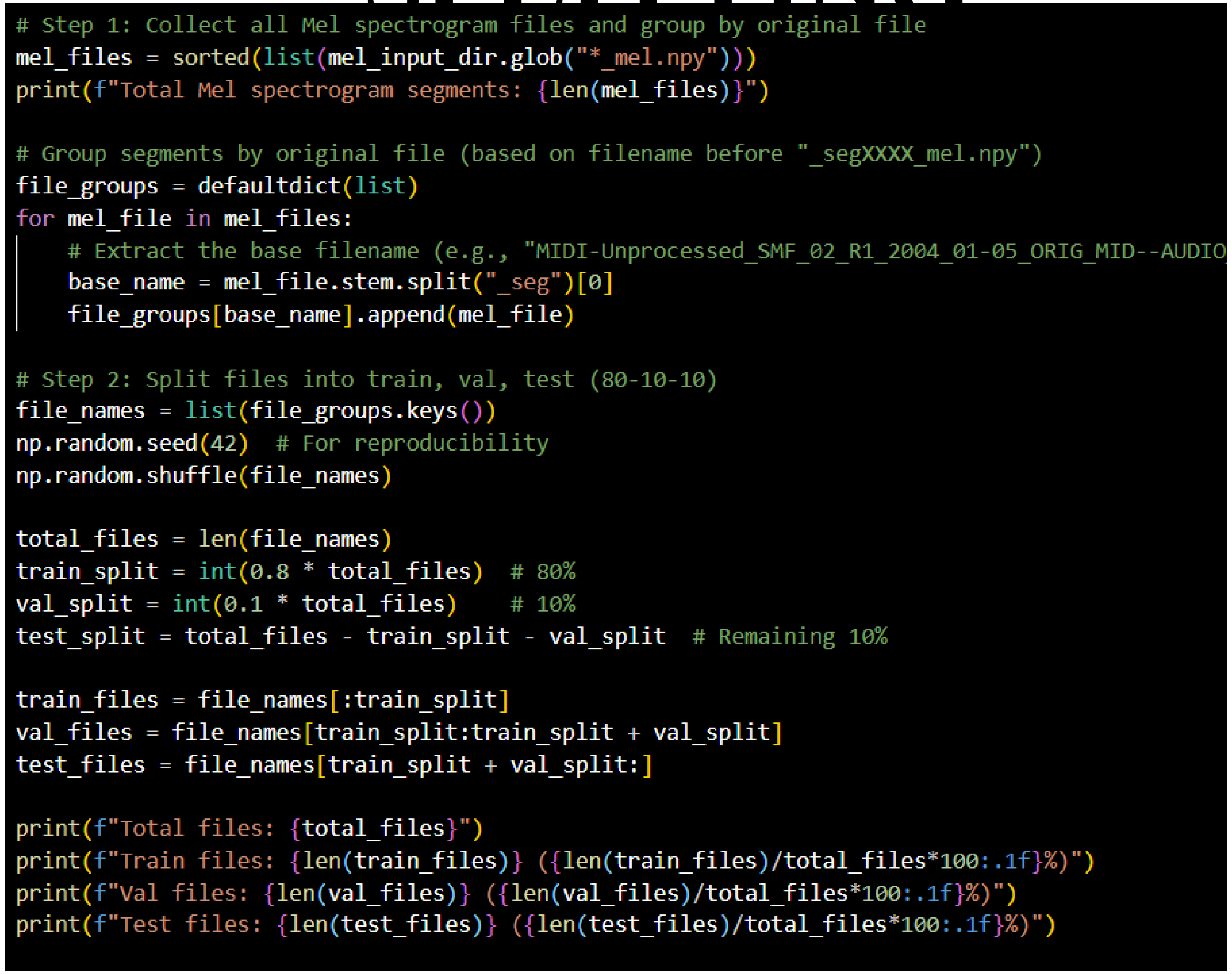
```python
# Preprocessing functions
def preprocess_wav_to_mel(wav_path, midi_duration):
    audio, _ = librosa.load(str(wav_path), sr=sr)
    # Trim audio to MIDI duration
    max_samples = int(midi_duration * sr)
    audio = audio[:max_samples] if len(audio) > max_samples else audio
    S = librosa.feature.melspectrogram(y=audio, 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:



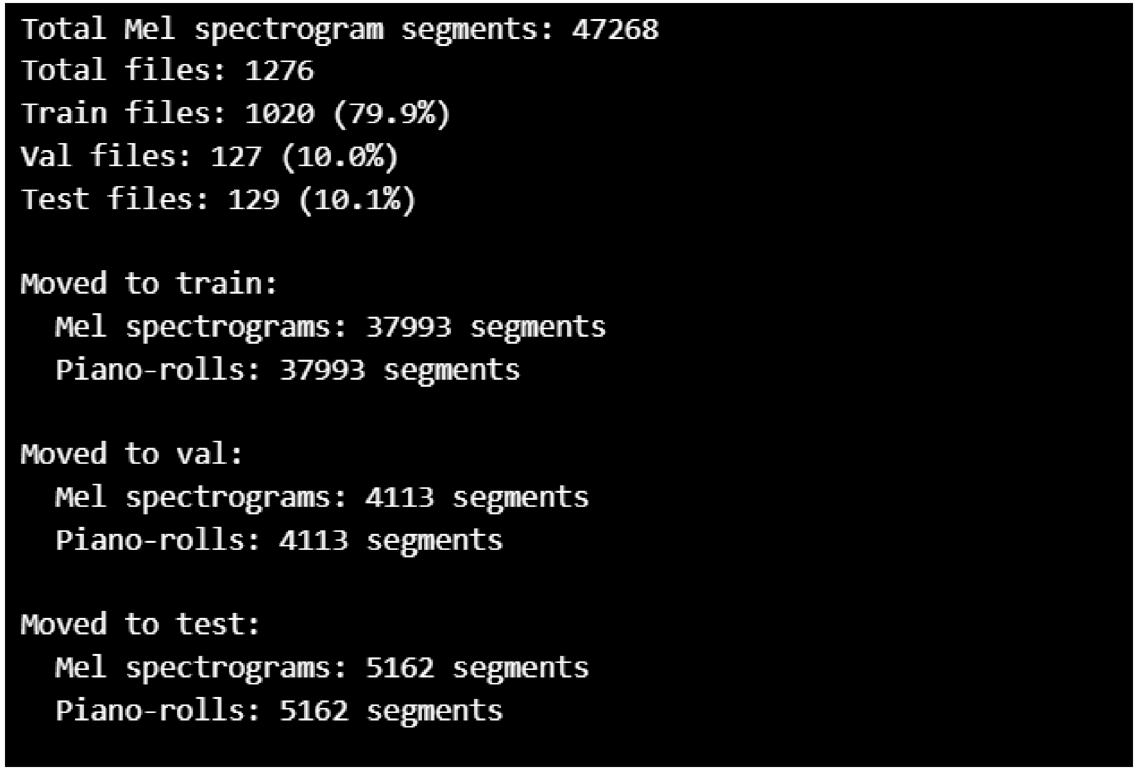
```
Processing file 427/1276: MIDI-Unprocessed_07_R2_2009_01_ORIG_MID--AUDIO_07_R2_2009_07_R2_2009_01_WAV.wav (from 2009)
  Saved 32 segments
Processing file 428/1276: MIDI-Unprocessed_07_R2_2009_01_ORIG_MID--AUDIO_07_R2_2009_07_R2_2009_02_WAV.wav (from 2009)
  Saved 31 segments
Processing file 429/1276: 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



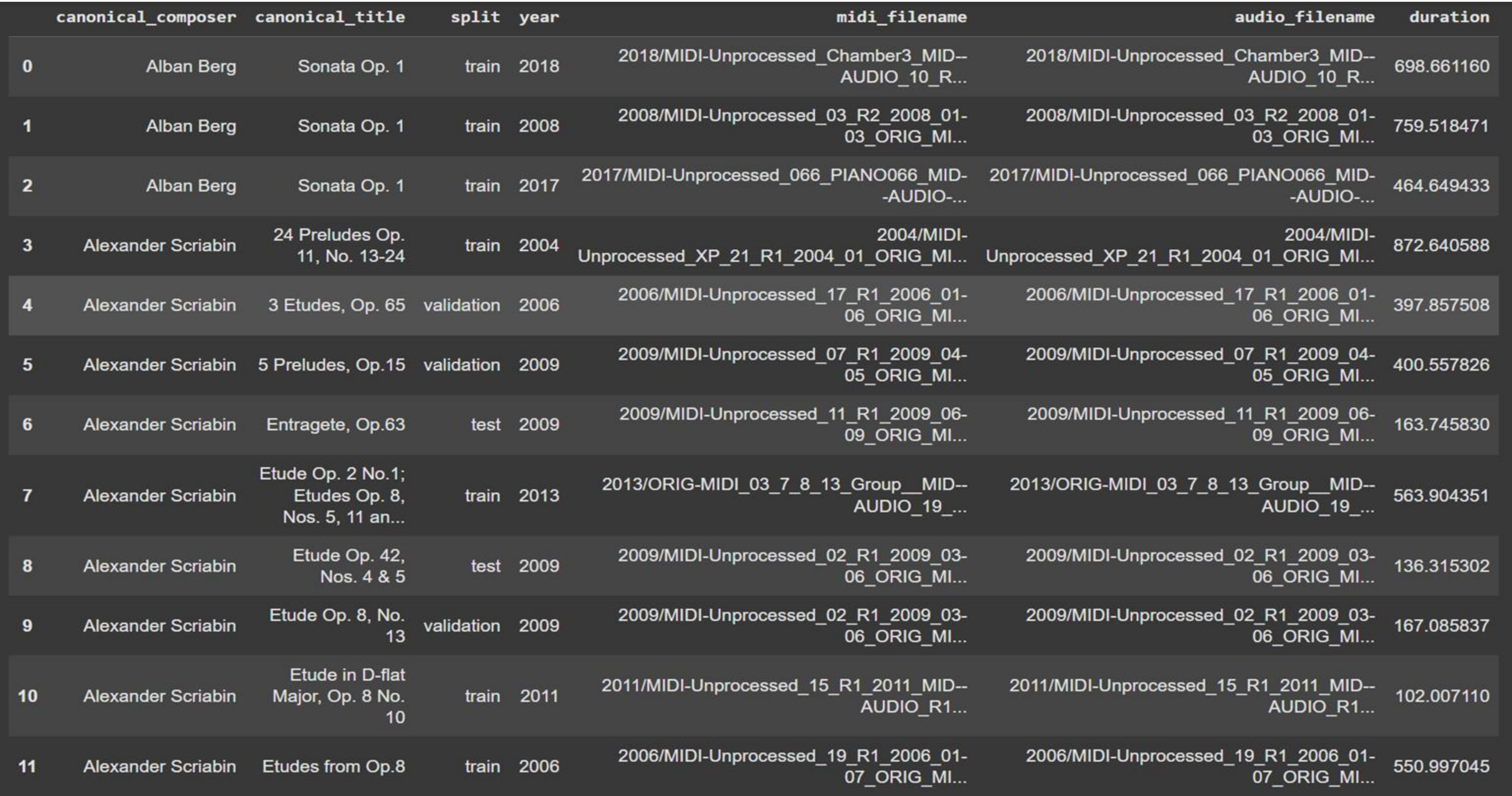
```python
# Step 1: Collect all Mel spectrogram files and group by original file
mel_files = sorted(list(mel_input_dir.glob("*_mel.npy")))
print(f"Total Mel spectrogram segments: {len(mel_files)}")

# 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:



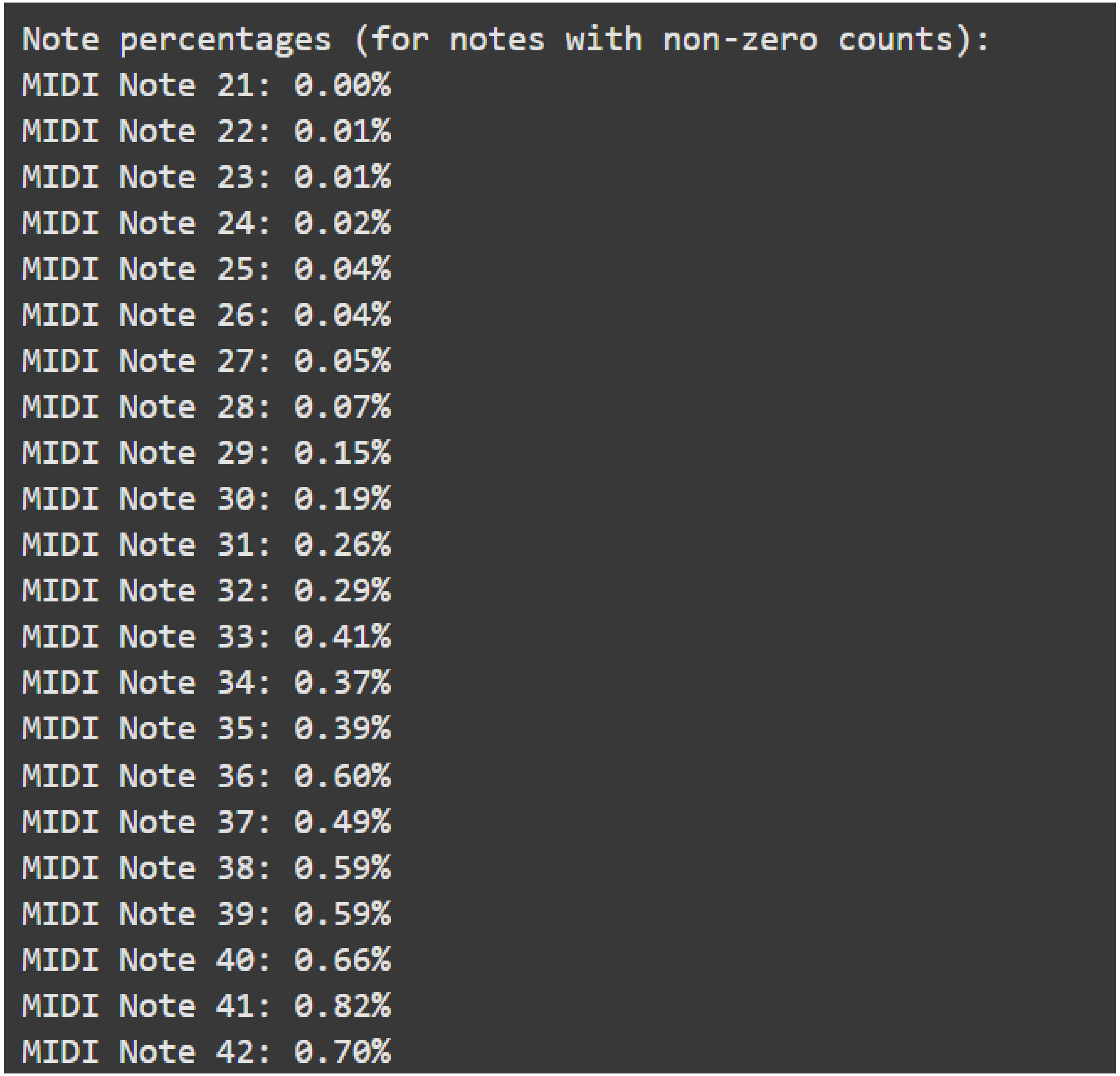
```
Total Mel spectrogram segments: 47268
Total files: 1276
Train files: 1020 (79.9%)
Val files: 127 (10.0%)
Test files: 129 (10.1%)

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
```

METADATA

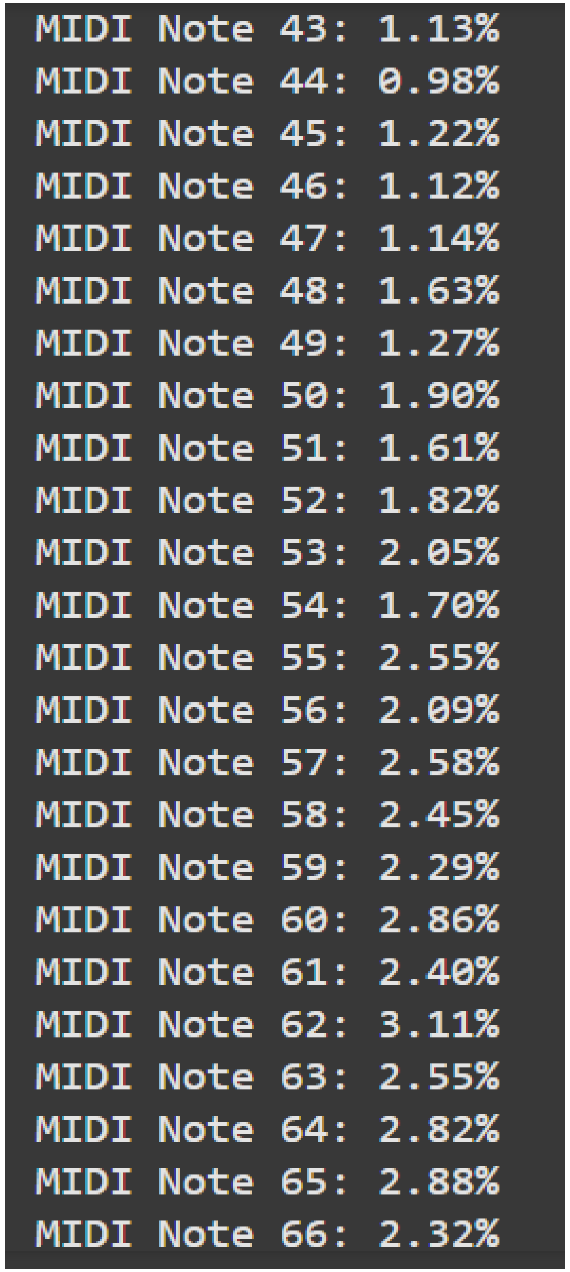


	canonical_composer	canonical_title	split	year	midi_filename	audio_filename	duration
0	Alban Berg	Sonata Op. 1	train	2018	2018/MIDI-Unprocessed_Chamber3_MID--AUDIO_10_R...	2018/MIDI-Unprocessed_Chamber3_MID--AUDIO_10_R...	698.661160
1	Alban Berg	Sonata Op. 1	train	2008	2008/MIDI-Unprocessed_03_R2_2008_01-03_ORIG_MI...	2008/MIDI-Unprocessed_03_R2_2008_01-03_ORIG_MI...	759.518471
2	Alban Berg	Sonata Op. 1	train	2017	2017/MIDI-Unprocessed_066_PIANO066_MID--AUDIO-...	2017/MIDI-Unprocessed_066_PIANO066_MID--AUDIO-...	464.649433
3	Alexander Scriabin	24 Preludes Op. 11, No. 13-24	train	2004	2004/MIDI-Unprocessed_XP_21_R1_2004_01_ORIG_MI...	2004/MIDI-Unprocessed_XP_21_R1_2004_01_ORIG_MI...	872.640588
4	Alexander Scriabin	3 Etudes, Op. 65	validation	2006	2006/MIDI-Unprocessed_17_R1_2006_01-06_ORIG_MI...	2006/MIDI-Unprocessed_17_R1_2006_01-06_ORIG_MI...	397.857508
5	Alexander Scriabin	5 Preludes, Op.15	validation	2009	2009/MIDI-Unprocessed_07_R1_2009_04-05_ORIG_MI...	2009/MIDI-Unprocessed_07_R1_2009_04-05_ORIG_MI...	400.557826
6	Alexander Scriabin	Entragete, Op.63	test	2009	2009/MIDI-Unprocessed_11_R1_2009_06-09_ORIG_MI...	2009/MIDI-Unprocessed_11_R1_2009_06-09_ORIG_MI...	163.745830
7	Alexander Scriabin	Etude Op. 2 No.1; Etudes Op. 8, Nos. 5, 11 an...	train	2013	2013/ORIG-MIDI_03_7_8_13_Group__MID--AUDIO_19_...	2013/ORIG-MIDI_03_7_8_13_Group__MID--AUDIO_19_...	563.904351
8	Alexander Scriabin	Etude Op. 42, Nos. 4 & 5	test	2009	2009/MIDI-Unprocessed_02_R1_2009_03-06_ORIG_MI...	2009/MIDI-Unprocessed_02_R1_2009_03-06_ORIG_MI...	136.315302
9	Alexander Scriabin	Etude Op. 8, No. 13	validation	2009	2009/MIDI-Unprocessed_02_R1_2009_03-06_ORIG_MI...	2009/MIDI-Unprocessed_02_R1_2009_03-06_ORIG_MI...	167.085837
10	Alexander Scriabin	Etude in D-flat Major, Op. 8 No. 10	train	2011	2011/MIDI-Unprocessed_15_R1_2011_MID--AUDIO_R1...	2011/MIDI-Unprocessed_15_R1_2011_MID--AUDIO_R1...	102.007110
11	Alexander Scriabin	Etudes from Op.8	train	2006	2006/MIDI-Unprocessed_19_R1_2006_01-07_ORIG_MI...	2006/MIDI-Unprocessed_19_R1_2006_01-07_ORIG_MI...	550.997045

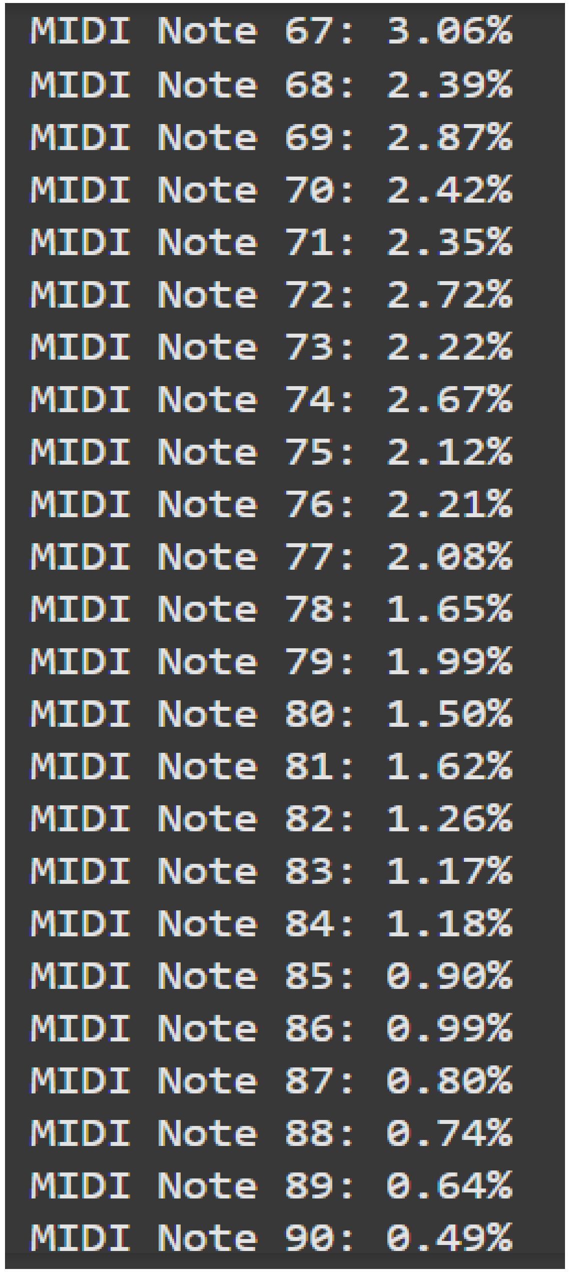
MODULE 1: DATA VISUALIZATION



