Deep Learning Based Automatic Music Transcription using CR-GCN

Report submitted to the SASTRA Deemed to be University
as the requirement for the course

CSE300 - MINI PROJECT

Submitted by
Selva Karthik S
(Reg No.:126003238, CSE)
Shanthosh Kumar
(Reg No.:126003241, CSE)
Rithvik L
(Reg No.:126003218, CSE)

SCHOOL OF COMPUTING

THANJAVUR, TAMIL NADU, INDIA – 613 401

May 2025

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

SCHOOL OF COMPUTING

THANJAVUR – 613 401

Bonafide Certificate

This is to certify that the report titled "Deep Learning Based Automatic Music Transcription using CR-GCN" submitted as a requirement for the course, CSE300 : MINI PROJECT for B.Tech. is a Bonafide record of the work done by Mr**. Selva Karthik** (Reg. No.:126003238, CSE), Mr. **Shanthosh Kumar** (Reg No.:126003241, CSE), Mr. **Rithvik L** (Reg. No.:126003218, CSE) during the academic year 2024-25, in the School of Computing, under my supervision.

Signature of Project Supervisor:

Name with Affiliation: Dr. Emily Jenifer A, Asst. Professor III, School of Computing

Date:

Mini Project Viva voice held on _____

Examiner 1 **Examiner 2**

Acknowledgements

We would like to thank our Honorable Chancellor **Prof. R. Sethuraman** for providing us with an opportunity and the necessary infrastructure for carrying out this project as a part of our curriculum.

We would like to thank our Honorable Vice-Chancellor **Dr. S. Vaidhyasubramaniam** and **Dr. S. Swaminathan**, Dean, Planning & Development, for the encouragement and strategic support at every step of our college life.

We extend our sincere thanks to **Dr. R. Chandramouli**, Registrar, SASTRA Deemed to be University for providing the opportunity to pursue this project.

We extend our heartfelt thanks to **Dr. V. S. Shankar Sriram**, Dean, School of Computing, **Dr. R. Muthaiah**, Associate Dean, Research, **Dr. K. Ramkumar**, Associate Dean, Academics, **Dr. D. Manivannan**, Associate Dean, Infrastructure, **Dr. R. Alageswaran**, Associate Dean, Students Welfare.

Our guide **Dr. Emily Jenifer A**, Asst. Professor III, School of Computing was the driving force behind this whole idea from the start. Her deep insight in the field and invaluable suggestions helped us in making progress throughout our project work. We also thank the project review panel members for their valuable comments and insights which made this project better.

We would like to extend our gratitude to all the teaching and non-teaching faculties of the School of Computing who have either directly or indirectly helped us in the completion of the project.

We gratefully acknowledge all the contributions and encouragement from my family and friends resulting in the successful completion of this project. We thank you all for providing us an opportunity to showcase our skills through projects.

LIST OF FIGURES

Figure No.	Title	Page No.
3.1	Overall framework of CR-GCN Model	3
3.2	Graph, adjacency matrix and degree matrix	3
3.3	Preprocessing MAESTRO v2 dataset	5
3.4	Verification of preprocessed dataset	5
3.5	Mel Spectrogram of a Preprocessed sample	6
3.6	Piano Roll of a Preprocessed sample	6
3.7	Note frequency distribution across splits	7
3.8	Velocity Distribution across splits	7
3.9	Overall CR-GCN Model Class	10
3.10	Generates an 88x88 co-occurrence matrix for the CR-GCN model using MAESTRO dataset's train split	11
3.11	Convolutional Stack layer of CR-GCN	12
3.12	Bi-LSTM layer of CR-GCN Model	13
3.13	GCN Layer of CR-GCN Model	13
3.14	Model Training Code	16
3.15	Training Epochs	16
4.1	views.py file	17
4.2	Multipart forms (html)	18
4.3	Multipart forms (javascript)	18
4.4	UI of web application	19
4.5	Logs of terminal during web application deployment	19
4.6	Chord Detection Code	20

4.7	Chord Detection	21
4.8	LilyPond Environment Augmentation Code	22
4.9	Music Sheet Generation	22
5.1	Confusion Matrix for Frame level Predictions	23
5.2	Training and Validation loss vs Epochs across three different losses	25

LIST OF TABLES

Table No.	Title	Page No.
3.1	MAESTRO v2 Dataset Metadata	8
3.2	Summary of CR-GCN Model and No. of Parameters	15
5.1	Evaluation Metrics of CR-GCN	23
5.2	Comparative Analysis on AMT Models	25

ABBREVIATIONS

AMT	Automatic Music Transcription
CNN	Convolutional Neural Network
GCN	Graph Convolutional Network
RNN	Recurrent Neural Network
LSTM	Long Short-Term Memory
CR-GCN	Channel Relationship-Based Graph Convolutional Network
MIR	Music Information Retrieval
MAESTRO	MIDI and Audio Edited for Synchronous Tracks and Organization
FFT	Fast Fourier Transform
t-SNE	t-distributed Stochastic Neighbor Embedding
URL	Uniform Resource Locator

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 show that the proposed methodology with F1-score of 92.77% 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.

Key Words: Automatic Music Transcription (AMT), Convolutional Neural Network (CNN), Graph Convolutional Network (GCN), bidirectional LSTMs, F1-score, Channel Relationship-Based Graph Convolutional Network (CR-GCN)

TABLE OF CONTENTS

Title	Page No.
Bonafide Certificate	(ii)
Acknowledgements	(iii)
List of Figures	(iv)
List of Tables	(v)
Abbreviations	(vi)
Abstract	(vii)
1. Summary of the base paper	1
2. Merits and Demerits of the base paper	2
3. Methodology	3
4. Deployment	17
5. Results and Discussions	23
6. Conclusions	26
7. References	27

1. SUMMARY OF BASE PAPER

Title: Polyphonic Piano Transcription Based on Graph Convolutional Network

Authors: Zhe Xiao, Xin Chen, Li Zhou

Journal: Signal Processing
Volume: 212
Year: 2023
Paper ID: 109134
ISSN: 0165-1684
Indexed in: SCI

The base paper introduces CR-GCN (Convolutional-Recurrent Graph Convolutional Network), a novel deep learning-based framework for automatic music transcription (AMT) of polyphonic piano recordings. The study addresses challenges such as dimension explosion, temporal continuity, and sparsity in music signals by integrating multiple strategies: feature extraction through convolutional neural networks (CNNs), temporal modeling using bidirectional long short-term memory (BiLSTM), and label dependency learning with Graph Convolutional Networks (GCN).

To further enhance transcription accuracy, the paper establishes a data-driven adjacency matrix for GCN to model implicit dependencies between musical notes, improving multi-note classification. The framework is trained on the MAESTRO dataset, demonstrating state-of-the-art performance in both frame-level and note-level transcription compared to traditional feature-based, statistical model-based, spectral decomposition-based, and deep learning-based methods. The proposed CR-GCN model particularly excels in capturing harmonic relationships between notes, leading to more accurate transcription of complex polyphonic piano pieces.

The paper concludes that integrating a music language model based on GCN with acoustic feature extraction significantly improves transcription performance. It highlights future research directions such as expanding the model to more diverse musical genres, refining adjacency matrix learning for better generalization, and incorporating explainable AI techniques for improved interpretability in music transcription.

2. MERITS AND DEMERITS OF BASE PAPER

Merits

- **Integration of Graph Convolutional Networks (GCN):** The paper introduces GCN to model interdependencies between musical notes, addressing the problem of dimension explosion in polyphonic piano transcription effectively.
- **Hybrid Deep Learning Framework:** The proposed CR-GCN model integrates convolutional, recurrent, and graph-based neural networks, using their combined strengths for improved transcription accuracy.
- **Data-Driven Adjacency Matrix:** Traditional models that treat each note as independent, but this paper establishes an adjacency matrix based on real co-occurrence patterns in music datasets, enhancing accuracy and interpretability.
- **Superior Frame-Level and Note-Level Performance:** Experimental results demonstrate that CR-GCN outperforms traditional and deep learning-based approaches, particularly in detecting chord structures and polyphonic music.
- **Robust Performance on Large Datasets:** The model is tested on the MAESTRO dataset, showing strong generalizability across various piano compositions.

Demerits

- **Computational Overhead:** The CR-GCN framework involves multiple deep learning components (CNN, RNN, and GCN), increasing training complexity and hardware requirements.
- **Sensitivity to Adjacency Matrix Threshold:** The performance of the model is highly dependent on the threshold value used to construct the adjacency matrix, requiring careful tuning for optimal results.
- **Challenges in Random Chord Transcription:** The paper acknowledges difficulties in transcribing random chords outside structured music compositions due to limitations in the learned note dependencies.

3. METHODOLOGY

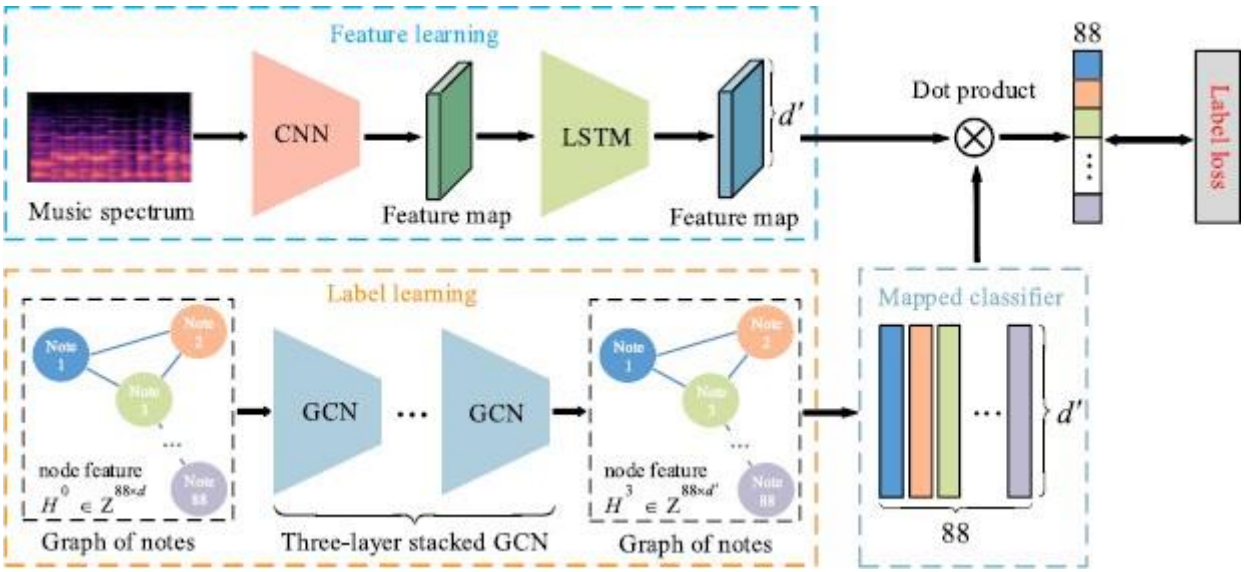

Figure 3.1 Overall framework of CR-GCN Model

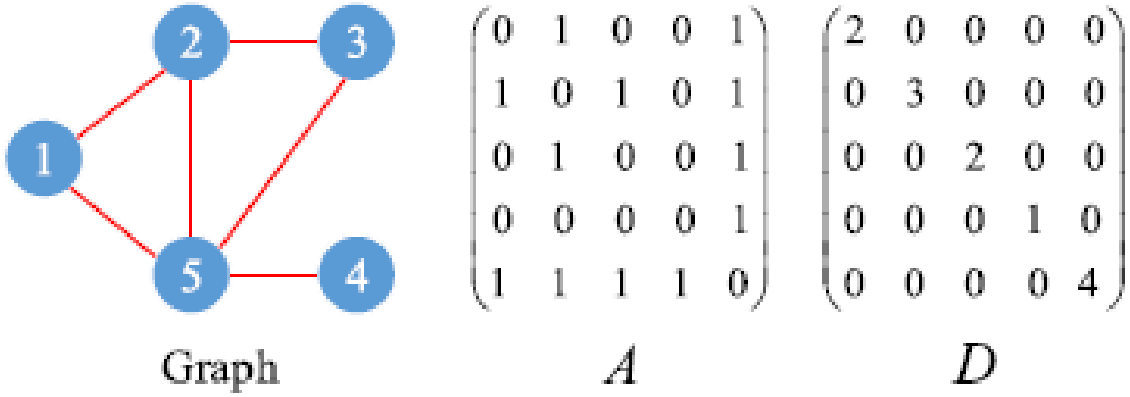

Figure 3.2 Graph, adjacency matrix and degree matrix

3.1 Data Preprocessing

The preprocessing pipeline for the MAESTRO v2.0.0 dataset transforms raw audio and MIDI files into PyTorch-compatible .pt files and .tsv files, enabling training of a music transcription model. It begins with robust MIDI parsing using the mido library to extract note events (onset, offset, note, velocity) while handling sustain pedal effects, followed by audio loading with librosa at a 16000 Hz sample rate, mono conversion, normalization, and scaling to 16-bit integers (multiplied by 32768.0). Piano roll labels are generated as matrices encoding onsets (3), frames (2), and offsets (1) across time steps calculated via the formula:

$$Frame\ Index = \text{round}(\frac{\text{Time} * \text{SAMPLE\_RATE}}{HOP\_LENGTH})$$