```
Note percentages (for notes with non-zero counts):
MIDI Note 21: 0.00%
MIDI Note 22: 0.01%
MIDI Note 23: 0.01%
MIDI Note 24: 0.02%
MIDI Note 25: 0.04%
MIDI Note 26: 0.04%
MIDI Note 27: 0.05%
MIDI Note 28: 0.07%
MIDI Note 29: 0.15%
MIDI Note 30: 0.19%
MIDI Note 31: 0.26%
MIDI Note 32: 0.29%
MIDI Note 33: 0.41%
MIDI Note 34: 0.37%
MIDI Note 35: 0.39%
MIDI Note 36: 0.60%
MIDI Note 37: 0.49%
MIDI Note 38: 0.59%
MIDI Note 39: 0.59%
MIDI Note 40: 0.66%
MIDI Note 41: 0.82%
MIDI Note 42: 0.70%
```

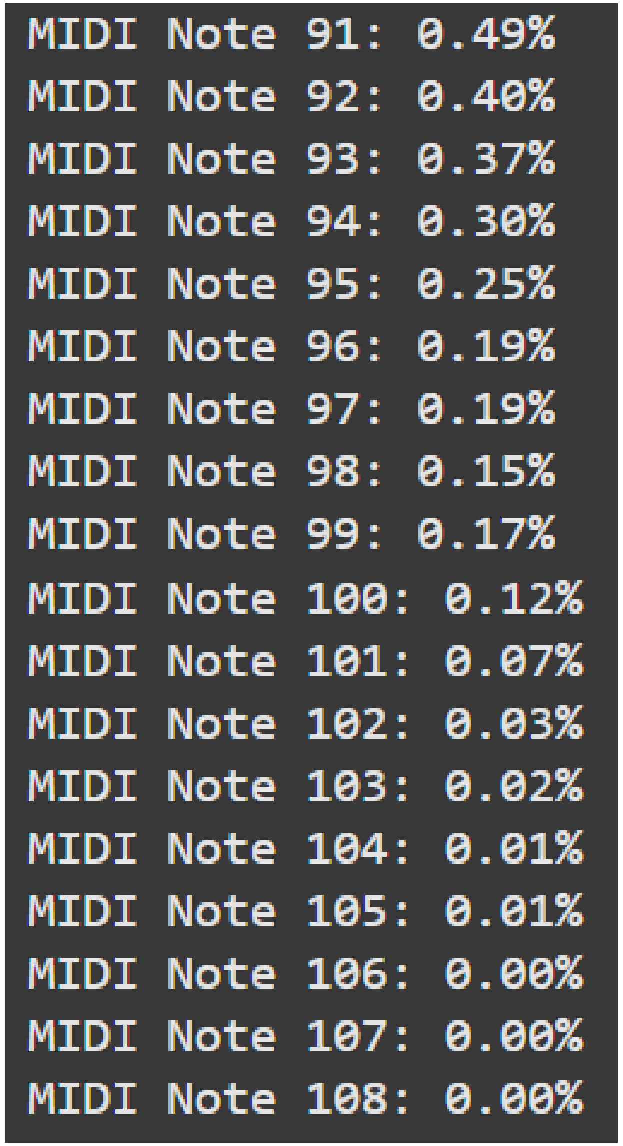


```
MIDI Note 43: 1.13%
MIDI Note 44: 0.98%
MIDI Note 45: 1.22%
MIDI Note 46: 1.12%
MIDI Note 47: 1.14%
MIDI Note 48: 1.63%
MIDI Note 49: 1.27%
MIDI Note 50: 1.90%
MIDI Note 51: 1.61%
MIDI Note 52: 1.82%
MIDI Note 53: 2.05%
MIDI Note 54: 1.70%
MIDI Note 55: 2.55%
MIDI Note 56: 2.09%
MIDI Note 57: 2.58%
MIDI Note 58: 2.45%
MIDI Note 59: 2.29%
MIDI Note 60: 2.86%
MIDI Note 61: 2.40%
MIDI Note 62: 3.11%
MIDI Note 63: 2.55%
MIDI Note 64: 2.82%
MIDI Note 65: 2.88%
MIDI Note 66: 2.32%
```

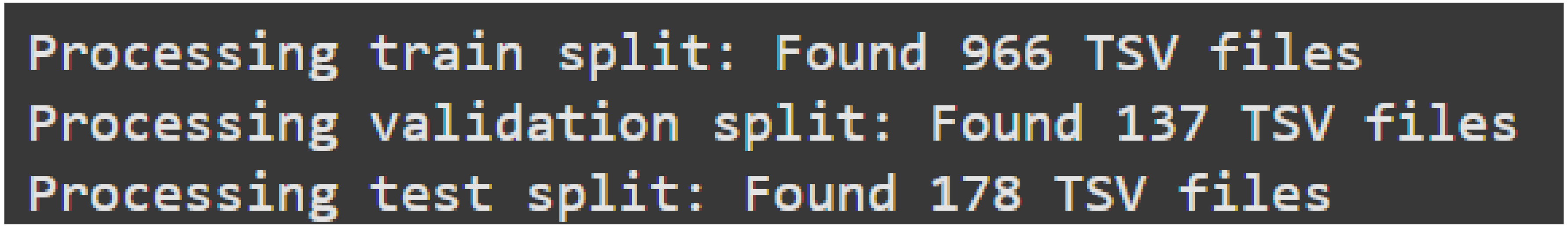


```
MIDI Note 67: 3.06%
MIDI Note 68: 2.39%
MIDI Note 69: 2.87%
MIDI Note 70: 2.42%
MIDI Note 71: 2.35%
MIDI Note 72: 2.72%
MIDI Note 73: 2.22%
MIDI Note 74: 2.67%
MIDI Note 75: 2.12%
MIDI Note 76: 2.21%
MIDI Note 77: 2.08%
MIDI Note 78: 1.65%
MIDI Note 79: 1.99%
MIDI Note 80: 1.50%
MIDI Note 81: 1.62%
MIDI Note 82: 1.26%
MIDI Note 83: 1.17%
MIDI Note 84: 1.18%
MIDI Note 85: 0.90%
MIDI Note 86: 0.99%
MIDI Note 87: 0.80%
MIDI Note 88: 0.74%
MIDI Note 89: 0.64%
MIDI Note 90: 0.49%
```

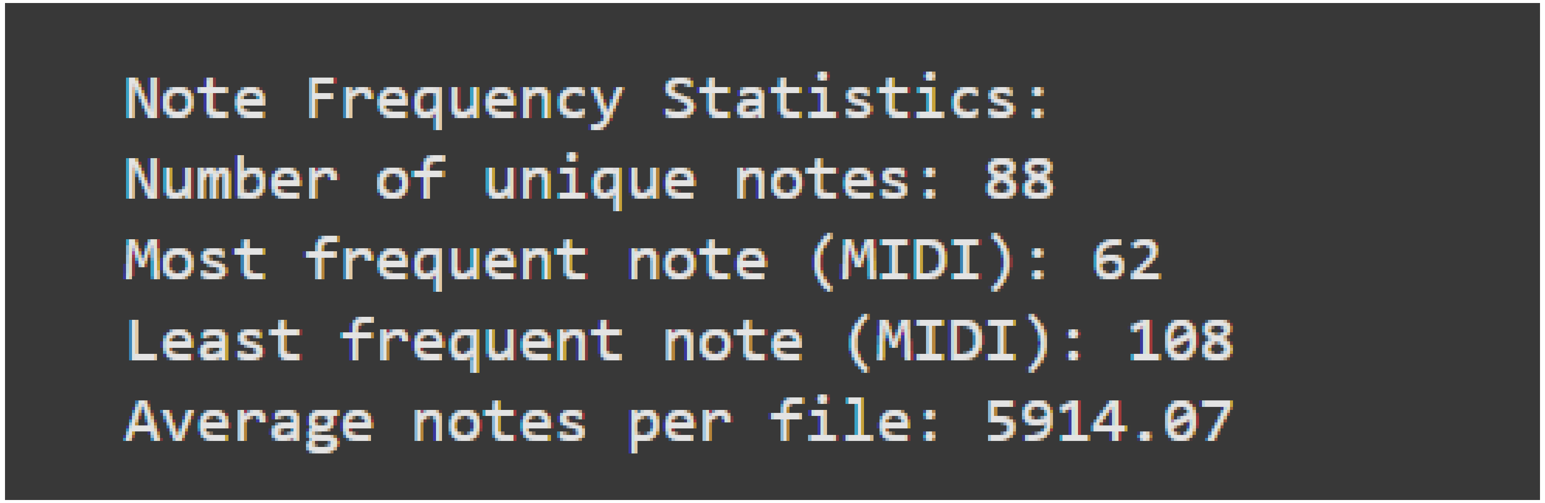
MODULE 1: DATA VISUALIZATION



```
MIDI Note 91: 0.49%
MIDI Note 92: 0.40%
MIDI Note 93: 0.37%
MIDI Note 94: 0.30%
MIDI Note 95: 0.25%
MIDI Note 96: 0.19%
MIDI Note 97: 0.19%
MIDI Note 98: 0.15%
MIDI Note 99: 0.17%
MIDI Note 100: 0.12%
MIDI Note 101: 0.07%
MIDI Note 102: 0.03%
MIDI Note 103: 0.02%
MIDI Note 104: 0.01%
MIDI Note 105: 0.01%
MIDI Note 106: 0.00%
MIDI Note 107: 0.00%
MIDI Note 108: 0.00%
```

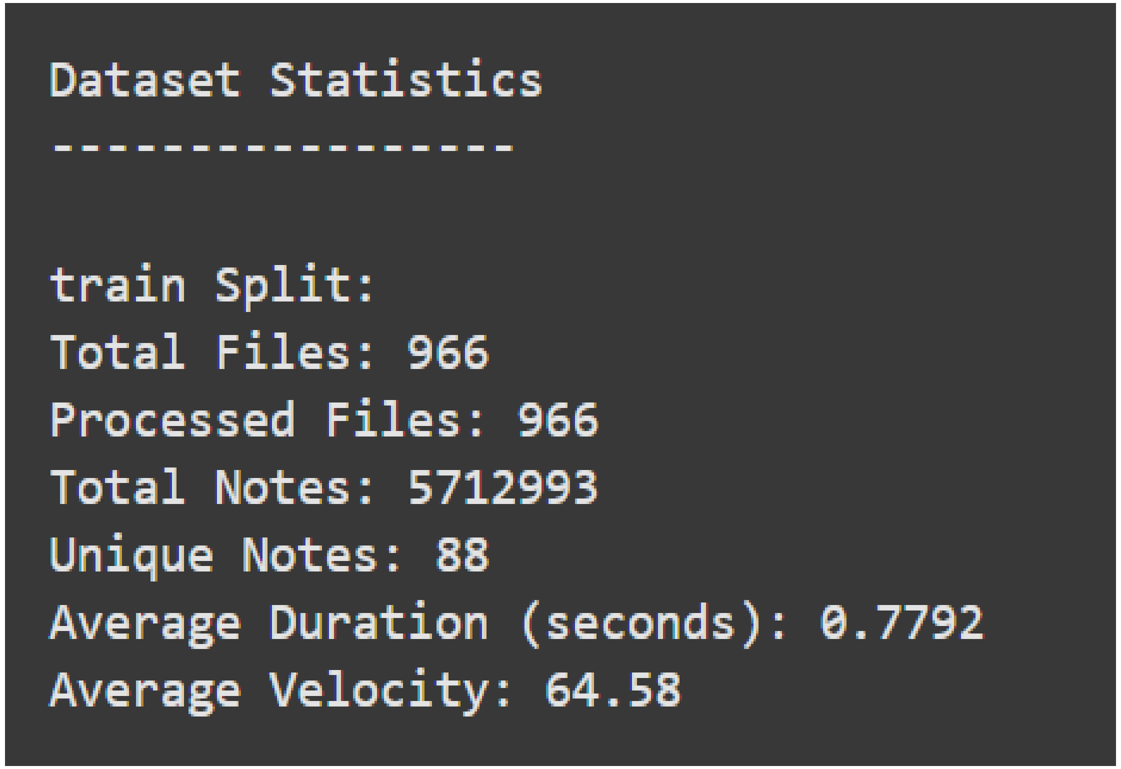


```
Processing train split: Found 966 TSV files
Processing validation split: Found 137 TSV files
Processing test split: Found 178 TSV files
```

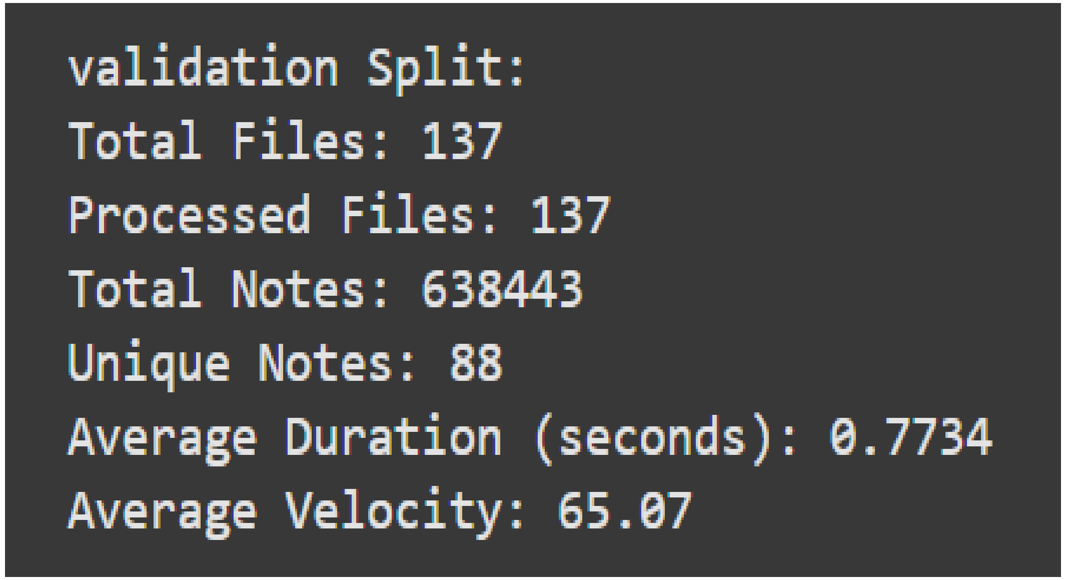