with velocity stored separately. Extensive validation ensures MIDI-audio duration alignment, non-empty labels, and distinct onset-frame representations, logging failures for transparency as seen in **Figure 3.3**. Outputs are organized into train, validation, and test directories, with a verification step confirming the integrity of .pt files as seen in **Figure 3.4**.

```
def preprocess_maestro():
    """Preprocess MAESTRO v2 dataset."""
    dataset_dir = r'E:\College\3rd Year\SEM 6\Mini Project\Dataset\maestro-v2.0.0\maestro-v2.0.0'
    output_dir = r'E:\College\3rd Year\SEM 6\Mini Project\Dataset\maestro-v2.0.0\preprocessed_cr_gcn'
    metadata_path = os.path.join(dataset_dir, 'maestro-v2.0.0.json')
    try:
        with open(metadata_path, 'r') as f:
            metadata = json.load(f)
        metadata = pd.DataFrame(metadata)
    except Exception as e:
        print(f"Error loading metadata {metadata_path}: {e}")
        return
    metadata['audio_path'] = metadata['audio_filename'].apply(lambda x: os.path.join(dataset_dir, x))
    metadata['midi_path'] = metadata['midi_filename'].apply(lambda x: os.path.join(dataset_dir, x))
    missing_audio = metadata[~metadata['audio_path'].apply(os.path.exists)]
    missing_midi = metadata[~metadata['midi_path'].apply(os.path.exists)]
    create_directories(output_dir)
    if not missing_audio.empty:
        print(f"Missing {len(missing_audio)} audio files")
        with open(os.path.join(output_dir, 'missing_audio.txt'), 'w') as f:
            f.write('\n'.join(missing_audio['audio_filename']))
    if not missing_midi.empty:
        print(f"Missing {len(missing_midi)} MIDI files")
        with open(os.path.join(output_dir, 'missing_midi.txt'), 'w') as f:
            f.write('\n'.join(missing_midi['midi_filename']))
    splits = ['train', 'validation', 'test']
    for split in splits:
        split_data = metadata[metadata['split'] == split]
        split_output = os.path.join(output_dir, split)
        print(f"\nProcessing {split} split ({len(split_data)} files)")
        progress_bar = tqdm(total=len(split_data), desc=f"Processing {split}", unit="file")
        failed_files = []
        for _, row in split_data.iterrows():
            audio_path = row['audio_path']
            midi_path = row['midi_path']
```

```
        midi_path = row['midi_path']
        base_name = os.path.splitext(os.path.basename(audio_path))[0]
        tsv_output = os.path.join(split_output, f"{base_name}.tsv")
        pt_output = os.path.join(split_output, f"{base_name}.pt")
        if os.path.exists(pt_output):
            try:
                data = torch.load(pt_output)
                if all(k in data for k in ['path', 'audio', 'label', 'velocity']):
                    progress_bar.update(1)
                    continue
            except Exception:
                pass
        audio, audio_length, audio_success = preprocess_audio(audio_path)
        if not audio_success:
            failed_files.append(base_name)
            print(f"Failed to process audio {base_name}")
            progress_bar.update(1)
            continue
        midi_data = parse_midi(midi_path)
        if midi_data.size == 0:
            failed_files.append(base_name)
            print(f"Empty MIDI data for {midi_path}")
            progress_bar.update(1)
            continue
        max_midi_time = np.max(midi_data[:, 1]) if midi_data.size > 0 else 0
        audio_duration = audio_length / SAMPLE_RATE
        if max_midi_time > audio_duration + 0.5:
            failed_files.append(base_name)
            print(f"MIDI duration {max_midi_time}s exceeds audio {audio_duration}s for {midi_path}")
            progress_bar.update(1)
            continue
        label, velocity, onsets, frames, offsets, message = generate_labels(audio_length, midi_data)
        if label is None:
            failed_files.append(base_name)
            print(f"Label validation failed for {midi_path}: {message}")
```

Figure 3.3 Preprocessing MAESTRO v2 dataset

```
def verify_dataset(output_dir):
    """Verify preprocessed dataset."""
    for split in ['train', 'validation', 'test']:
        split_dir = os.path.join(output_dir, split)
        pt_files = list(Path(split_dir).glob('*.pt'))
        tsv_files = set(f.stem for f in Path(split_dir).glob('*.tsv'))
        print(f"\nVerifying {split} split ({len(pt_files)} PT files)")

        for pt_file in pt_files:
            try:
                data = torch.load(pt_file)
                if not all(k in data for k in ['path', 'audio', 'label', 'velocity']):
                    print(f"Invalid PT file: {pt_file}")
                    continue
                if data['label'].shape[1] != (MAX_MIDI - MIN_MIDI + 1):
                    print(f"Invalid label shape in {pt_file}: {data['label'].shape}")
                if data['velocity'].shape != data['label'].shape:
                    print(f"Velocity shape mismatch in {pt_file}")
                if pt_file.stem not in tsv_files:
                    print(f"Missing TSV for {pt_file}")
            except Exception as e:
                print(f"Error loading {pt_file}: {e}")

if __name__ == "__main__":
    preprocess_maestro()
    verify_dataset(r'E:\College\3rd Year\SEM 6\Mini Project\Dataset\maestro-v2.0.0\preprocessed_cr_gcn')
```