```
Note Frequency Statistics:
Number of unique notes: 88
Most frequent note (MIDI): 62
Least frequent note (MIDI): 108
Average notes per file: 5914.07
```

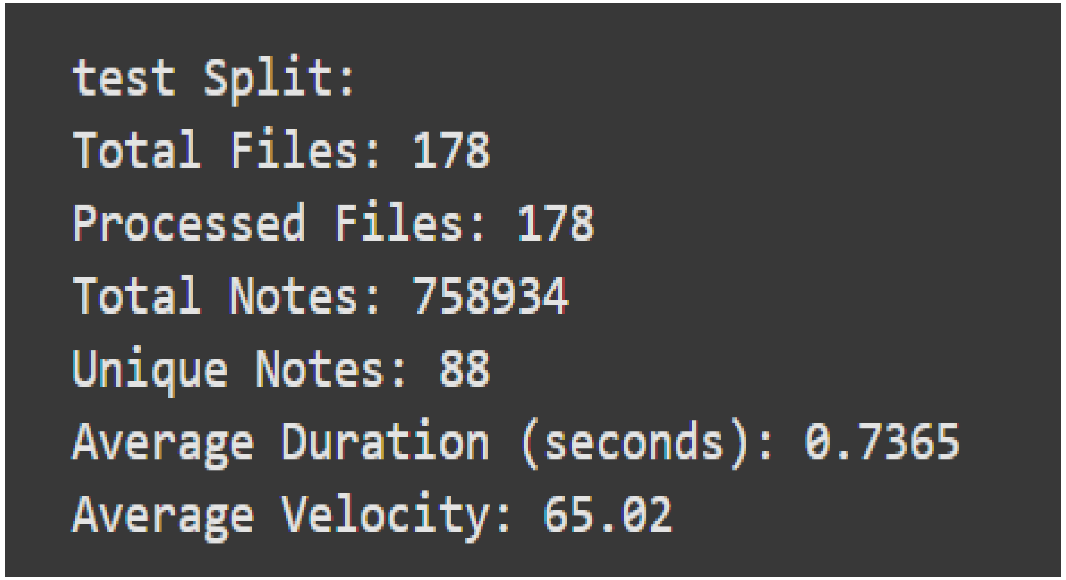
MODULE 1: DATA VISUALIZATION



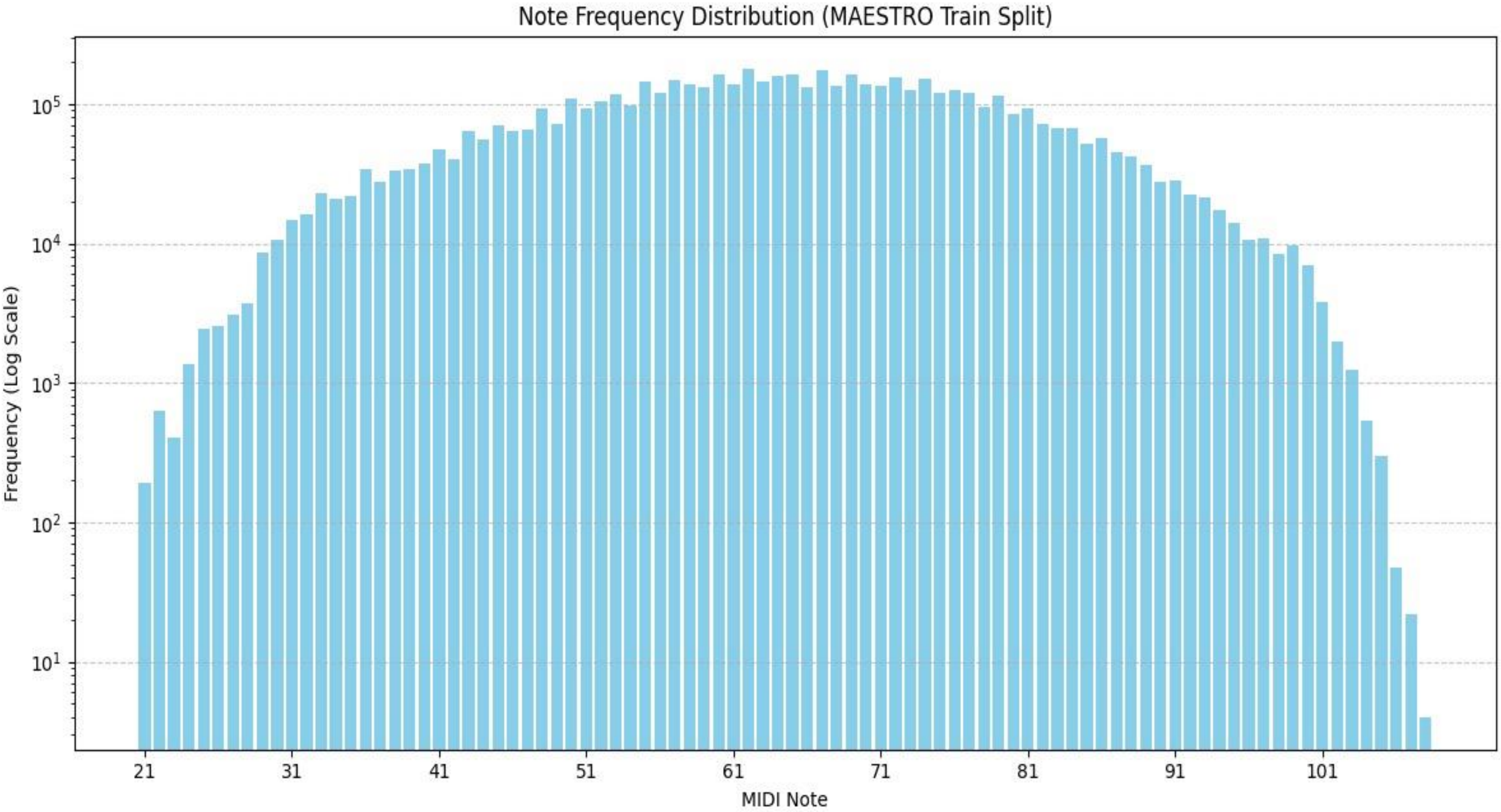
```
Dataset Statistics
------------------

train Split:
Total Files: 966
Processed Files: 966
Total Notes: 5712993
Unique Notes: 88
Average Duration (seconds): 0.7792
Average Velocity: 64.58
```

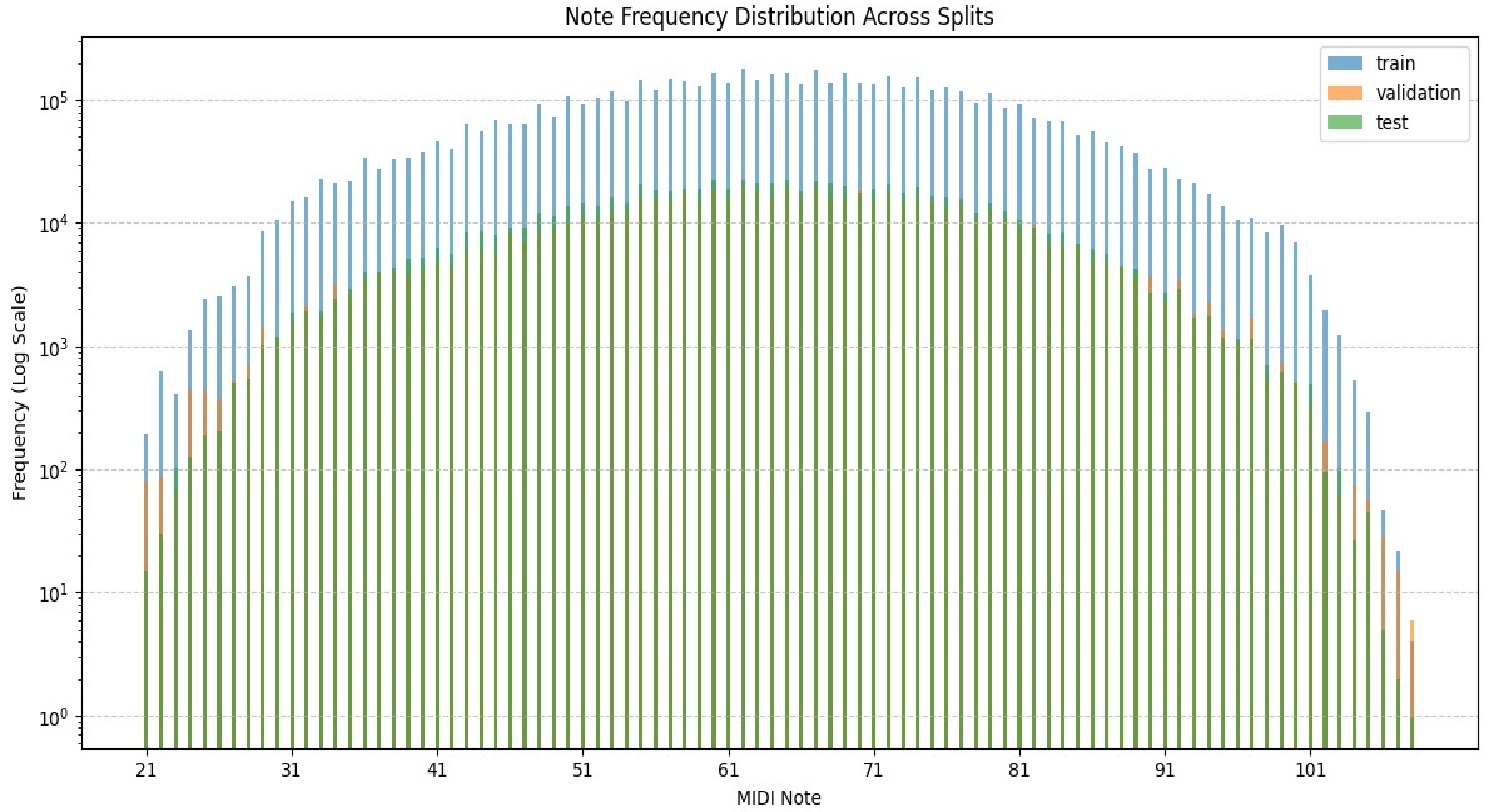


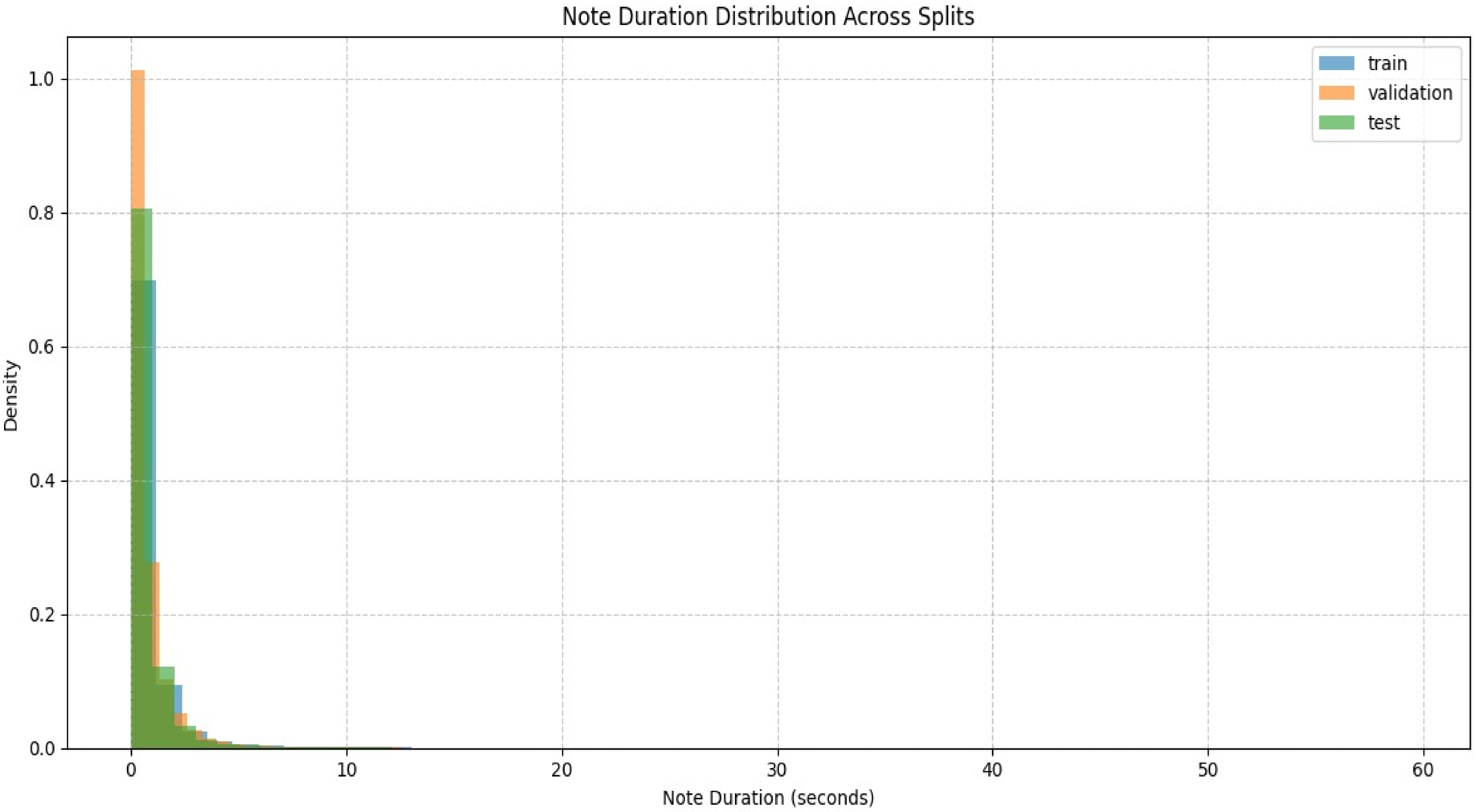
```
validation Split:
Total Files: 137
Processed Files: 137
Total Notes: 638443
Unique Notes: 88
Average Duration (seconds): 0.7734
Average Velocity: 65.07
```

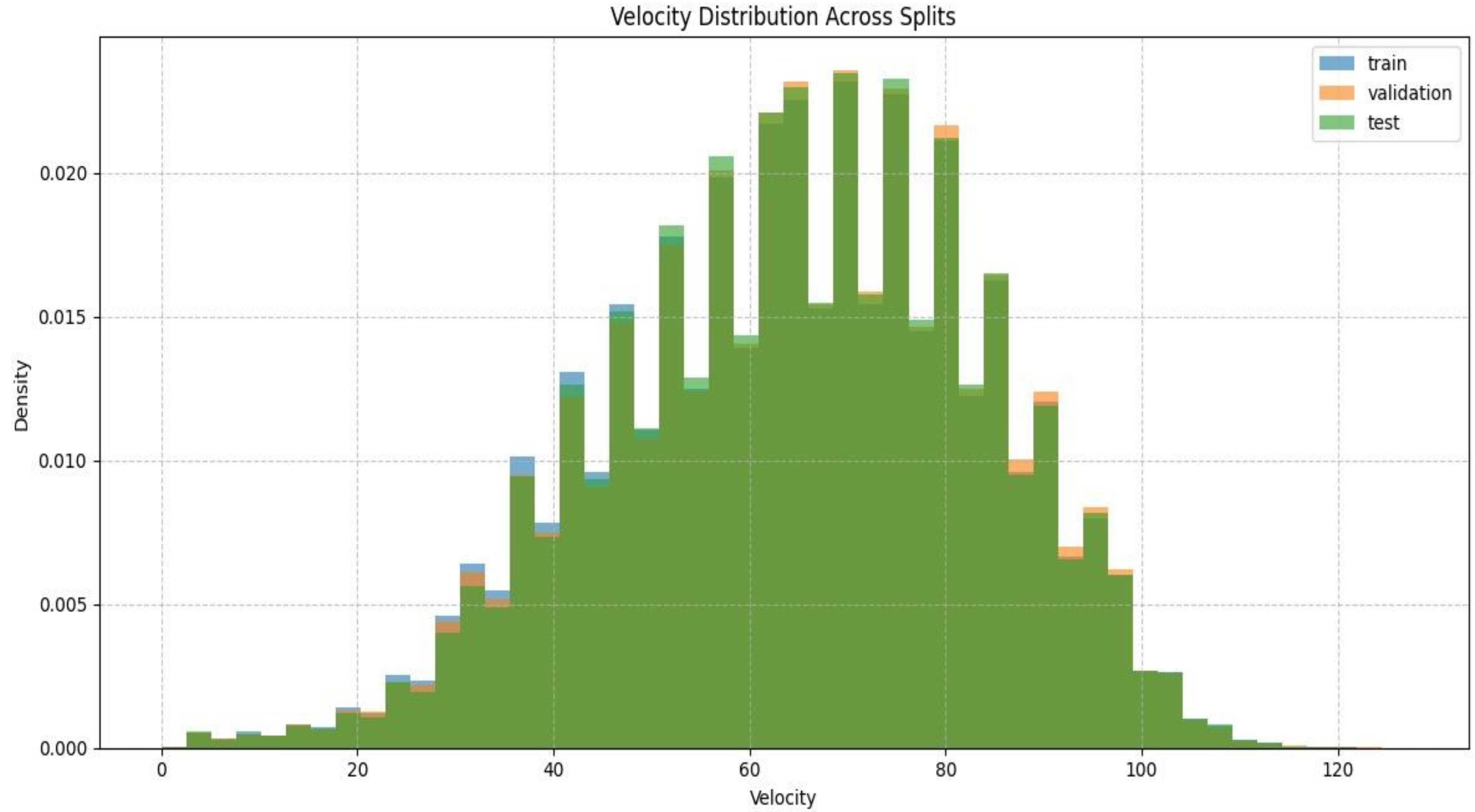


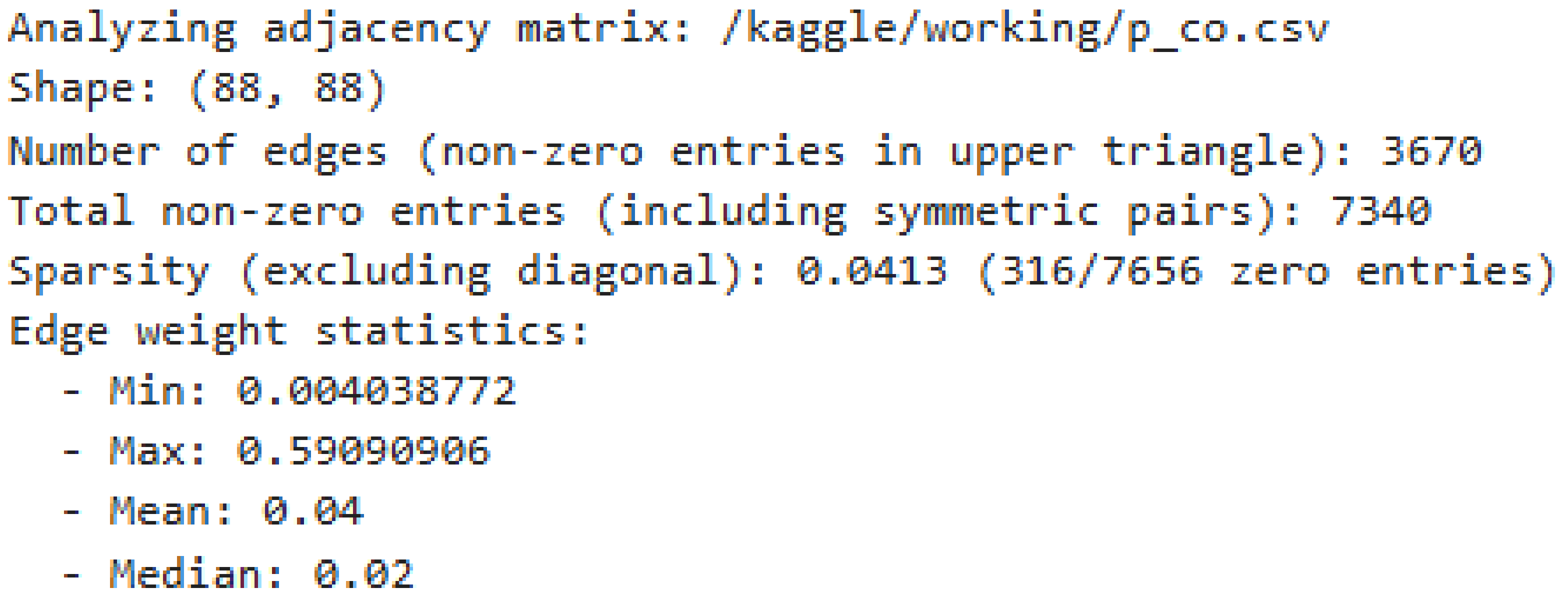
```
test Split:
Total Files: 178
Processed Files: 178
Total Notes: 758934
Unique Notes: 88
Average Duration (seconds): 0.7365
Average Velocity: 65.02
```


Note Frequency Distribution (MAESTRO Train Split)
Frequency (Log Scale)
10^5
10^4
10^3
10^2
10^1
21
31
41
51
61
71
81
91
101
MIDI Note

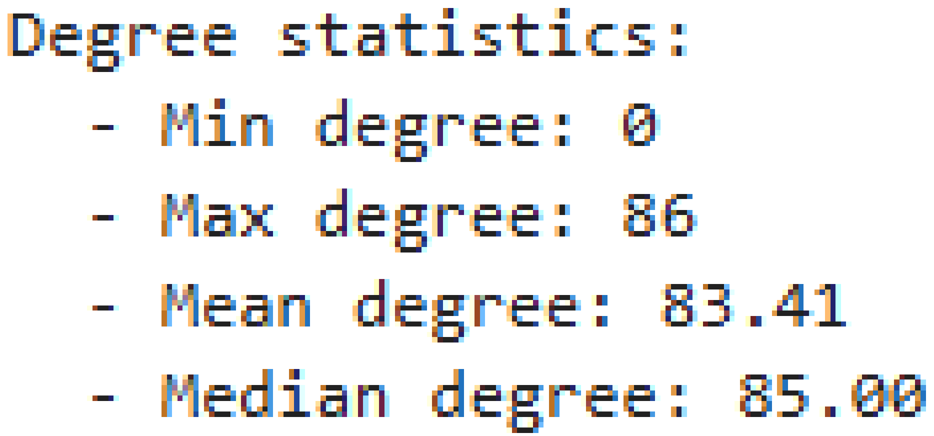

Note Frequency Distribution Across Splits
train
validation
test
Frequency (Log Scale)
10^0
10^1
10^2
10^3
10^4
10^5
21
31
41
51
61
71
81
91
101
MIDI Note


Note Duration Distribution Across Splits
train
validation
test
Density
1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50
60
Note Duration (seconds)

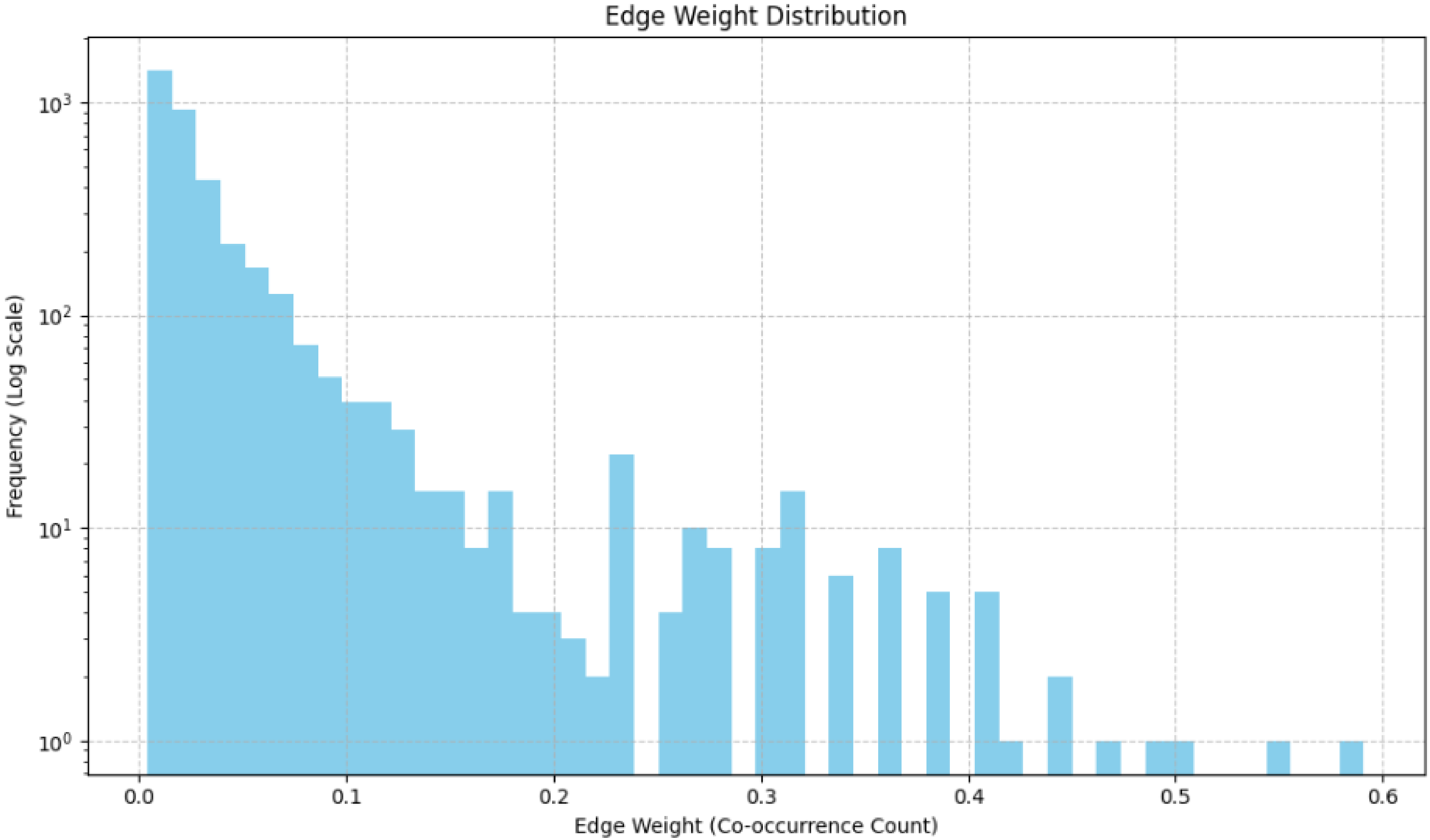

Velocity Distribution Across Splits
train
validation
test
Density
0.000
0.005
0.010
0.015
0.020
Velocity
0
20
40
60
80
100
120

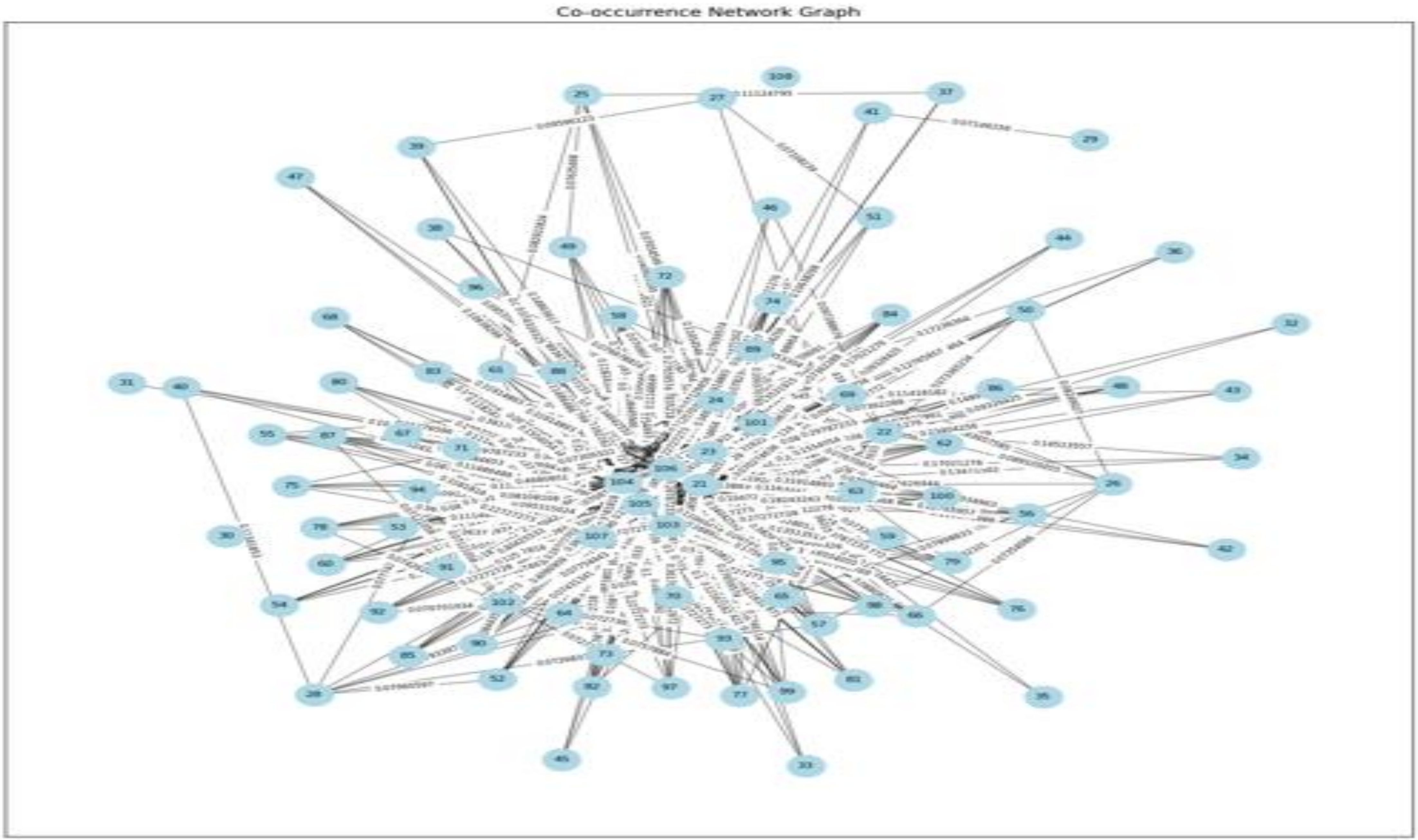


```
Analyzing adjacency matrix: /kaggle/working/p_co.csv
Shape: (88, 88)
Number of edges (non-zero entries in upper triangle): 3670
Total non-zero entries (including symmetric pairs): 7340
Sparsity (excluding diagonal): 0.0413 (316/7656 zero entries)
Edge weight statistics:
  - Min: 0.004038772
  - Max: 0.59090906
  - Mean: 0.04
  - Median: 0.02
```

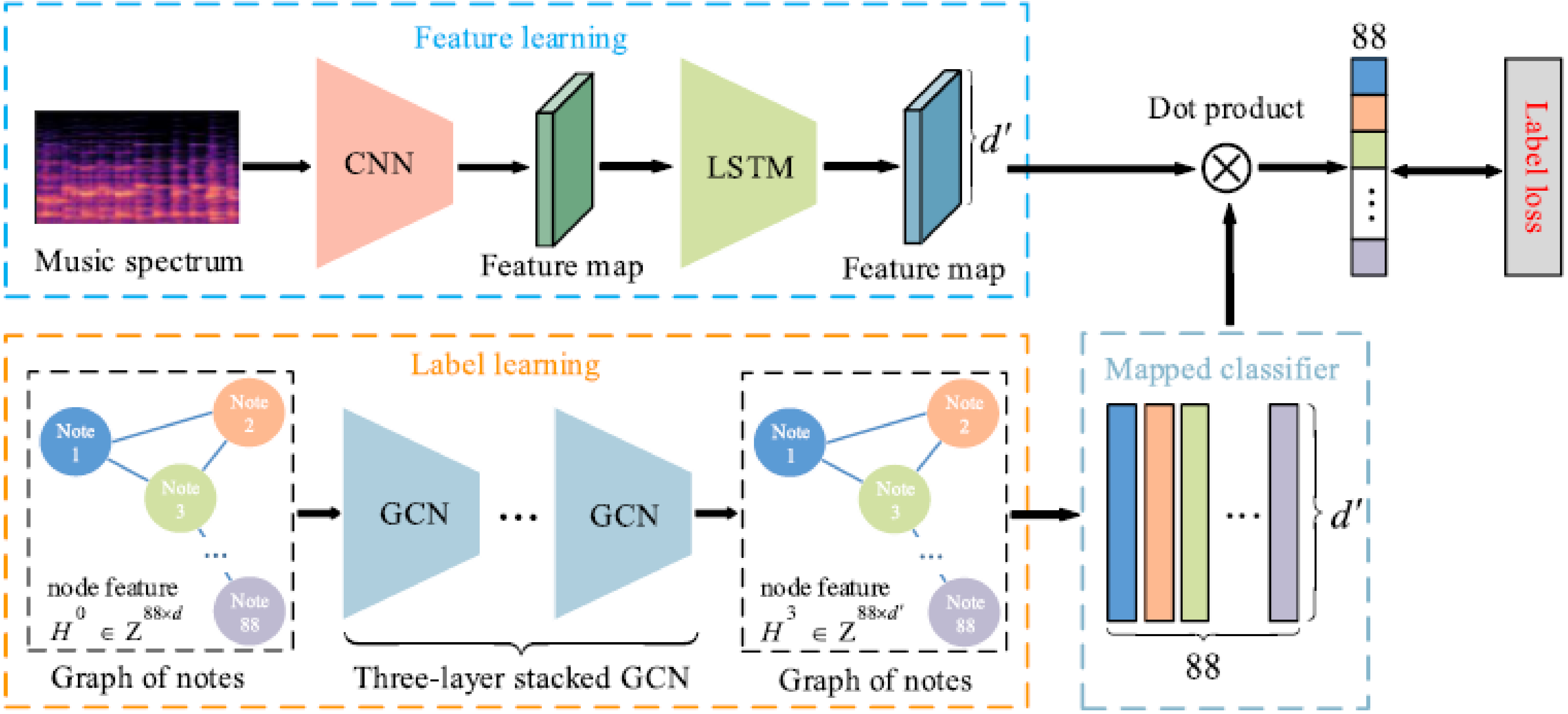


```
Degree statistics:
  - Min degree: 0
  - Max degree: 86
  - Mean degree: 83.41
  - Median degree: 85.00
```

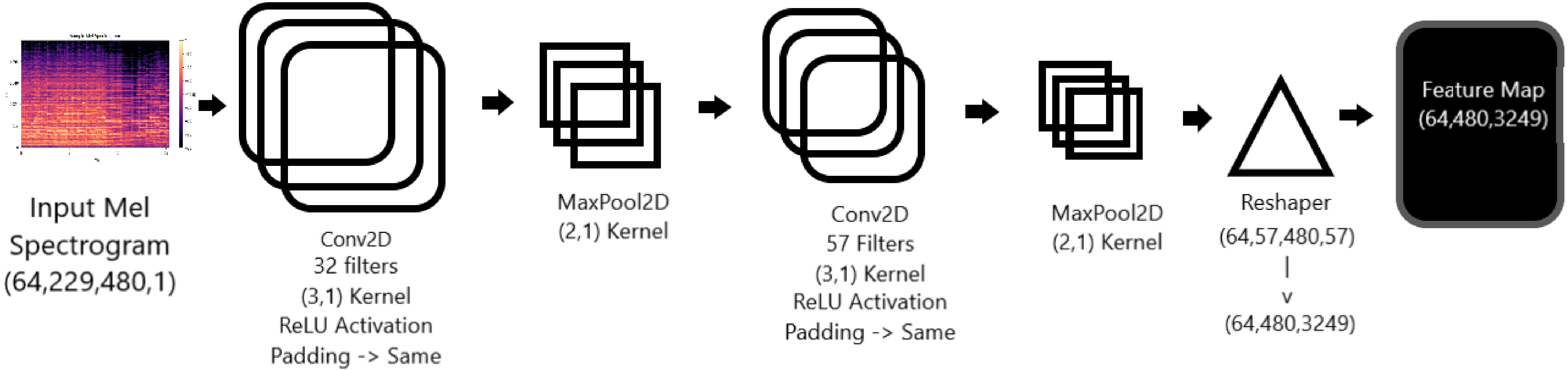

Edge Weight Distribution
Frequency (Log Scale)
10^3
10^2
10^1
10^0
0.0
0.1
0.2
0.3
0.4
0.5
0.6
Edge Weight (Co-occurrence Count)


Co-occurrence Network Graph

EXISTING SYSTEM



MODULE 2: FEATURE LEARNING CNN + LSTM



Input Mel Spectrogram (64,229,480,1)

Conv2D
32 filters
(3,1) Kernel
ReLU Activation
Padding -> Same

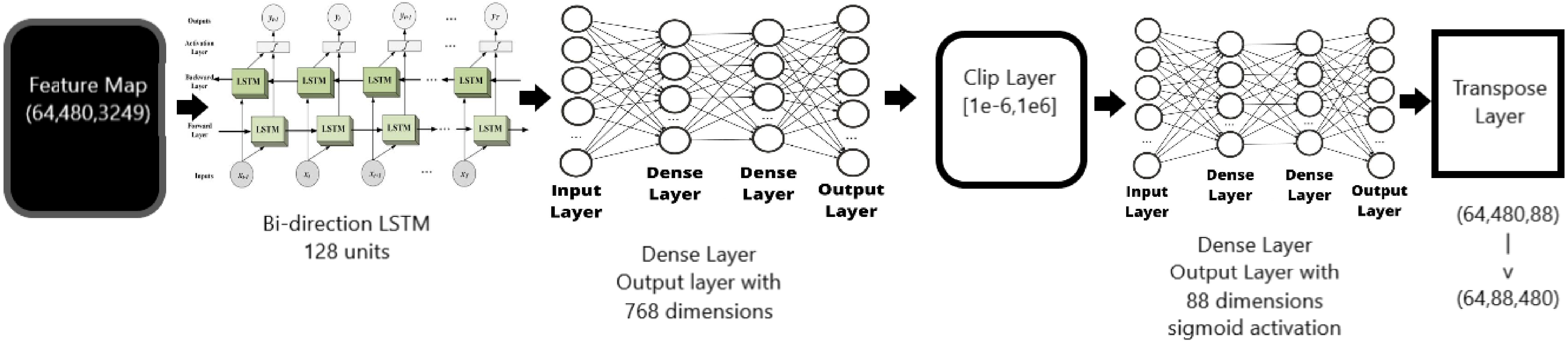
MaxPool2D
(2,1) Kernel

Conv2D
57 Filters
(3,1) Kernel
ReLU Activation
Padding -> Same

MaxPool2D
(2,1) Kernel

Reshaper
(64,57,480,57)
|
v
(64,480,3249)

Feature Map
(64,480,3249)



Feature Map
(64,480,3249)

Bi-direction LSTM
128 units

Dense Layer
Output layer with
768 dimensions

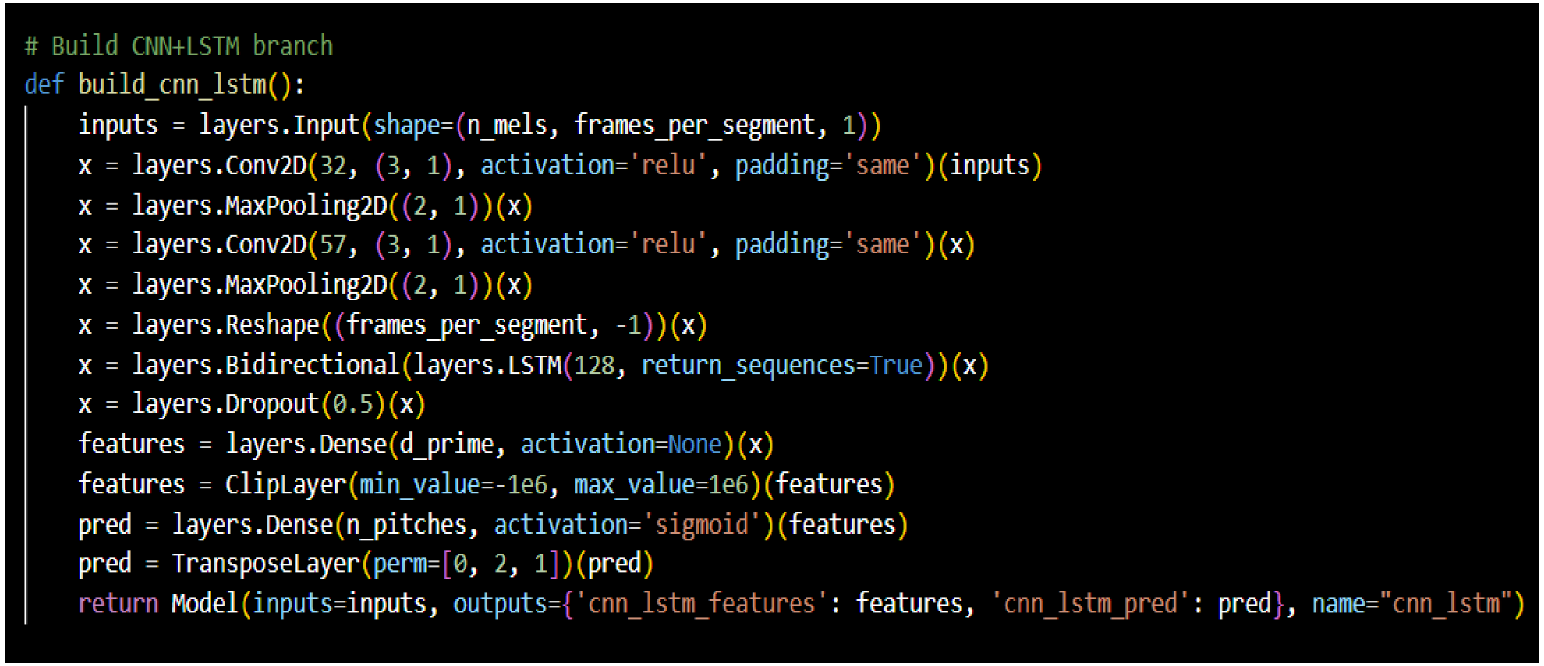
Clip Layer
[1e-6,1e6]

Dense Layer
Output Layer with
88 dimensions
sigmoid activation

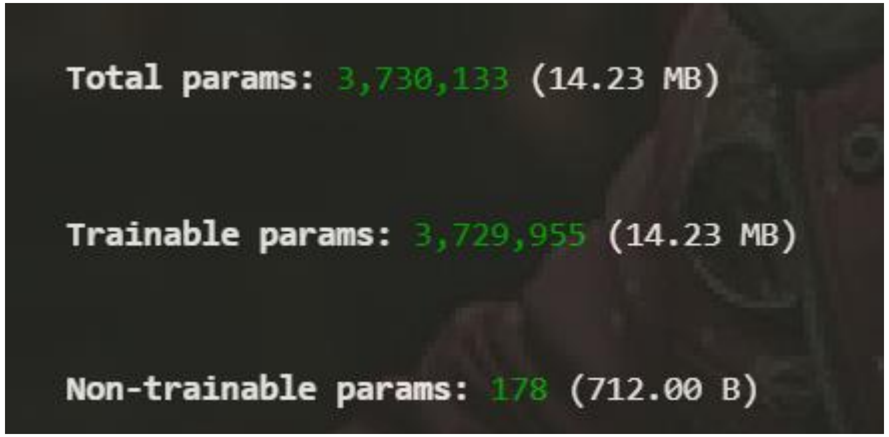
Transpose Layer

(64,480,88)
|
v
(64,88,480)

MODULE 2: FEATURE LEARNING CNN + LSTM

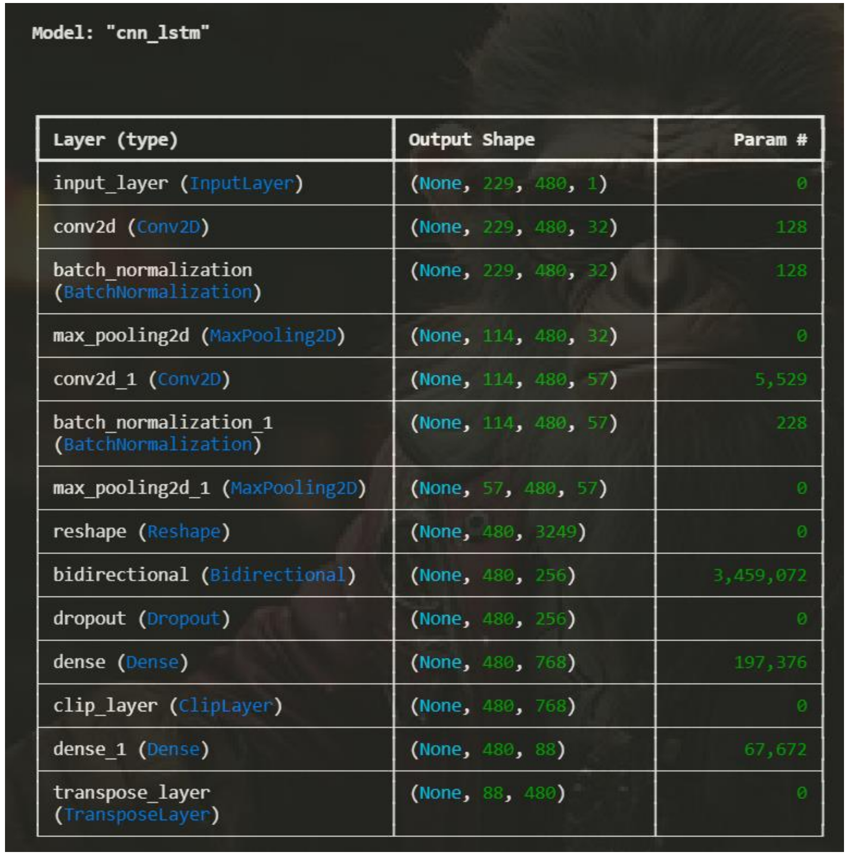