Figure 3.4 Verification of preprocessed dataset

3.2 Dataset Visualization

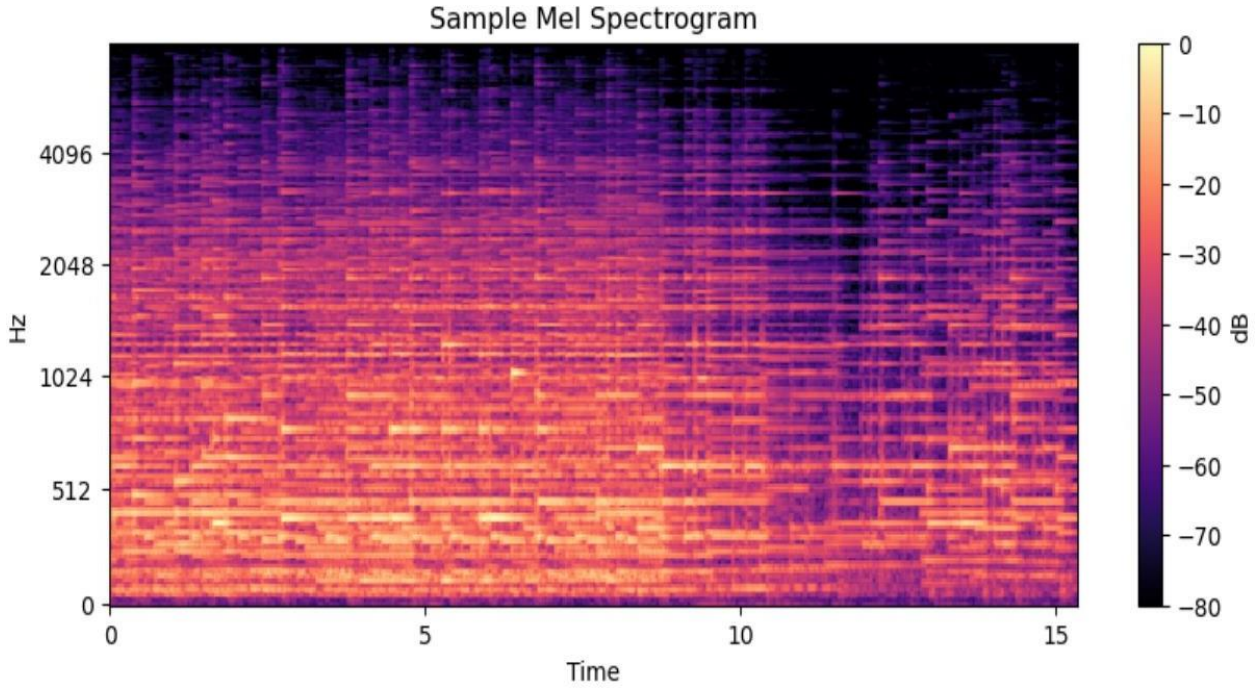

Figure 3.5 Mel Spectrogram of a Preprocessed sample

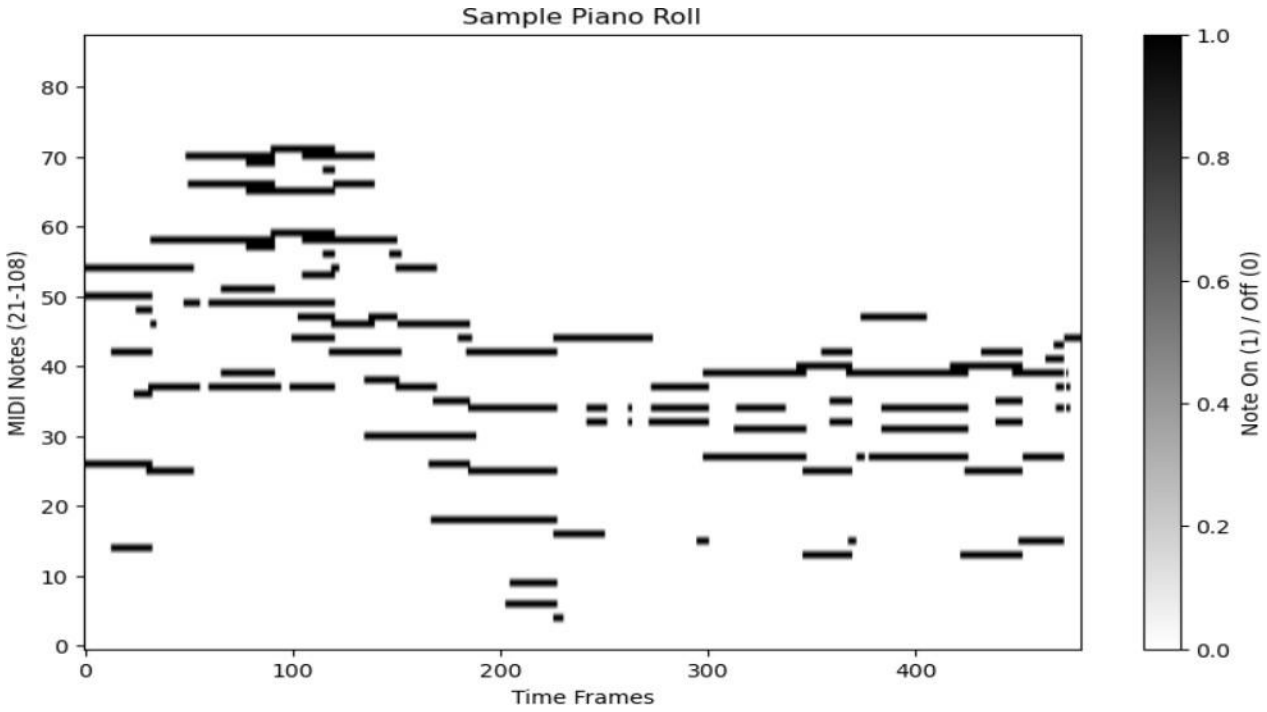

Figure 3.6 Piano Roll of a Preprocessed sample

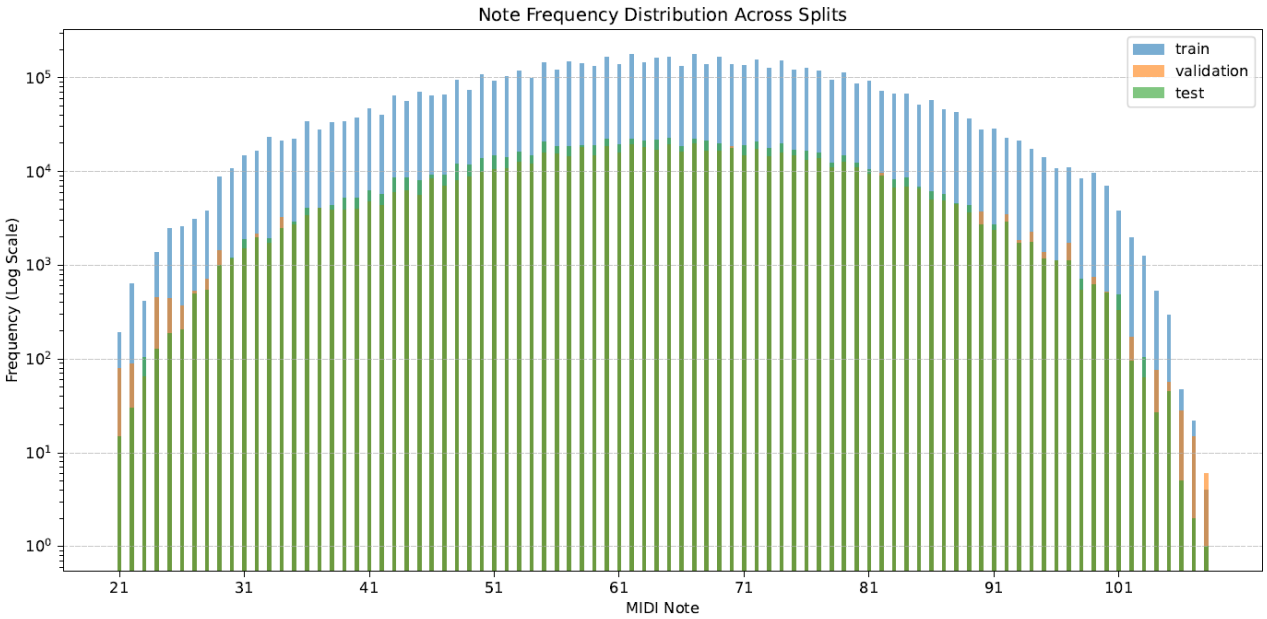

Figure 3.7 Note frequency distribution across splits

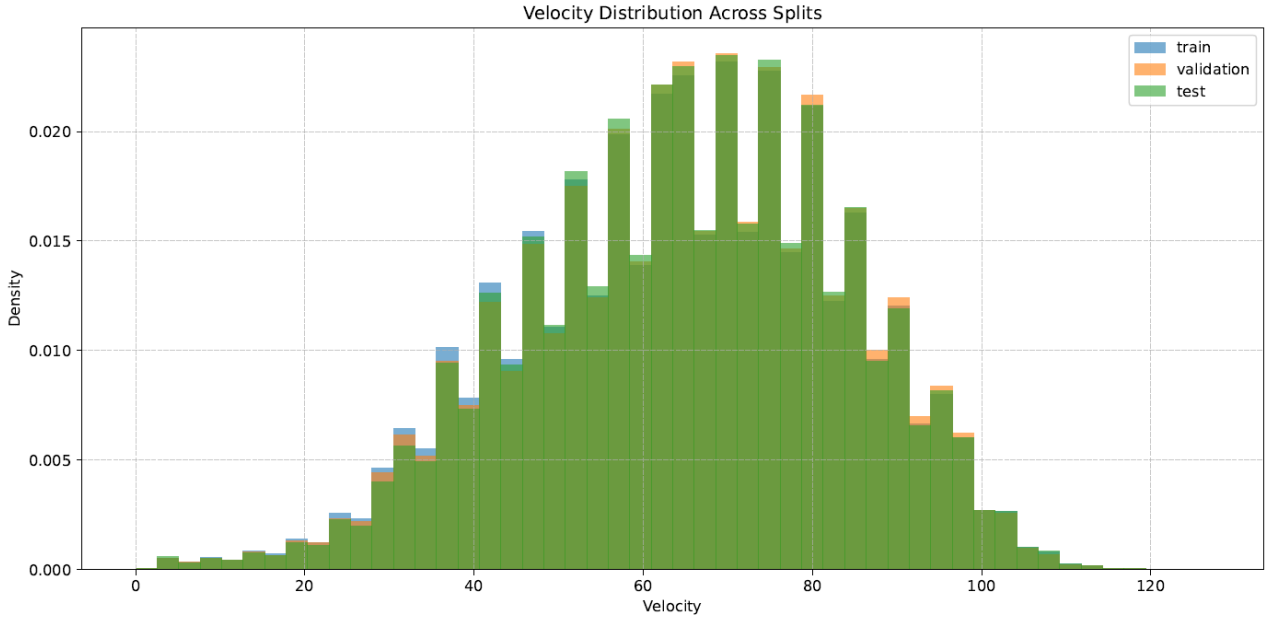

Figure 3.8 Velocity Distribution across splits

Composer	Title (Short)	Split	Yr	MIDI File (Truncated)	Audio File (Truncated)	Dur (s)
Alban Berg	Sonata Op. 1	train	2018	...Chamber3...wav--1.midi	...Chamber3...wav--1.wav	698.66
Alban Berg	Sonata Op. 1	train	2008	...R2_2008...wav--2.midi	...R2_2008...wav--2.wav	759.52
Alban Berg	Sonata Op. 1	train	2017	...Piano-e_3-02...wav--3.midi	...Piano-e_3-02...wav--3.wav	464.65
A. Scriabin	24 Preludes Op.11 No.13–24	train	2004	...XP_21_R1_2004...Track01_wav.midi	...XP_21_R1_2004...Track01_wav.wav	872.64
A. Scriabin	3 Etudes Op. 65	valid	2006	...17_R1_2006...Track04_wav.midi	...17_R1_2006...Track04_wav.wav	397.86
A. Scriabin	5 Preludes Op. 15	valid	2009	...07_R1_2009_04_WAV.midi	...07_R1_2009_04_WAV.wav	400.56
A. Scriabin	Entrag ete Op. 63	test	2009	...11_R1_2009_07_WAV.midi	...11_R1_2009_07_WAV.wav	163.75
A. Scriabin	Etudes Op. 2 & Op. 8 Nos. 5,11,12	train	2013	...19_R2_2013_wav--3.midi	...19_R2_2013_wav--3.wav	563.90
A. Scriabin	Etudes Op.42 No.4& 5	test	2009	...02_R1_2009_04_WAV.midi	...02_R1_2009_04_WAV.wav	136.32
A. Scriabin	Etude Op.8 No.13	valid	2009	...02_R1_2009_05_WAV.midi	...02_R1_2009_05_WAV.wav	167.09
A. Scriabin	Etude Op.8 No.10	train	2011	...R1-D6_09_Track09_wav.midi	...R1-D6_09_Track09_wav.wav	102.01

Table 3.1 MAESTRO v2 Dataset Metadata

3.3 Feature Extraction

The feature extraction pipeline in the music transcription system processes mel-spectrograms through the CR-GCN model's ConvStack, BiLSTM, and GCN modules to derive high-level features for note prediction. The ConvStack uses two 3x3 convolutional layers with batch normalization, ReLU, and 0.25 dropout to extract spatial and temporal patterns, followed by a fully connected layer. The BiLSTM captures bidirectional temporal dependencies for onset and offset predictions, while the GCN refines frame predictions by applying a precomputed adjacency matrix modeling piano key relationships. This matrix, loaded from p_co.csv, defines co-occurrence probabilities between keys. Together, these components ensure robust feature extraction for accurate onset, offset, frame, and velocity predictions in polyphonic piano audio.

```
class CR_GCN_Model(nn.Module):
    def __init__(self, input_features, output_features, model_complexity=48):
        super().__init__()
        model_size = model_complexity * 16
        sequence_model = lambda input_size, output_size: BiLSTM(input_size, output_size // 2)

        self.onset_stack = nn.Sequential(
            ConvStack(input_features, model_size),
            sequence_model(model_size, model_size),