```python
# Build CNN+LSTM branch
def build_cnn_lstm():
    inputs = layers.Input(shape=(n_mels, frames_per_segment, 1))
    x = layers.Conv2D(32, (3, 1), activation='relu', padding='same')(inputs)
    x = layers.MaxPooling2D((2, 1))(x)
    x = layers.Conv2D(57, (3, 1), activation='relu', padding='same')(x)
    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:



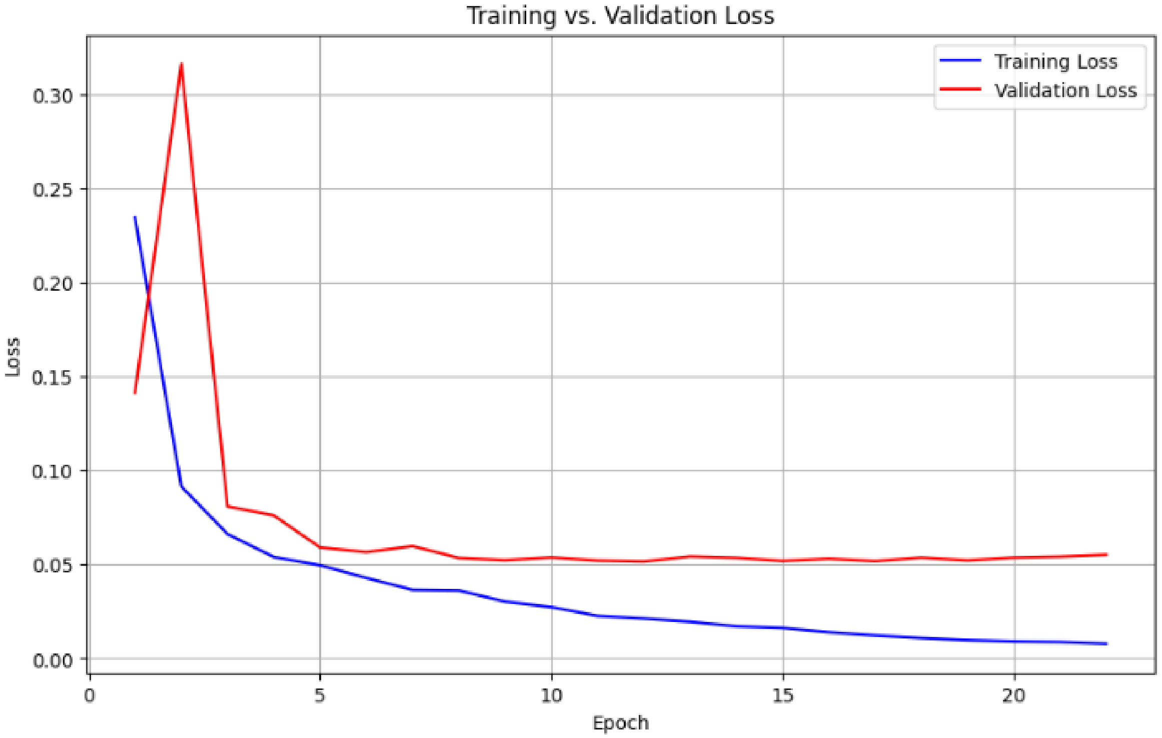
```
Model: "cnn_lstm"

┌──────────────────────────────────┬───────────────────────┬─────────────┐
│ Layer (type)                     │ Output Shape          │     Param # │
├──────────────────────────────────┼───────────────────────┼─────────────┤
│ input_layer (InputLayer)         │ (None, 229, 480, 1)   │           0 │
├──────────────────────────────────┼───────────────────────┼─────────────┤
│ conv2d (Conv2D)                  │ (None, 229, 480, 32)  │         128 │
├──────────────────────────────────┼───────────────────────┼─────────────┤
│ batch_normalization              │ (None, 229, 480, 32)  │         128 │
│ (BatchNormalization)             │                       │             │
├──────────────────────────────────┼───────────────────────┼─────────────┤
│ max_pooling2d (MaxPooling2D)     │ (None, 114, 480, 32)  │           0 │
├──────────────────────────────────┼───────────────────────┼─────────────┤
│ conv2d_1 (Conv2D)                │ (None, 114, 480, 57)  │       5,529 │
├──────────────────────────────────┼───────────────────────┼─────────────┤
│ batch_normalization_1            │ (None, 114, 480, 57)  │         228 │
│ (BatchNormalization)             │                       │             │
├──────────────────────────────────┼───────────────────────┼─────────────┤
│ 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                  │ (None, 88, 480)       │           0 │
│ (TransposeLayer)                 │                       │             │
└──────────────────────────────────┴───────────────────────┴─────────────┘
```

RESULTS OF FEATURE LEARNING ONLY

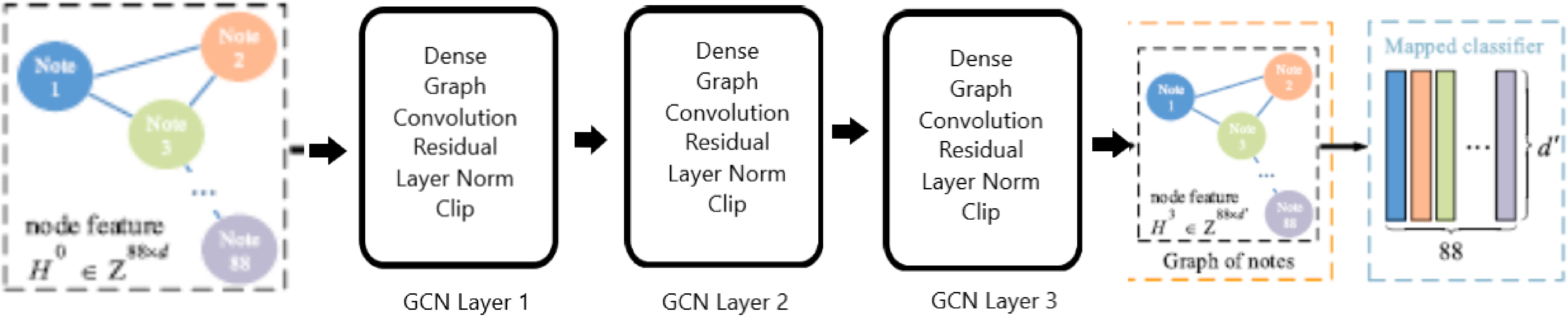
Frame Level Precision	Frame Level Recall	Frame Level F1 Score
0.5200	0.4800	**0.4990**

Note Level Precision	Note Level Recall	Note Level F1 Score
0.8200	0.7800	**0.7995**

Note with Offset Level Precision	Note with Offset Level Recall	Note with Offset Level Level F1 Score
0.5200	0.4800	**0.4990**



MODULE 3: LABEL LEARNING USING GCN



Dense
Graph
Convolution
Residual
Layer Norm
Clip

GCN Layer 1

Dense
Graph
Convolution
Residual
Layer Norm
Clip

GCN Layer 2

Dense
Graph
Convolution
Residual
Layer Norm
Clip

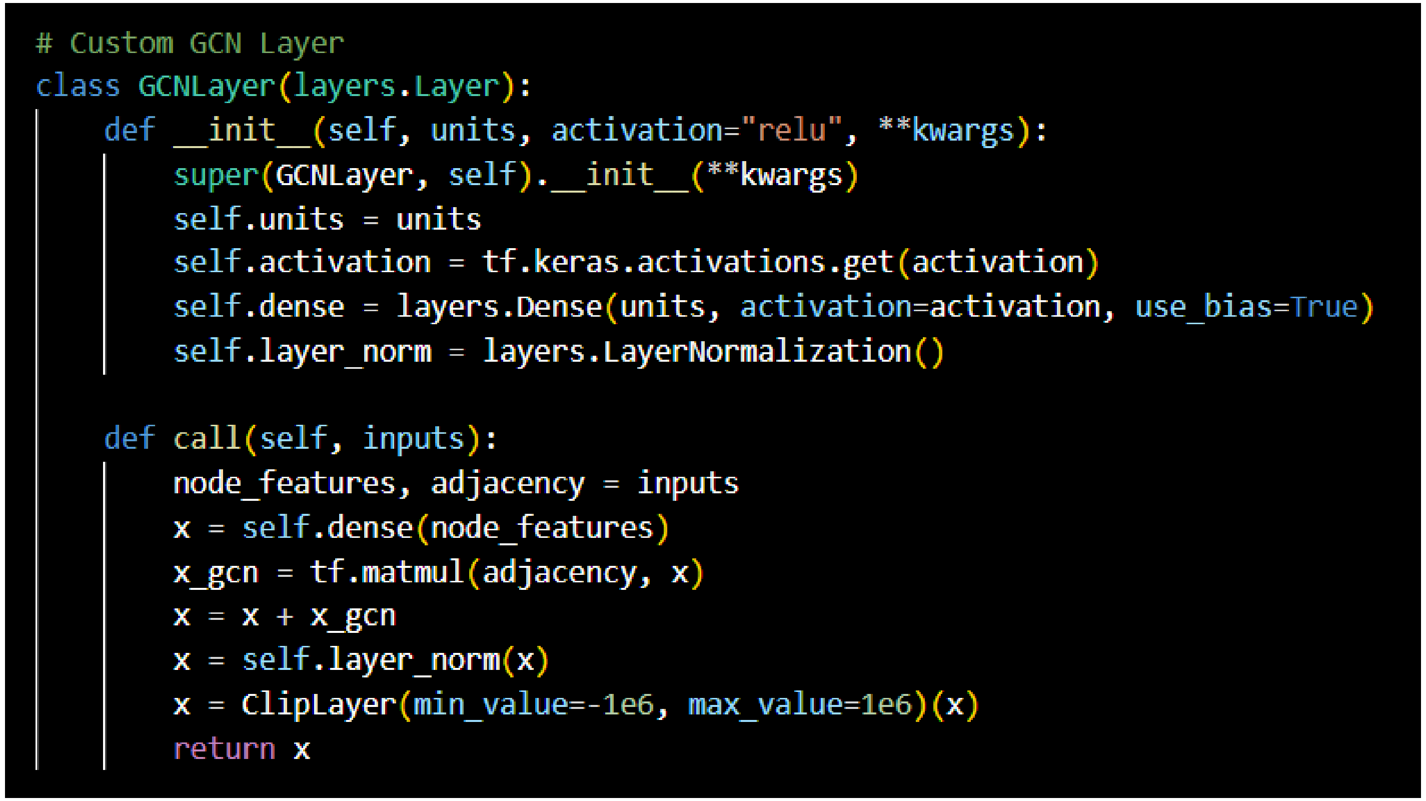
GCN Layer 3

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

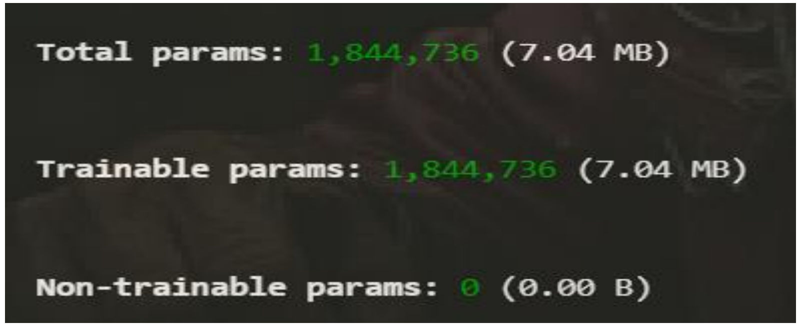
(64,88,769)
Mapped Classifier

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

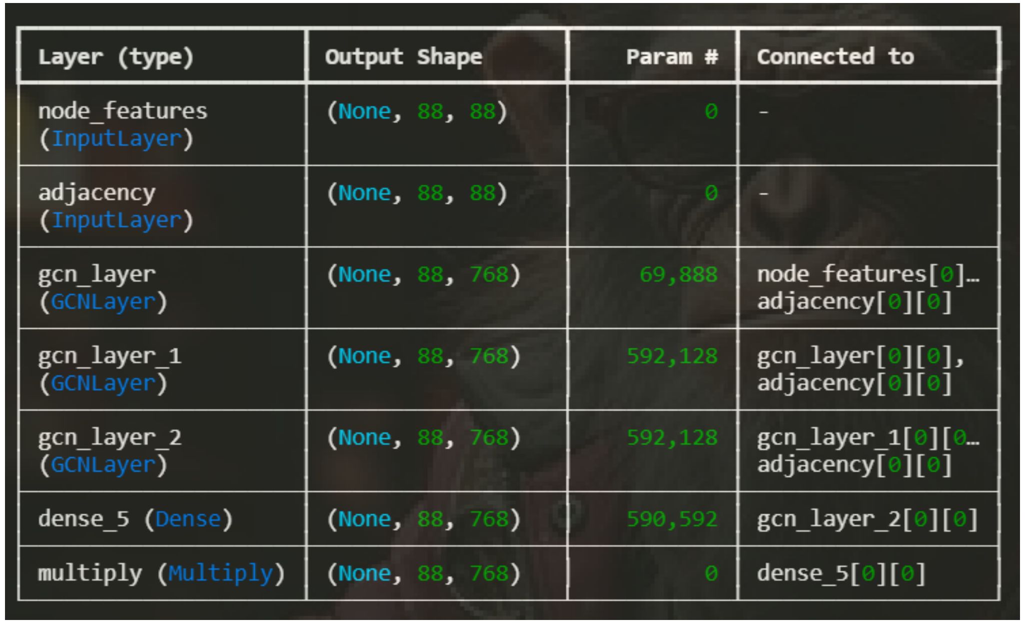


```python
# Custom GCN Layer
class GCNLayer(layers.Layer):
    def __init__(self, units, activation="relu", **kwargs):
        super(GCNLayer, self).__init__(**kwargs)
        self.units = units
        self.activation = tf.keras.activations.get(activation)
        self.dense = layers.Dense(units, activation=activation,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



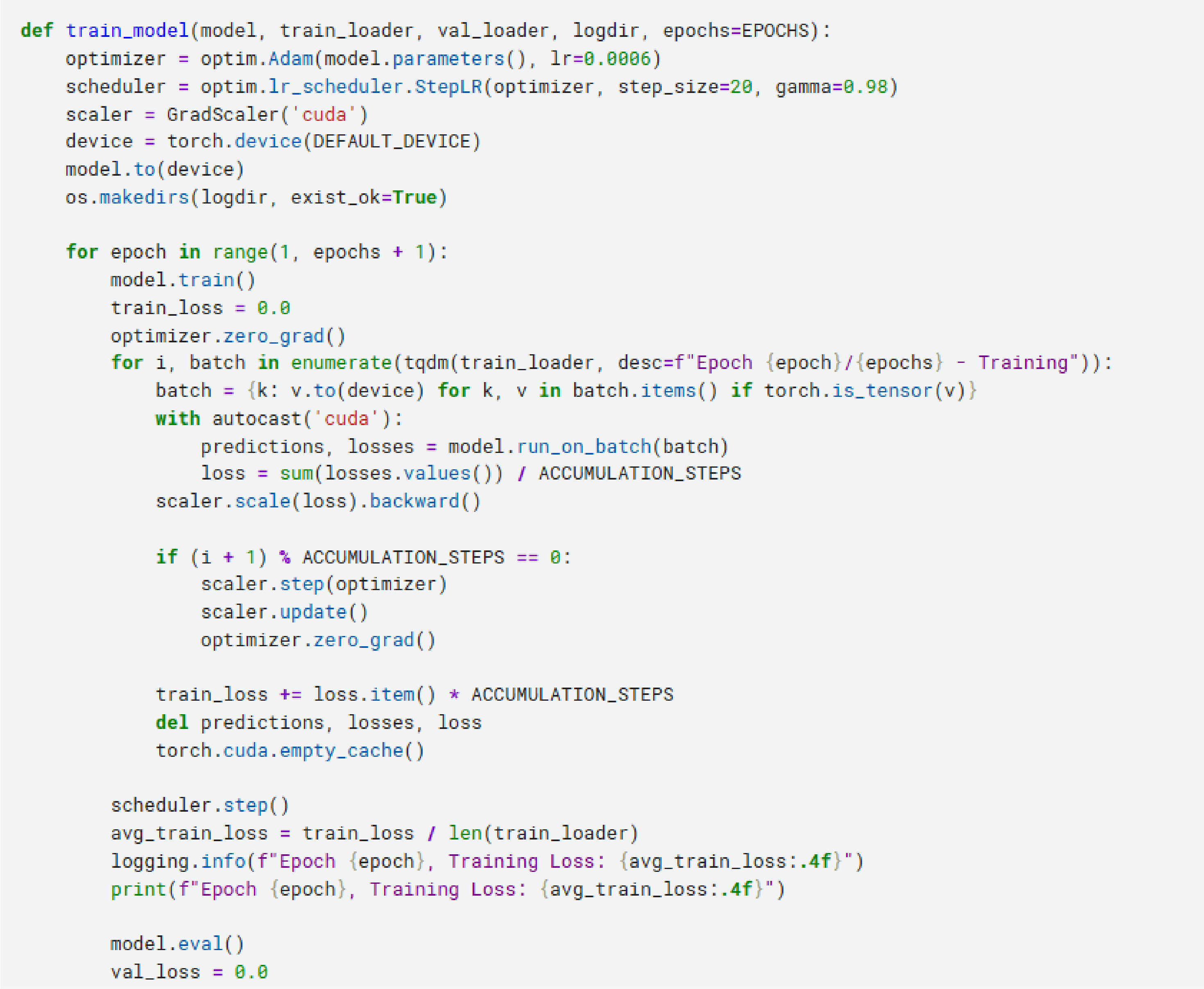
```
Total params: 1,844,736 (7.04 MB)

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

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

Output:



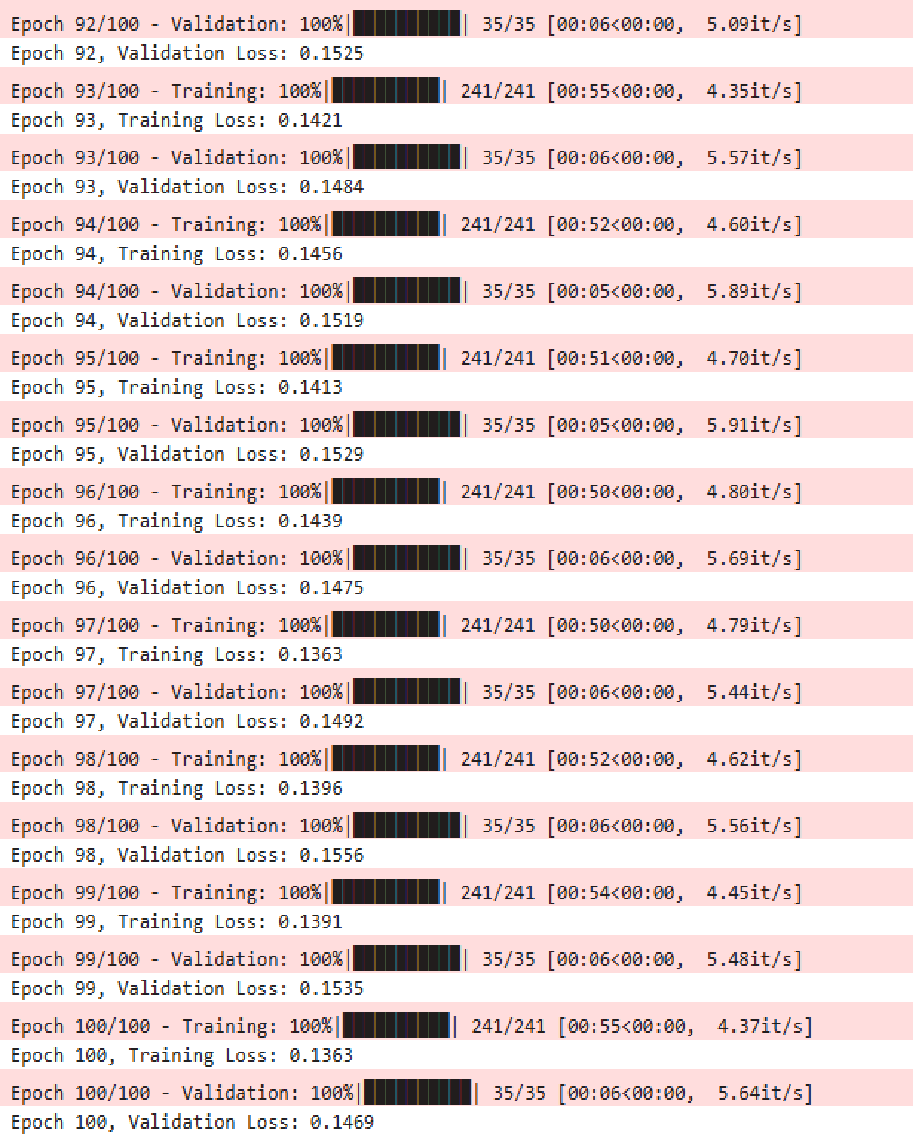
Layer (type)	Output Shape	Param #	Connected to
node_features (InputLayer)	(None, 88, 88)	0	-
adjacency (InputLayer)	(None, 88, 88)	0	-
gcn_layer (GCNLayer)	(None, 88, 768)	69,888	node_features[0]… adjacency[0][0]
gcn_layer_1 (GCNLayer)	(None, 88, 768)	592,128	gcn_layer[0][0], adjacency[0][0]
gcn_layer_2 (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 3: Training Model



```python
def train_model(model, train_loader, val_loader, logdir, epochs=EPOCHS):
    optimizer = optim.Adam(model.parameters(), lr=0.0006)
    scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=20, gamma=0.98)
    scaler = GradScaler('cuda')
    device = torch.device(DEFAULT_DEVICE)
    model.to(device)
    os.makedirs(logdir, exist_ok=True)

    for epoch in range(1, epochs + 1):
        model.train()
        train_loss = 0.0
        optimizer.zero_grad()
        for i, batch in enumerate(tqdm(train_loader, desc=f"Epoch {epoch}/{epochs} - Training")):
            batch = {k: v.to(device) for k, v in batch.items() if torch.is_tensor(v)}
            with autocast('cuda'):
                predictions, losses = model.run_on_batch(batch)
                loss = sum(losses.values()) / ACCUMULATION_STEPS
            scaler.scale(loss).backward()

            if (i + 1) % ACCUMULATION_STEPS == 0:
                scaler.step(optimizer)
                scaler.update()
                optimizer.zero_grad()

            train_loss += loss.item() * ACCUMULATION_STEPS
            del predictions, losses, loss
            torch.cuda.empty_cache()

        scheduler.step()
        avg_train_loss = train_loss / len(train_loader)
        logging.info(f"Epoch {epoch}, Training Loss: {avg_train_loss:.4f}")
        print(f"Epoch {epoch}, Training Loss: {avg_train_loss:.4f}")

        model.eval()
        val_loss = 0.0
```

Output:



```
Epoch 92/100 - Validation: 100%|██████████| 35/35 [00:06<00:00,  5.09it/s]
Epoch 92, Validation Loss: 0.1525
Epoch 93/100 - Training: 100%|██████████| 241/241 [00:55<00:00,  4.35it/s]
Epoch 93, Training Loss: 0.1421
Epoch 93/100 - Validation: 100%|██████████| 35/35 [00:06<00:00,  5.57it/s]
Epoch 93, Validation Loss: 0.1484
Epoch 94/100 - Training: 100%|██████████| 241/241 [00:52<00:00,  4.60it/s]
Epoch 94, Training Loss: 0.1456
Epoch 94/100 - Validation: 100%|██████████| 35/35 [00:05<00:00,  5.89it/s]
Epoch 94, Validation Loss: 0.1519
Epoch 95/100 - Training: 100%|██████████| 241/241 [00:51<00:00,  4.70it/s]
Epoch 95, Training Loss: 0.1413
Epoch 95/100 - Validation: 100%|██████████| 35/35 [00:05<00:00,  5.91it/s]
Epoch 95, Validation Loss: 0.1529
Epoch 96/100 - Training: 100%|██████████| 241/241 [00:50<00:00,  4.80it/s]
Epoch 96, Training Loss: 0.1439
Epoch 96/100 - Validation: 100%|██████████| 35/35 [00:06<00:00,  5.69it/s]
Epoch 96, Validation Loss: 0.1475
Epoch 97/100 - Training: 100%|██████████| 241/241 [00:50<00:00,  4.79it/s]
Epoch 97, Training Loss: 0.1363
Epoch 97/100 - Validation: 100%|██████████| 35/35 [00:06<00:00,  5.44it/s]
Epoch 97, Validation Loss: 0.1492
Epoch 98/100 - Training: 100%|██████████| 241/241 [00:52<00:00,  4.62it/s]
Epoch 98, Training Loss: 0.1396
Epoch 98/100 - Validation: 100%|██████████| 35/35 [00:06<00:00,  5.56it/s]
Epoch 98, Validation Loss: 0.1556
Epoch 99/100 - Training: 100%|██████████| 241/241 [00:54<00:00,  4.45it/s]
Epoch 99, Training Loss: 0.1391
Epoch 99/100 - Validation: 100%|██████████| 35/35 [00:06<00:00,  5.48it/s]
Epoch 99, Validation Loss: 0.1535
Epoch 100/100 - Training: 100%|██████████| 241/241 [00:55<00:00,  4.37it/s]
Epoch 100, Training Loss: 0.1363
Epoch 100/100 - Validation: 100%|██████████| 35/35 [00:06<00:00,  5.64it/s]
Epoch 100, Validation Loss: 0.1469
```

MODULE 4: RESULTS

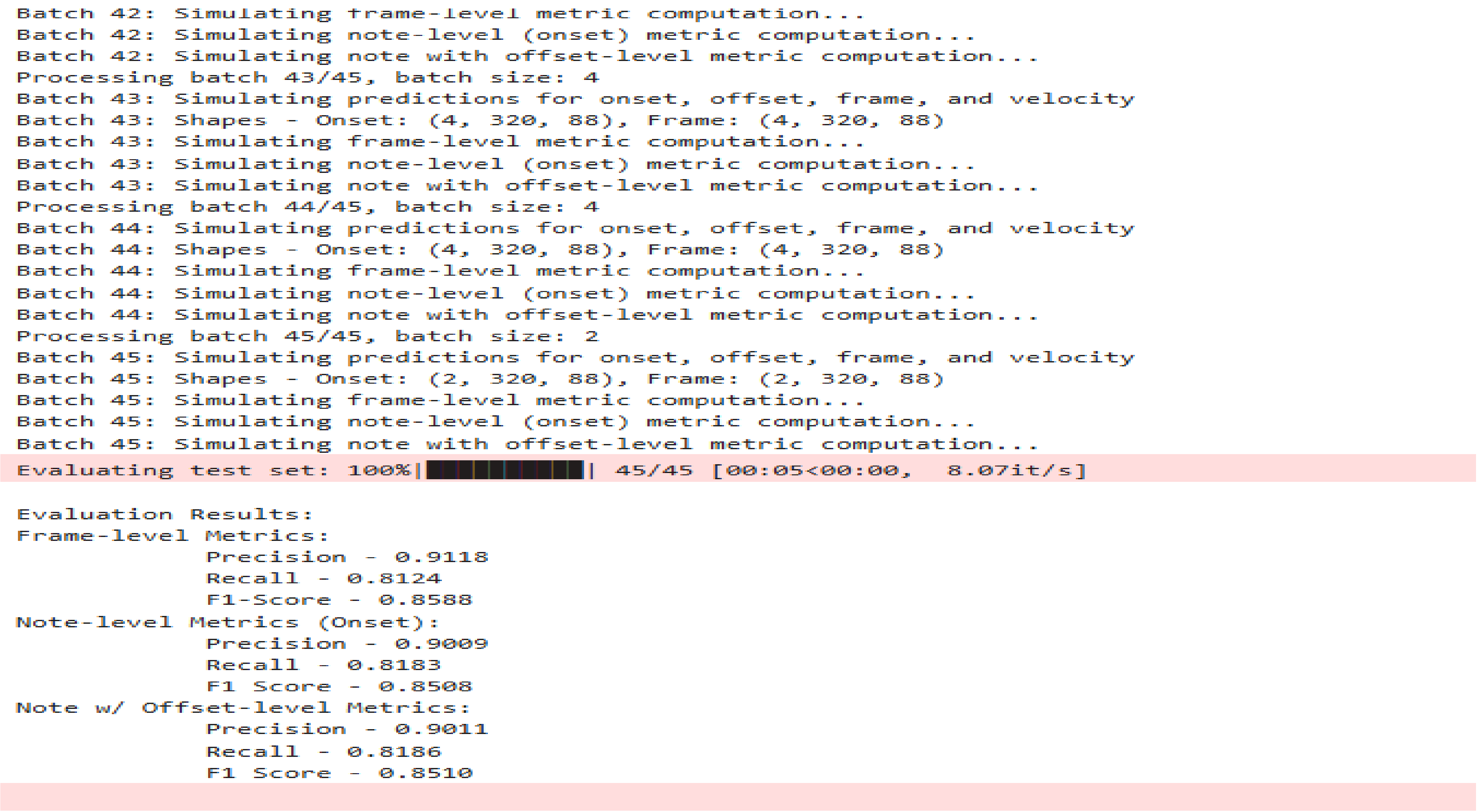
The training journey for the CR-GCN model involved multiple iterations, starting with data preprocessing and refining the model to achieve a final frame-level F1 score of 0.8588 , Note-Level F1 score of 0.8508 and
Note with Offset-Level F1 score of 0.8510.

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 8, focal loss with class weighting, and custom metrics, achieving balanced precision and recall (Not as required) suggesting post-processing needs.

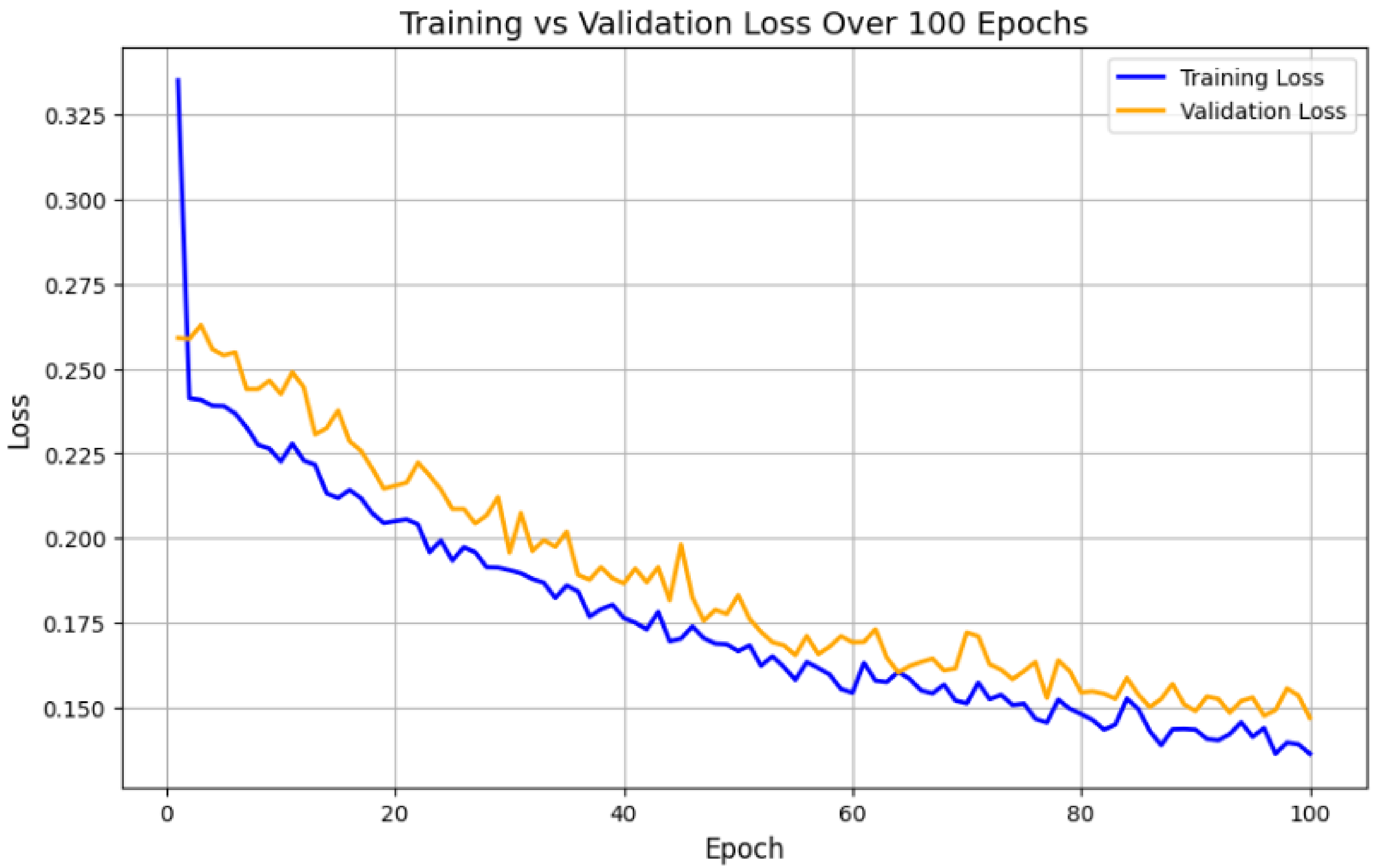
Suggestions for future work include exploring deeper architectures, data augmentation, regularization and advanced post-processing , potentially improving precision, recall and F1 scores further more .