            nn.Linear(model_size, output_features)
        )

        self.offset_stack = nn.Sequential(
            ConvStack(input_features, model_size),
            sequence_model(model_size, model_size),
            nn.Linear(model_size, output_features)
        )

        self.frame_stack = nn.Sequential(
            ConvStack(input_features, model_size),
            nn.Linear(model_size, output_features)
        )

        # Combined stack to handle dimensions correctly
        self.combined_stack = nn.Sequential(
            sequence_model(output_features * 3, output_features * 3),  # Keep same dimension
            nn.Linear(output_features * 3, output_features)
        )

        # GCN as separate module
        self.gcn = GCN(output_features, model_size)

        self.velocity_stack = nn.Sequential(
            ConvStack(input_features, model_size),
            nn.Linear(model_size, output_features)
```

```
def forward(self, mel):
    onset_pred = self.onset_stack(mel)
    offset_pred = self.offset_stack(mel)
    activation_pred = self.frame_stack(mel)

    # Concatenate predictions
    combined_pred = torch.cat([onset_pred.detach(), offset_pred.detach(), activation_pred], dim=-1)

    # Process through sequence model and linear layer
    frame_pred = self.combined_stack(combined_pred)

    # Apply GCN separately
    frame_pred = self.gcn(frame_pred)

    velocity_pred = self.velocity_stack(mel)
    return onset_pred, offset_pred, activation_pred, frame_pred, velocity_pred

def run_on_batch(self, batch):
    mel = melspectrogram(batch['audio'].reshape(-1, batch['audio'].shape[-1])[:, :-1]).transpose(-1, -2)
    onset_pred, offset_pred, activation_pred, frame_pred, velocity_pred = self(mel)
    predictions = {
        'onset': onset_pred.reshape(*batch['onset'].shape),
        'offset': offset_pred.reshape(*batch['offset'].shape),
        'frame': frame_pred.reshape(*batch['frame'].shape),
        'velocity': velocity_pred.reshape(*batch['velocity'].shape)
    }
    losses = {
        'loss/onset': F.binary_cross_entropy_with_logits(predictions['onset'], batch['onset']),
        'loss/offset': F.binary_cross_entropy_with_logits(predictions['offset'], batch['offset']),
        'loss/frame': F.binary_cross_entropy_with_logits(predictions['frame'], batch['frame']),
        'loss/velocity': F.mse_loss(predictions['velocity'], batch['velocity'])
    }
    return predictions, losses
```

Figure 3.9 Overall CR-GCN Model Class

The co-occurrence matrix M counts note pairs in each frame:

$$Mij = \sum_{Frames} \mathrm{I}(note\ i\ in\ frame).\ I(note\ j\ in\ frame)$$

where I (note i in frame) = 1 if note i is present, else 0, and M is symmetric (Mij = Mji).

The matrix is normalized to a conditional probability matrix P:

$$P_{ij} = \frac{M_{ij}}{\sum_{m=1}^{c} M_{im}},$$

where $c = 88$ is the number of notes, and $\sum_{m=1}^{c} M_{im}$ is the total occurrences of note i.