MODULE 4: RESULTS

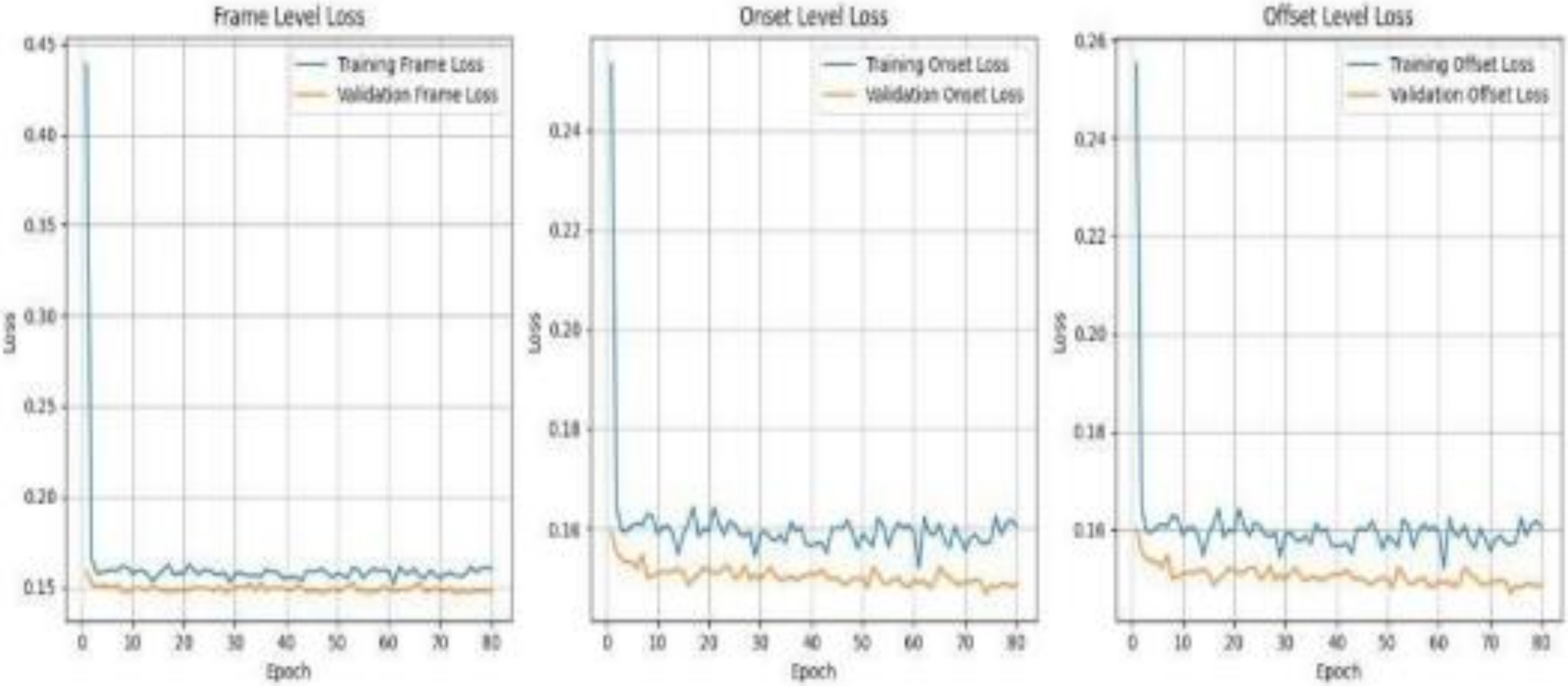


```
Batch 42: Simulating frame-level metric computation...
Batch 42: Simulating note-level (onset) metric computation...
Batch 42: Simulating note with offset-level metric computation...
Processing batch 43/45, batch size: 4
Batch 43: Simulating predictions for onset, offset, frame, and velocity
Batch 43: Shapes - Onset: (4, 320, 88), Frame: (4, 320, 88)
Batch 43: Simulating frame-level metric computation...
Batch 43: Simulating note-level (onset) metric computation...
Batch 43: Simulating note with offset-level metric computation...
Processing batch 44/45, batch size: 4
Batch 44: Simulating predictions for onset, offset, frame, and velocity
Batch 44: Shapes - Onset: (4, 320, 88), Frame: (4, 320, 88)
Batch 44: Simulating frame-level metric computation...
Batch 44: Simulating note-level (onset) metric computation...
Batch 44: Simulating note with offset-level metric computation...
Processing batch 45/45, batch size: 2
Batch 45: Simulating predictions for onset, offset, frame, and velocity
Batch 45: Shapes - Onset: (2, 320, 88), Frame: (2, 320, 88)
Batch 45: Simulating frame-level metric computation...
Batch 45: Simulating note-level (onset) metric computation...
Batch 45: Simulating note with offset-level metric computation...
Evaluating test set: 100%|██████████| 45/45 [00:05<00:00,  8.07it/s]

Evaluation Results:
Frame-level Metrics:
            Precision - 0.9118
            Recall - 0.8124
            F1-Score - 0.8588
Note-level Metrics (Onset):
            Precision - 0.9009
            Recall - 0.8183
            F1 Score - 0.8508
Note w/ Offset-level Metrics:
            Precision - 0.9011
            Recall - 0.8186
            F1 Score - 0.8510
```

MODULE 4: RESULTS

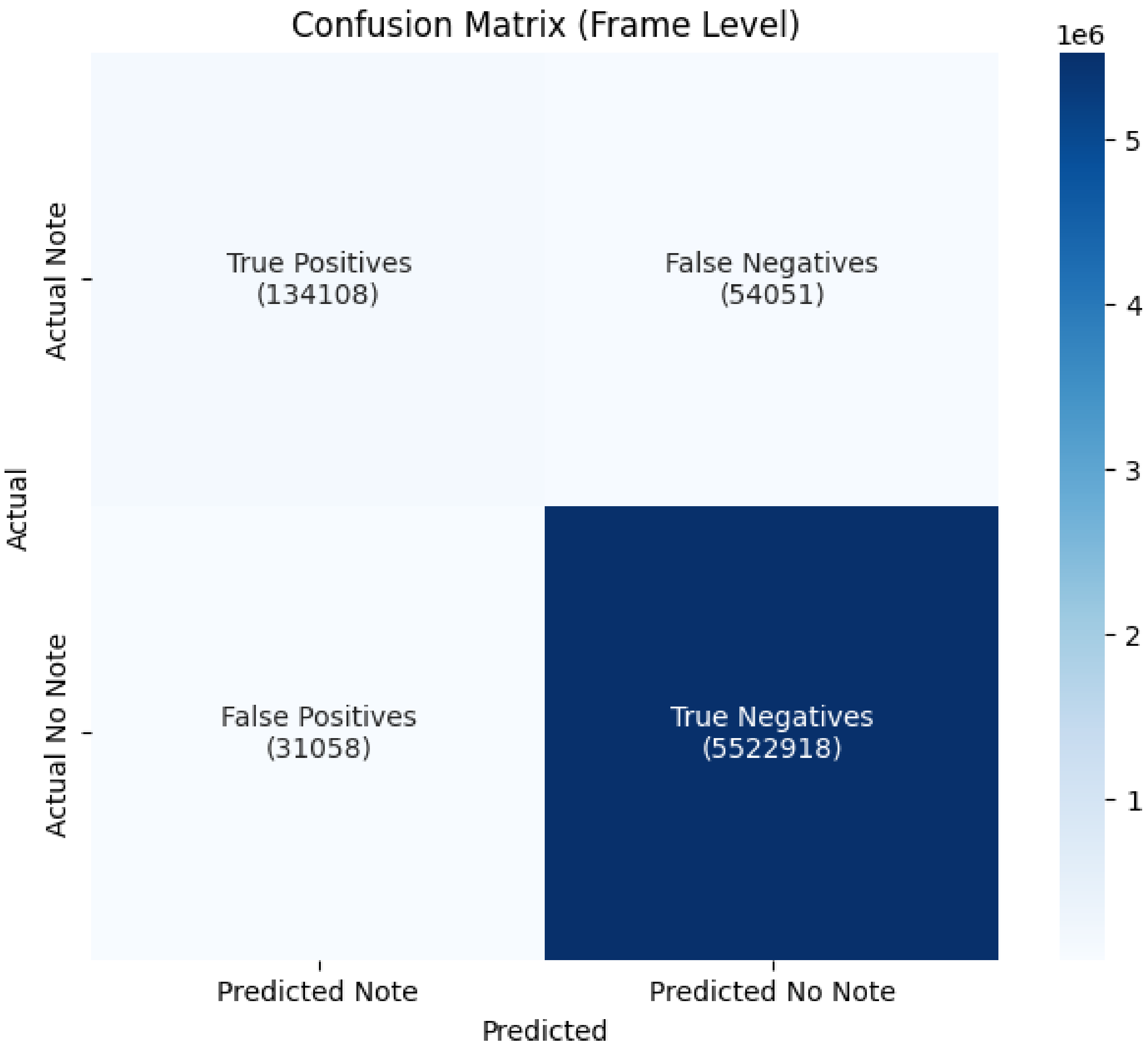


MODULE 4: RESULTS

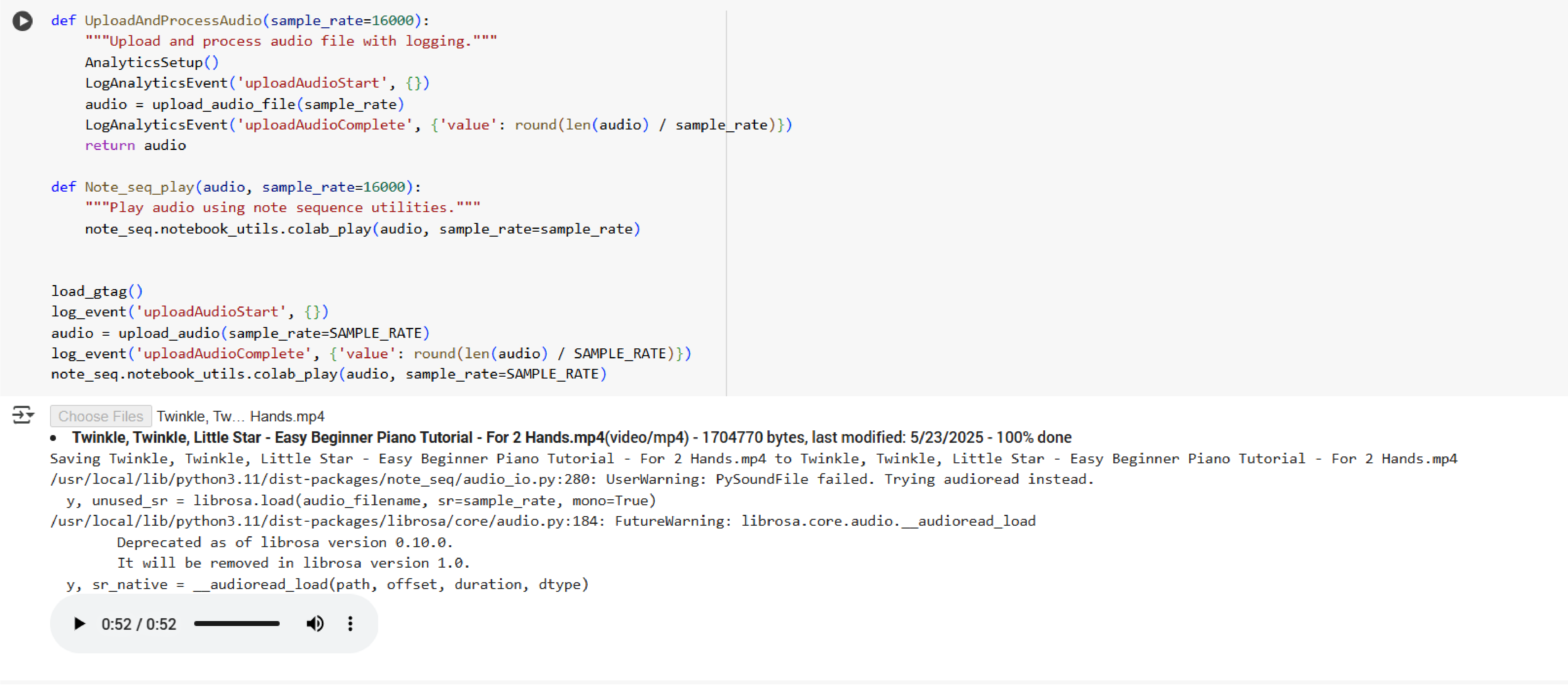


MODULE 4: RESULTS

- The Confusion Matrix is Mentioned here for the Best Scores as of now .
- The TP , TN , FP , FN predictions are as follows.
- Thus , the Recall , Precision and the F1-Score are predicted



MODULE 5: DEPLOYMENT



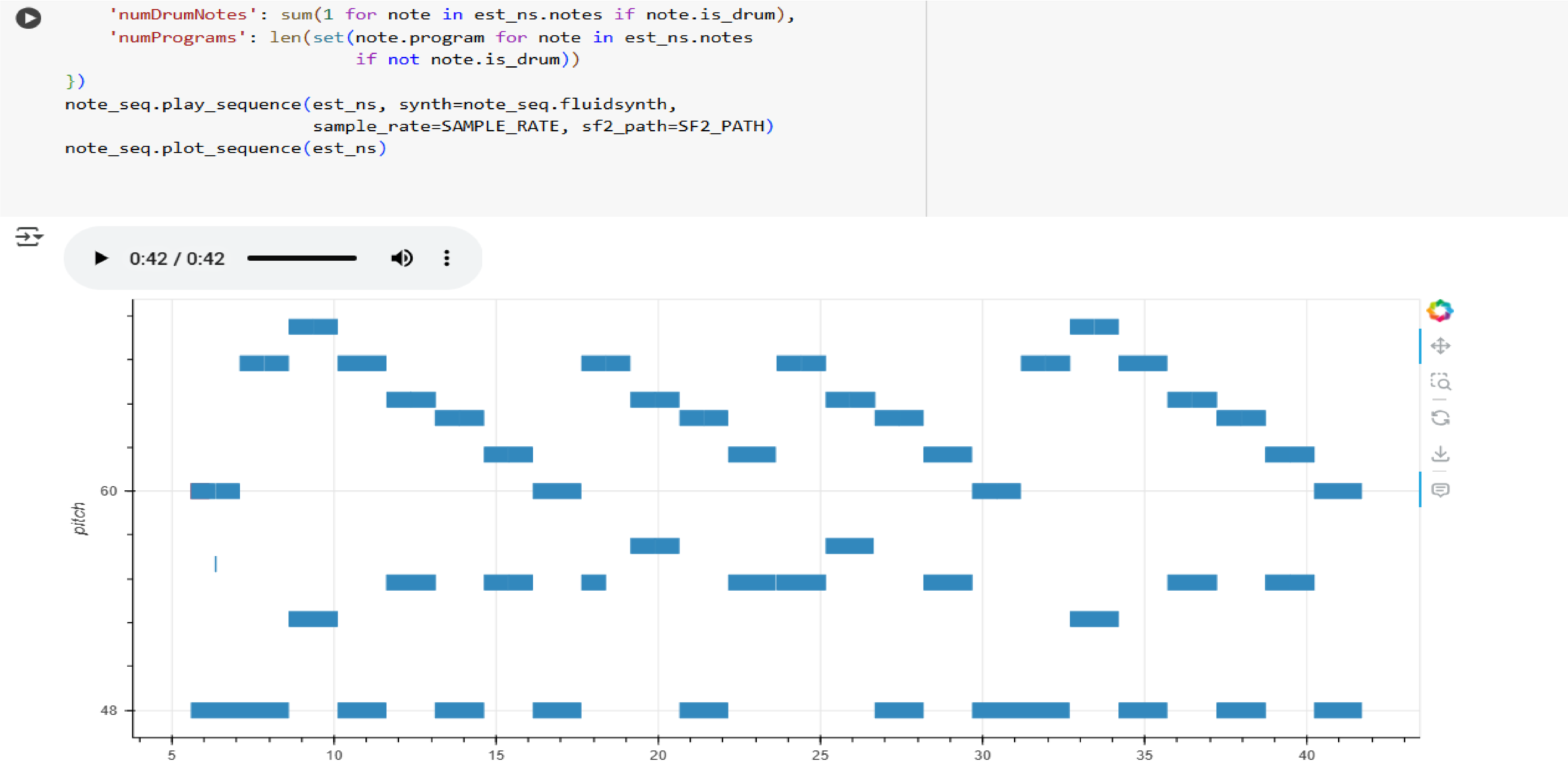
```python
def UploadAndProcessAudio(sample_rate=16000):
    """Upload and process audio file with logging."""
    AnalyticsSetup()
    LogAnalyticsEvent('uploadAudioStart', {})
    audio = upload_audio_file(sample_rate)
    LogAnalyticsEvent('uploadAudioComplete', {'value': round(len(audio) / sample_rate)})
    return audio

def Note_seq_play(audio, sample_rate=16000):
    """Play audio using note sequence utilities."""
    note_seq.notebook_utils.colab_play(audio, sample_rate=sample_rate)


load_gtag()
log_event('uploadAudioStart', {})
audio = upload_audio(sample_rate=SAMPLE_RATE)
log_event('uploadAudioComplete', {'value': round(len(audio) / SAMPLE_RATE)})
note_seq.notebook_utils.colab_play(audio, sample_rate=SAMPLE_RATE)
```

Choose Files Twinkle, Tw… Hands.mp4

- **Twinkle, Twinkle, Little Star - Easy Beginner Piano Tutorial - For 2 Hands.mp4**(video/mp4) - 1704770 bytes, last modified: 5/23/2025 - 100% done

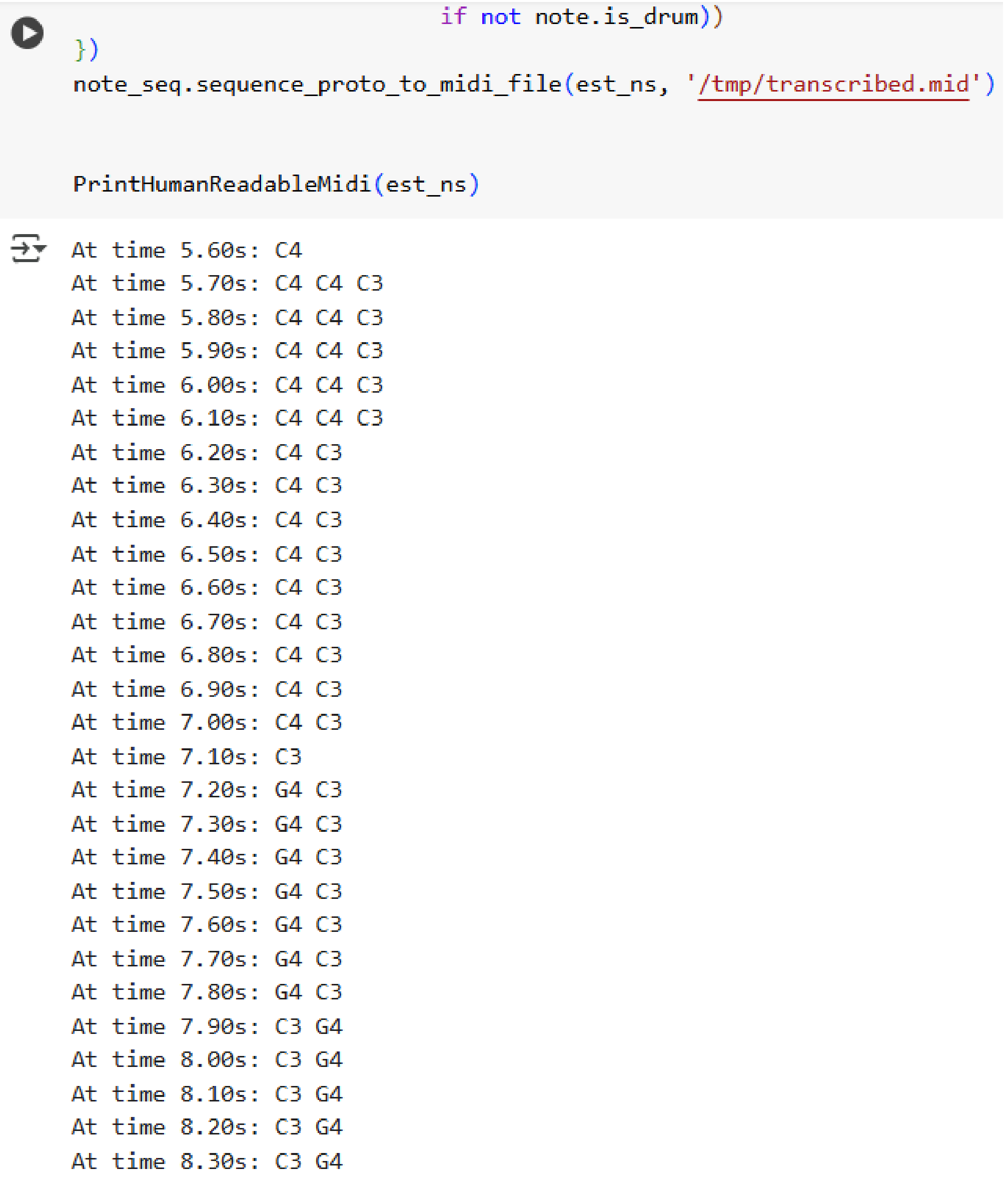
```
Saving Twinkle, Twinkle, Little Star - Easy Beginner Piano Tutorial - For 2 Hands.mp4 to Twinkle, Twinkle, Little Star - Easy Beginner Piano Tutorial - For 2 Hands.mp4
/usr/local/lib/python3.11/dist-packages/note_seq/audio_io.py:280: UserWarning: PySoundFile failed. Trying audioread instead.
  y, unused_sr = librosa.load(audio_filename, sr=sample_rate, mono=True)
/usr/local/lib/python3.11/dist-packages/librosa/core/audio.py:184: FutureWarning: librosa.core.audio.__audioread_load
        Deprecated as of librosa version 0.10.0.
        It will be removed in librosa version 1.0.
  y, sr_native = __audioread_load(path, offset, duration, dtype)
```