The adjacency matrix A is obtained by binarizing P with threshold τ:

$$A_{ij} = \begin{cases} 0, & \text{if } P_{ij} < \tau \\ 1, & \text{if } P_{ij} \geq \tau \end{cases},$$

where $\tau = 0.1$ balances node aggregation in the GCN.

```python
def create_adjacency_matrix():
    co_occurrence = np.zeros((N_KEYS, N_KEYS), dtype=np.float32)
    train_files = glob.glob(os.path.join(DATA_DIR, 'train', '*.tsv'))
    print(f"Found {len(train_files)} TSV files in train split")

    for tsv_path in tqdm(train_files, desc="Processing TSV files"):
        try:

            df = pd.read_csv(tsv_path, sep='\t')
            if not {'start_time', 'end_time', 'pitch'}.issubset(df.columns):
                print(f"Skipping {tsv_path}: Missing required columns")
                continue
            df = df[(df['pitch'] >= MIN_MIDI) & (df['pitch'] <= MIN_MIDI + N_KEYS - 1)]
            if df.empty:
                print(f"Skipping {tsv_path}: No valid pitches")
                continue

            df['pitch'] = df['pitch'] - MIN_MIDI

            events = []
            for _, row in df.iterrows():
                events.append((row['start_time'], 1, row['pitch']))  # Note on
                events.append((row['end_time'], -1, row['pitch']))  # Note off
            events.sort()
            active_notes = set()
            last_time = events[0][0]
            for time, event_type, pitch in events:
                if time - last_time > WINDOW_MS / 1000.0:
                    for note1 in active_notes:
                        for note2 in active_notes:
                            if note1 != note2:
                                co_occurrence[int(note1)][int(note2)] += 1
                    last_time = time
                if event_type == 1:
                    active_notes.add(pitch)
                if event_type == 1:
                    active_notes.add(pitch)
                else:
                    active_notes.discard(pitch)

        except Exception as e:
            print(f"Error processing {tsv_path}: {e}")
            continue

    # Normalize the matrix (symmetric)
    co_occurrence = (co_occurrence + co_occurrence.T) / 2
    row_sums = co_occurrence.sum(axis=1, keepdims=True)
    row_sums[row_sums == 0] = 1  # Avoid division by zero
    co_occurrence = co_occurrence / row_sums

    # Save the matrix
    np.savetxt(OUTPUT_PATH, co_occurrence, delimiter=',', fmt='%.6f')
    print(f"Adjacency matrix saved to {OUTPUT_PATH}")

if __name__ == "__main__":
    create_adjacency_matrix()
```

Figure 3.10 Generates an 88x88 co-occurrence matrix for the CR-GCN model using MAESTRO dataset's train split

Feature Learning Part:

In **Figure 3.11** shows the ConvStack class, a core feature extraction component in the music transcription system. It processes mel-spectrograms using two 3x3 convolutional layers with batch normalization, ReLU, and 0.25 dropout, extracting spatial and temporal features. These features are reshaped and passed through a fully connected layer, enabling the CR-GCN's model to predict note onsets, offsets, frames, and velocities.

```
class ConvStack(nn.Module):
    def __init__(self, input_features, output_features):
        super().__init__()
        self.cnn = nn.Sequential(
            nn.Conv2d(input_features, output_features, (3, 3), padding=1),
            nn.BatchNorm2d(output_features),
            nn.ReLU(),
            nn.Dropout(0.25),
            nn.Conv2d(output_features, output_features, (3, 3), padding=1),
            nn.BatchNorm2d(output_features),
            nn.ReLU(),
            nn.Dropout(0.25),
        )
        self.fc = nn.Sequential(
            nn.Linear(output_features, output_features),
            nn.Dropout(0.25)
        )

    def forward(self, mel):
        # mel: (batch, time, N_MELS) -> treat N_MELS as channels
        x = mel.unsqueeze(-1)  # (batch, time, N_MELS, 1)
        x = x.transpose(1, 2)  # (batch, N_MELS, time, 1)
        x = self.cnn(x)  # (batch, output_features, time, 1)
        x = x.squeeze(-1).transpose(1, 2)  # (batch, time, output_features)
        x = self.fc(x)
        return x
```

Figure 3.11 Convolutional Stack layer of CR-GCN

Figure 3.12 shows the BiLSTM class, a feature extraction component in the music transcription system. It processes convolutional features using a bidirectional LSTM to capture temporal dependencies, followed by a linear layer to maintain input dimensions. This module enhances onset and offset predictions in the CR-GCN's model by modeling sequential patterns in polyphonic piano audio.

```python
class BiLSTM(nn.Module):
    def __init__(self, input_size, hidden_size):
        super().__init__()
        self.lstm = nn.LSTM(input_size, hidden_size, batch_first=True, bidirectional=True)
        self.linear = nn.Linear(hidden_size * 2, input_size)

    def forward(self, x):
        x, _ = self.lstm(x)
        x = self.linear(x)
        return x
```

Figure 3.12 Bi-LSTM layer of CR-GCN Model

Label Learning Part:

Figure 3.13 shows the GCN class, a feature extraction component in the music transcription system. It refines frame predictions by applying an adjacency matrix to model piano key relationships, using three linear layers with ReLU and 0.25 dropout. This module enhances the CR-GCN model's ability to capture key co-occurrences in polyphonic piano audio.

```python
class GCN(nn.Module):
    def __init__(self, input_dim, hidden_dim):
        super().__init__()
        self.adj = None
        self.linear1 = nn.Linear(input_dim, hidden_dim)
        self.linear2 = nn.Linear(hidden_dim, hidden_dim)
        self.linear3 = nn.Linear(hidden_dim, input_dim)
        self.dropout = nn.Dropout(0.25)

    def set_adj(self, adj):
        self.adj = adj

    def forward(self, x):
        if self.adj is None:
            raise ValueError("Adjacency matrix not set. Call set_adj() first.")
        # x: (batch, time, input_dim)
        batch, time, _ = x.shape
        # Apply adjacency matrix first
        x = x.permute(0, 2, 1)  # (batch, input_dim, time)
        x = torch.matmul(self.adj, x)  # (batch, input_dim, time)
        x = x.permute(0, 2, 1)  # (batch, time, input_dim)
        # Reshape to (batch * time, input_dim) for linear layer
        x = x.reshape(batch * time, -1)
        x = F.relu(self.linear1(x))  # (batch * time, hidden_dim)
        x = self.dropout(x)
        # Apply second linear layer
        x = F.relu(self.linear2(x))  # (batch * time, hidden_dim)
        x = self.dropout(x)
        # Final linear layer
        x = self.linear3(x)  # (batch * time, input_dim)
        x = x.reshape(batch, time, -1)  # (batch, time, input_dim)
        return x
```

Figure 3.13 GCN Layer of CR-GCN Model

3.3 Training Part

Model Summary:

Layer Name	Type	Output Shape	# Parameters	Connected To
input_layer	InputLayer	(None, 320, 229)	0	-
onset_stack_cnn_0	Conv2D	(None, 768, 320, 1)	1,583,616	input_layer
onset_stack_cnn_1	BatchNorm2D	(None, 768, 320, 1)	1,536	onset_stack_cnn_0
onset_stack_cnn_2	ReLU	(None, 768, 320, 1)	0	onset_stack_cnn_1
onset_stack_cnn_3	Dropout	(None, 768, 320, 1)	0	onset_stack_cnn_2
onset_stack_cnn_4	Conv2D	(None, 768, 320, 1)	5,309,184	onset_stack_cnn_3
onset_stack_cnn_5	BatchNorm2D	(None, 768, 320, 1)	1,536	onset_stack_cnn_4
onset_stack_cnn_6	ReLU	(None, 768, 320, 1)	0	onset_stack_cnn_5
onset_stack_cnn_7	Dropout	(None, 768, 320, 1)	0	onset_stack_cnn_6
onset_stack_fc_0	Linear	(None, 320, 768)	590,592	onset_stack_cnn_7
onset_stack_fc_1	Dropout	(None, 320, 768)	0	onset_stack_fc_0
offset_stack_cnn_0	Conv2D	(None, 768, 320, 1)	1,583,616	onset_stack_fc_1
offset_stack_cnn_1	BatchNorm2D	(None, 768, 320, 1)	1,536	offset_stack_cnn_0
offset_stack_cnn_2	ReLU	(None, 768, 320, 1)	0	offset_stack_cnn_1
offset_stack_cnn_3	Dropout	(None, 768, 320, 1)	0	offset_stack_cnn_2
offset_stack_cnn_4	Conv2D	(None, 768, 320, 1)	5,309,184	offset_stack_cnn_3
offset_stack_cnn_5	BatchNorm2D	(None, 768, 320, 1)	1,536	offset_stack_cnn_4
offset_stack_cnn_6	ReLU	(None, 768, 320, 1)	0	offset_stack_cnn_5
offset_stack_cnn_7	Dropout	(None, 768, 320, 1)	0	offset_stack_cnn_6
offset_stack_fc_0	Linear	(None, 320, 768)	590,592	offset_stack_cnn_7
offset_stack_fc_1	Dropout	(None, 320, 768)	0	offset_stack_fc_0
frame_stack_cnn_0	Conv2D	(None, 768, 320, 1)	1,583,616	offset_stack_fc_1
frame_stack_cnn_1	BatchNorm2D	(None, 768, 320, 1)	1,536	frame_stack_cnn_0
frame_stack_cnn_2	ReLU	(None, 768, 320, 1)	0	frame_stack_cnn_1
frame_stack_cnn_3	Dropout	(None, 768, 320, 1)	0	frame_stack_cnn_2
frame_stack_cnn_4	Conv2D	(None, 768, 320, 1)	5,309,184	frame_stack_cnn_3
frame_stack_cnn_5	BatchNorm2D	(None, 768, 320, 1)	1,536	frame_stack_cnn_4
frame_stack_cnn_6	ReLU	(None, 768, 320, 1)	0	frame_stack_cnn_5

Layer Name	Type	Output Shape	# Parameters	Connected To
frame_stack_cnn_7	Dropout	(None, 768, 320, 1)	0	frame_stack_cnn_6
frame_stack_fc_0	Linear	(None, 320, 768)	590,592	frame_stack_cnn_7
frame_stack_fc_1	Dropout	(None, 320, 768)	0	frame_stack_fc_0
combined_stack_cnn_0	Conv2D	(None, 768, 320, 1)	1,583,616	frame_stack_fc_1
combined_stack_cnn_1	BatchNorm2D	(None, 768, 320, 1)	1,536	combined_stack_cnn_0
combined_stack_cnn_2	ReLU	(None, 768, 320, 1)	0	combined_stack_cnn_1
combined_stack_cnn_3	Dropout	(None, 768, 320, 1)	0	combined_stack_cnn_2
combined_stack_cnn_4	Conv2D	(None, 768, 320, 1)	5,309,184	combined_stack_cnn_3
combined_stack_cnn_5	BatchNorm2D	(None, 768, 320, 1)	1,536	combined_stack_cnn_4
combined_stack_cnn_6	ReLU	(None, 768, 320, 1)	0	combined_stack_cnn_5
combined_stack_cnn_7	Dropout	(None, 768, 320, 1)	0	combined_stack_cnn_6
combined_stack_fc_0	Linear	(None, 320, 768)	590,592	combined_stack_cnn_7
combined_stack_fc_1	Dropout	(None, 320, 768)	0	combined_stack_fc_0
onset_lstm_lstm	LSTM	(None, 320, 768)	3,545,088	combined_stack_fc_1
onset_lstm_linear	Dense	(None, 320, 768)	590,592	onset_lstm_lstm
offset_lstm_lstm	LSTM	(None, 320, 768)	3,545,088	onset_lstm_linear
offset_lstm_linear	Dense	(None, 320, 768)	590,592	offset_lstm_lstm
frame_gcn_matmul	MatMul	(None, 320, 88)	0	offset_lstm_linear
frame_gcn_linear1	Dense	(None, 320, 768)	68,352	frame_gcn_matmul
frame_gcn_linear2	Dense	(None, 320, 768)	590,592	frame_gcn_linear1
frame_gcn_linear3	Dense	(None, 320, 88)	67,672	frame_gcn_linear2
frame_gcn_dropout1	Dropout	(None, 320, 768)	0	frame_gcn_linear3
frame_gcn_dropout2	Dropout	(None, 320, 768)	0	frame_gcn_dropout1
onset_linear	Dense	(None, 320, 88)	67,672	frame_gcn_dropout2
offset_linear	Dense	(None, 320, 88)	67,672	onset_linear
frame_linear	Dense	(None, 320, 88)	67,672	offset_linear
combined_linear	Dense	(None, 320, 88)	67,672	frame_linear

Table 3.2 Summary of CR-GCM Model and No. of Parameters

Figure 3.14 shows the train_model function, which trains the CR-GCN model using MAESTRO dataset's train and validation splits with a batch size of 8. It employs an Adam optimizer with an exponentially decaying learning rate (initially 0.00035) and binary cross-entropy loss for onset, offset, and frame predictions. The model was trained for 100 epochs, incorporating callbacks for learning rate reduction, model checkpointing, and early stopping based on validation loss.

```python
def train_model():
    train_gen = PianoDataGenerator('train', batch_size=8)
    val_gen = PianoDataGenerator('validation', batch_size=8)

    model = CR_GCN_Model(input_features=N_MELS, output_features=NOTES)

    initial_learning_rate = 0.00035
    lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
        initial_learning_rate, decay_steps=10000, decay_rate=0.97, staircase=True
    )

    model.compile(
        optimizer=tf.keras.optimizers.Adam(learning_rate=lr_schedule),
        loss={
            'onset': 'binary_crossentropy',
            'offset': 'binary_crossentropy',
            'frame': 'binary_crossentropy'
        },
        loss_weights={'onset': 1.0, 'offset': 1.0, 'frame': 1.0}
    )

    z1 = callbacks.ReduceLROnPlateau(monitor='val_loss', patience=12, factor=0.75, min_lr=1e-6, verbose=1),
    z2 = callbacks.ModelCheckpoint("best_model.h5", save_best_only=True, monitor="val_loss", verbose=1),
    z3 = callbacks.EarlyStopping(monitor="val_loss", patience=15, restore_best_weights=True)


    history = model.fit(train_gen, validation_data=val_gen, epochs=50, callbacks=[z2 , z1 , z3])

    return model, history
```