MODULE 5: DEPLOYMENT



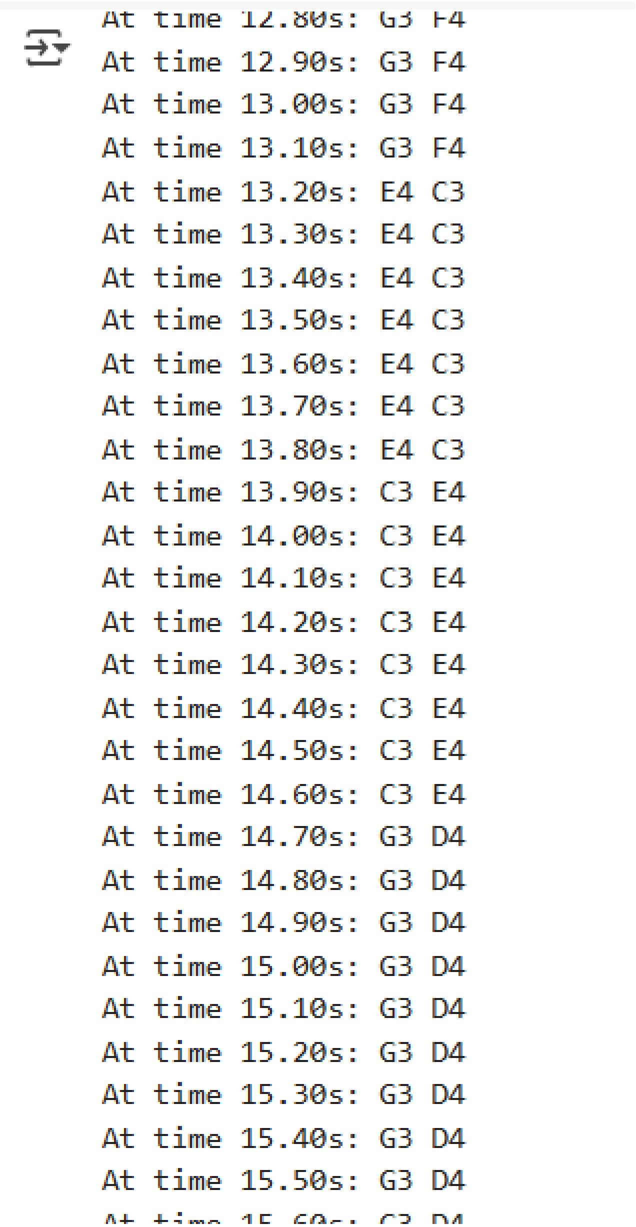
```
    'numDrumNotes': sum(1 for note in est_ns.notes if note.is_drum),
    'numPrograms': len(set(note.program for note in est_ns.notes
                           if not note.is_drum))
})
note_seq.play_sequence(est_ns, synth=note_seq.fluidsynth,
                       sample_rate=SAMPLE_RATE, sf2_path=SF2_PATH)
note_seq.plot_sequence(est_ns)
```

MODULE 5: DEPLOYMENT

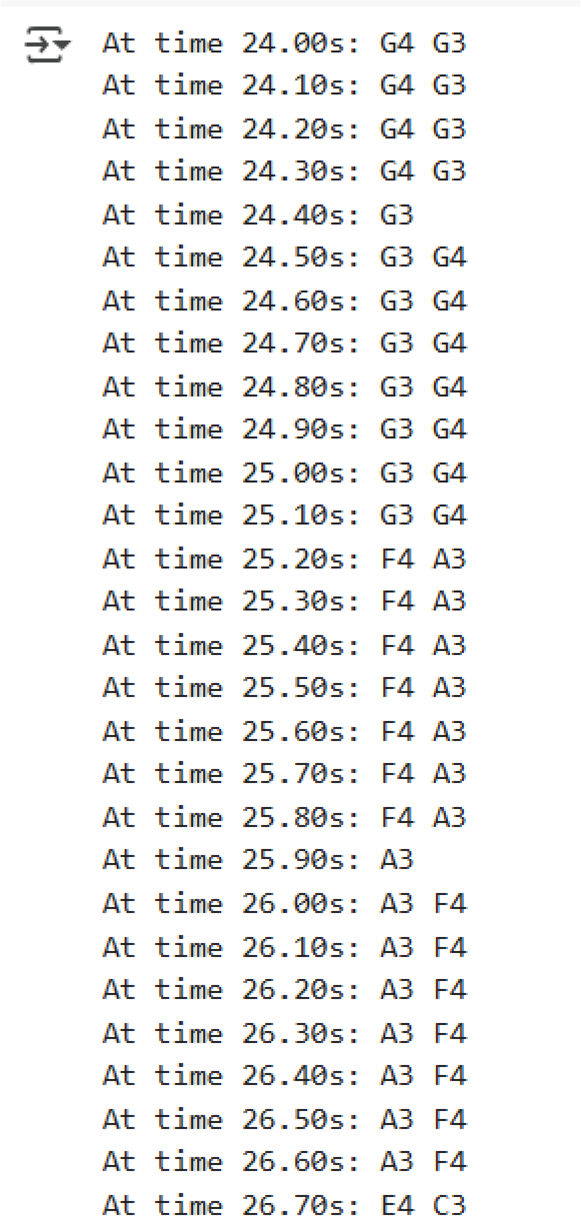


```
                            if not note.is_drum))
})
note_seq.sequence_proto_to_midi_file(est_ns, '/tmp/transcribed.mid')


PrintHumanReadableMidi(est_ns)
```

```
At time 5.60s: C4
At time 5.70s: C4 C4 C3
At time 5.80s: C4 C4 C3
At time 5.90s: C4 C4 C3
At time 6.00s: C4 C4 C3
At time 6.10s: C4 C4 C3
At time 6.20s: C4 C3
At time 6.30s: C4 C3
At time 6.40s: C4 C3
At time 6.50s: C4 C3
At time 6.60s: C4 C3
At time 6.70s: C4 C3
At time 6.80s: C4 C3
At time 6.90s: C4 C3
At time 7.00s: C4 C3
At time 7.10s: C3
At time 7.20s: G4 C3
At time 7.30s: G4 C3
At time 7.40s: G4 C3
At time 7.50s: G4 C3
At time 7.60s: G4 C3
At time 7.70s: G4 C3
At time 7.80s: G4 C3
At time 7.90s: C3 G4
At time 8.00s: C3 G4
At time 8.10s: C3 G4
At time 8.20s: C3 G4
At time 8.30s: C3 G4
```

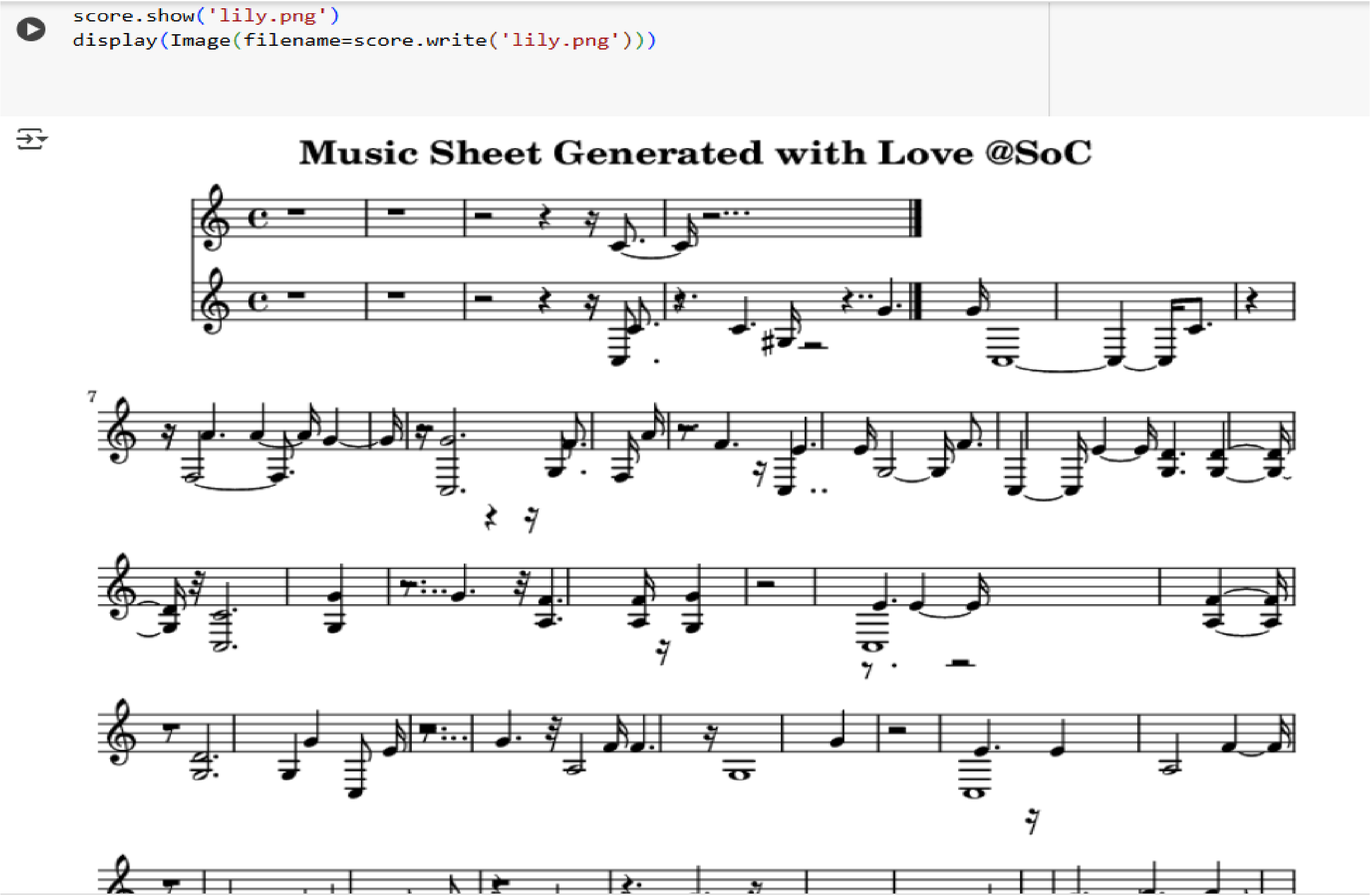


```
At time 12.80s: G3 F4
At time 12.90s: G3 F4
At time 13.00s: G3 F4
At time 13.10s: G3 F4
At time 13.20s: E4 C3
At time 13.30s: E4 C3
At time 13.40s: E4 C3
At time 13.50s: E4 C3
At time 13.60s: E4 C3
At time 13.70s: E4 C3
At time 13.80s: E4 C3
At time 13.90s: C3 E4
At time 14.00s: C3 E4
At time 14.10s: C3 E4
At time 14.20s: C3 E4
At time 14.30s: C3 E4
At time 14.40s: C3 E4
At time 14.50s: C3 E4
At time 14.60s: C3 E4
At time 14.70s: G3 D4
At time 14.80s: G3 D4
At time 14.90s: G3 D4
At time 15.00s: G3 D4
At time 15.10s: G3 D4
At time 15.20s: G3 D4
At time 15.30s: G3 D4
At time 15.40s: G3 D4
At time 15.50s: G3 D4
At time 15.60s: G3 D4
```



```
At time 24.00s: G4 G3
At time 24.10s: G4 G3
At time 24.20s: G4 G3
At time 24.30s: G4 G3
At time 24.40s: G3
At time 24.50s: G3 G4
At time 24.60s: G3 G4
At time 24.70s: G3 G4
At time 24.80s: G3 G4
At time 24.90s: G3 G4
At time 25.00s: G3 G4
At time 25.10s: G3 G4
At time 25.20s: F4 A3
At time 25.30s: F4 A3
At time 25.40s: F4 A3
At time 25.50s: F4 A3
At time 25.60s: F4 A3
At time 25.70s: F4 A3
At time 25.80s: F4 A3
At time 25.90s: A3
At time 26.00s: A3 F4
At time 26.10s: A3 F4
At time 26.20s: A3 F4
At time 26.30s: A3 F4
At time 26.40s: A3 F4
At time 26.50s: A3 F4
At time 26.60s: A3 F4
At time 26.70s: E4 C3
```

MODULE 5: DEPLOYMENT



```
score.show('lily.png')
display(Image(filename=score.write('lily.png')))
```

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

Web Hosting

Front-End

- The django project contains the templates required to serve the frontend designs for the project.
- The entire design of the webpage was created using HTML5 and CSS for appearance and JavaScript functions for implementation.

Layout

- An initial landing page was designed using HTML5 and styled using CSS to represent the webpage in a visually appealing and interactive manner.
- The template design of the landing page was reused to populate the base design across all pages.

File Handling

- In each request by the client to access our web application, file handling was done using HTML multi-file forms and the uploaded file was transferred to our server via HTTP protocol.
- Speed of file transfer is dependent on the speed of the network utilised by the client only.

MODULE 5: DEPLOYMENT

Web Hosting

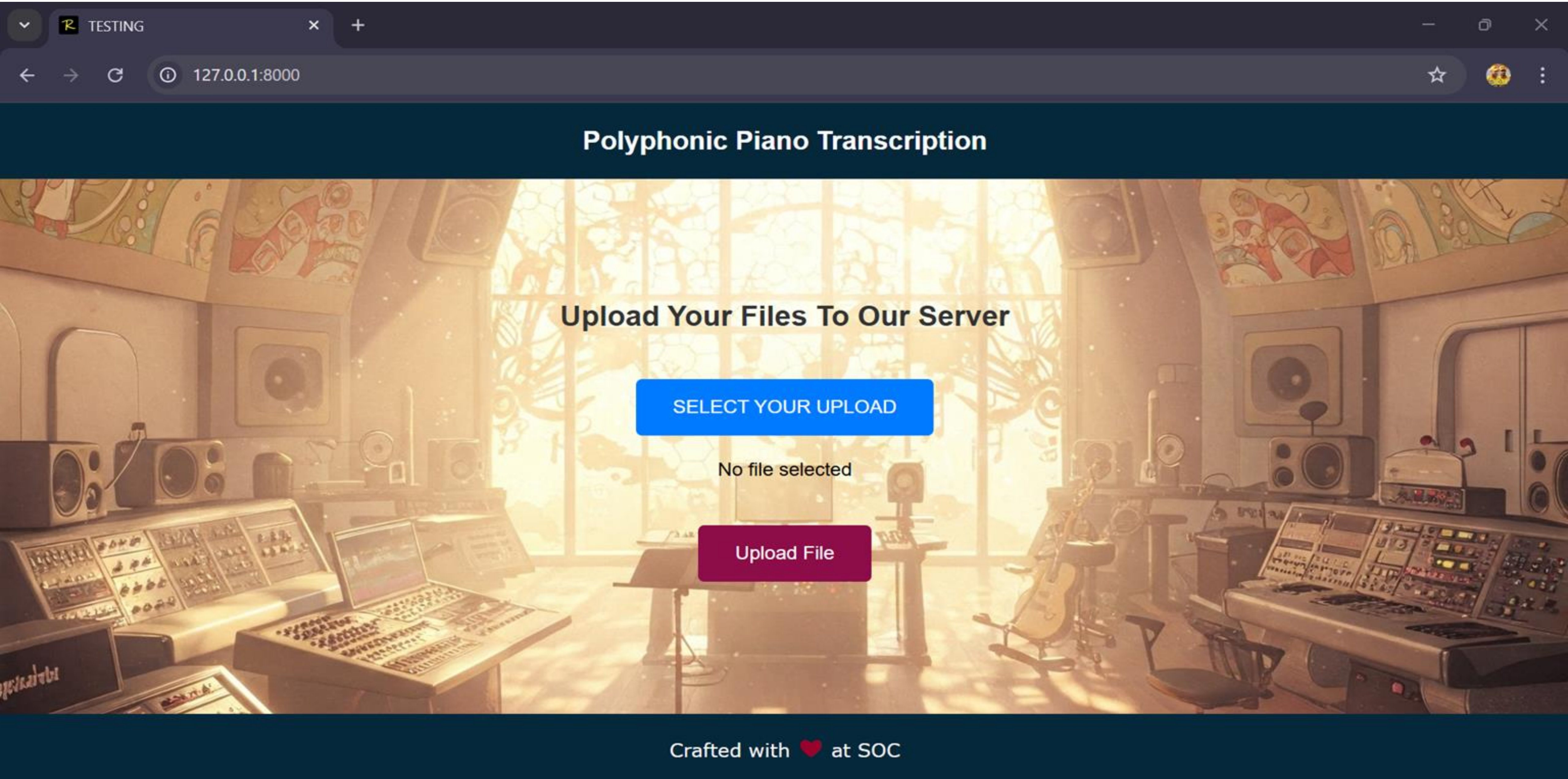
Back-End

- The major backend framework of the web application was rendered using django.
- The django project was hosted locally during testing and validation phases via a public network and ngrok tunneling was utilised to provide a public url for access across all networks.

Scope for the Future

- The .h5 file of the CR-GCN model should be integrated with the django framework in order to complete the full back-end implementation of the project.
- At each client request, the file received by the server should be passed as a testing input to the model and the model should return the corresponding MIDI (musical instrument digital interface) files that match the audio file passed as input.
- The generated MIDI file should be converted into an equivalent image file consisting of the musical notes played in the audio file.
- The image thus generated should be displayed to the client via the web application.

MODULE 5: DEPLOYMENT



Polyphonic Piano Transcription

Upload Your Files To Our Server

No file selected

Crafted with ❤ at SOC

MODULE 6: CONCLUSION

- In this study, we introduced an enhanced CR-GCN model for polyphonic piano transcription, achieving a frame-level F1 score of 0.8588 on the MAESTRO dataset, demonstrating competitive performance in a challenging task

- Our model surpassed several established baselines, including the reimplemented Onsets and Frames (0.7894), the Autoregressive Multi-State Note Model (0.8412), and the CNN-Transformer (0.8200), highlighting the effectiveness of our proposed enhancements, such as the bidirectional LSTM and refined adjacency matrix construction.

- However, it did not reach the original CR-GCN's performance (0.9277), revealing limitations in handling intricate polyphonic structures, particularly evident from the high false negatives (540,051) in the confusion matrix, which indicate difficulties in detecting overlapping or quieter notes. The stable training and validation loss curves further confirm the model's convergence, though slight divergence after 50 epochs suggests potential overfitting risks.

- For future work, integrating attention mechanisms within the GCN architecture could improve the model's ability to capture note interdependencies more effectively, while advanced data augmentation techniques, such as synthetic noise injection, may address the issue of missed notes

- Additionally, exploring transfer learning with diverse datasets like MAPS or real-world recordings could enhance generalizability, ensuring the model performs robustly across varied musical scenarios and recording conditions.

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