Figure 3.14 Model Training Code

```
Epoch 1/100
120/120 ━━━━━━━━━━━━━━━━━━━━ 0s 6s/step - frame_loss: 0.3918 - loss: 0.8196 - offset_loss: 0.2128 - onset_loss: 0.2150
Epoch 1: val_loss improved from inf to 0.44653, saving model to best_model.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy
120/120 ━━━━━━━━━━━━━━━━━━━━ 771s 6s/step - frame_loss: 0.3907 - loss: 0.8177 - offset_loss: 0.2124 - onset_loss: 0.2147 - val_frame_loss: 0.1491 -
Epoch 2/100
120/120 ━━━━━━━━━━━━━━━━━━━━ 0s 6s/step - frame_loss: 0.1606 - loss: 0.4701 - offset_loss: 0.1551 - onset_loss: 0.1544
Epoch 2: val_loss improved from 0.44653 to 0.38229, saving model to best_model.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy
120/120 ━━━━━━━━━━━━━━━━━━━━ 717s 6s/step - frame_loss: 0.1606 - loss: 0.4700 - offset_loss: 0.1550 - onset_loss: 0.1544 - val_frame_loss: 0.1463 -
Epoch 3/100
120/120 ━━━━━━━━━━━━━━━━━━━━ 0s 6s/step - frame_loss: 0.1525 - loss: 0.3782 - offset_loss: 0.1123 - onset_loss: 0.1134
Epoch 3: val_loss did not improve from 0.38229
120/120 ━━━━━━━━━━━━━━━━━━━━ 693s 6s/step - frame_loss: 0.1525 - loss: 0.3781 - offset_loss: 0.1123 - onset_loss: 0.1133 - val_frame_loss: 0.1498 -
Epoch 4/100
120/120 ━━━━━━━━━━━━━━━━━━━━ 0s 6s/step - frame_loss: 0.1460 - loss: 0.3109 - offset_loss: 0.0827 - onset_loss: 0.0822
Epoch 4: val_loss improved from 0.38229 to 0.30411, saving model to best_model.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy
120/120 ━━━━━━━━━━━━━━━━━━━━ 724s 6s/step - frame_loss: 0.1459 - loss: 0.3109 - offset_loss: 0.0827 - onset_loss: 0.0822 - val_frame_loss: 0.1489 -
Epoch 5/100
120/120 ━━━━━━━━━━━━━━━━━━━━ 0s 6s/step - frame_loss: 0.1367 - loss: 0.2733 - offset_loss: 0.0689 - onset_loss: 0.0677
Epoch 5: val_loss improved from 0.30411 to 0.26538, saving model to best_model.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy
120/120 ━━━━━━━━━━━━━━━━━━━━ 699s 6s/step - frame_loss: 0.1367 - loss: 0.2732 - offset_loss: 0.0689 - onset_loss: 0.0677 - val_frame_loss: 0.1240 -
Epoch 6/100
120/120 ━━━━━━━━━━━━━━━━━━━━ 0s 6s/step - frame_loss: 0.1244 - loss: 0.2505 - offset_loss: 0.0629 - onset_loss: 0.0632
Epoch 6: val_loss improved from 0.26538 to 0.23948, saving model to best_model.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy
120/120 ━━━━━━━━━━━━━━━━━━━━ 726s 6s/step - frame_loss: 0.1243 - loss: 0.2504 - offset_loss: 0.0629 - onset_loss: 0.0632 - val_frame_loss: 0.1064 -
Epoch 7/100
120/120 ━━━━━━━━━━━━━━━━━━━━ 0s 6s/step - frame_loss: 0.1017 - loss: 0.2062 - offset_loss: 0.0524 - onset_loss: 0.0521
Epoch 7: val_loss improved from 0.23948 to 0.23106, saving model to best_model.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy
120/120 ━━━━━━━━━━━━━━━━━━━━ 712s 6s/step - frame_loss: 0.1017 - loss: 0.2062 - offset_loss: 0.0524 - onset_loss: 0.0521 - val_frame_loss: 0.0995 -
```

Figure 3.15 Training Epochs

4. DEPLOYMENT

4.1 Web-Application

The major idea behind the deployment component of our project was to develop a real time web application which is accessible by anyone with the URL. Users accessing the web application would be able to upload music files (of .wav format specifically) of their choice to our server. Our server would process the music file received from the user and utilise the deep learning model built to convert the music file into its corresponding midi files. These midi files are converted into an image of the corresponding music notes which are then displayed back to the user via the web application. Thus, the user would be able to view, take a snapshot or download the music notes displayed to them via the web application for further reference.

Design of the web application

The whole framework of the web application was designed using the django library of python. The front-end designs were implemented using HTML5/CSS and javascript. The back-end integration was also performed using Django library. For testing purposes, the web application was hosted using a private IP over a common network. For deployment purposes, the web application was hosted using a public IP provided by ngrok. **Figure 4.1** is a snapshot of the views.py file of the django framework specifying the back-end function behind file transfer from user to our server.

```
from django.shortcuts import render,redirect
from django.utils import timezone
from django.http import HttpResponse, HttpResponseRedirect
from django.conf import settings
import os

def home(req):
    return render(req,'home.html')

def store(request):
    if request.method == 'POST':

        if 'myfile' not in request.FILES or not request.FILES['myfile']:
            return redirect('/')

        file = request.FILES['myfile']
        filename = file.name
        filepath = os.path.join(settings.MEDIA_ROOT, filename)
        with open(filepath, 'wb') as f:
            for chunk in file.chunks():
                f.write(chunk)
        return render(request, 'display.html')
    else:
        return render(request, 'home.html')
```

Figure 4.1 views.py File

File-Handling

As mentioned earlier, the users shall upload their .wav audio files to our server in order to obtain the corresponding music notes of that audio file. To facilitate this in the back-end component, the function displayed in the views.py file(as shown in the previous image) is utilized to store the received file to our server for further processing. In order to provide a user interface for uploading multiple media files to our web application, the concept of multipart forms were used. The snippet mentioning the code in the html file for implementing multipart forms is displayed in **Figure 4.2 and Figure 4.3**. For file transfer from client to server, basic HTTP protocol was used.

```
<div class="content">
    <h2>Upload Your Files To Our Server</h2>
    <form method="post" action="upload/" enctype="multipart/form-data">
        {% csrf_token %}
        <input type="file" id="myfile" name="myfile" multiple hidden />
        <label for="myfile" class="custom-file-label">SELECT YOUR UPLOAD</label><br>
        <span id="file-name-display"></span>
        <br /><br />
        <input type="submit" value="Upload File" />
    </form>
</div>
```

Figure 4.2 Multipart Forms (HTML)

```
function updateFileNameDisplay() {
    const fileInput = document.getElementById('myfile');
    const fileNameDisplay = document.getElementById('file-name-display');

    // Add an event listener to detect when a file is selected
    fileInput.addEventListener('change', function(event) {
        const files = event.target.files;

        if (files.length > 1) {
            // If multiple files are selected, show the number of files
            fileNameDisplay.textContent = `${files.length} files selected`;
        } else if (files.length === 1) {
            // Display the name of the single selected file
            fileNameDisplay.textContent = files[0].name;
        } else {
            // If no file is selected, show placeholder text
            fileNameDisplay.textContent = 'No file selected';
        }
    });

    // Initialize the display to "No file selected" when the page loads
    fileNameDisplay.textContent = 'No file selected';
}
```

Figure 4.3 Multipart Forms (Javascript)

Scope for the Future

Although the designs have been implemented and the web application is responsive, the final CR-GCN model developed has to be integrated with the django framework in order to process the audio files uploaded and return the corresponding music sheets. For now, a unidirectional file transfer from user to our server and storing the received file in our server has only been implemented. Hence a bidirectional exchange of data between client and server needs to be implemented too. **Figure 4.4** is a snapshot of the final design of the web application developed so far. **Figure 4.5** enumerates the logs shown in the terminal while users are accessing the web application across networks.

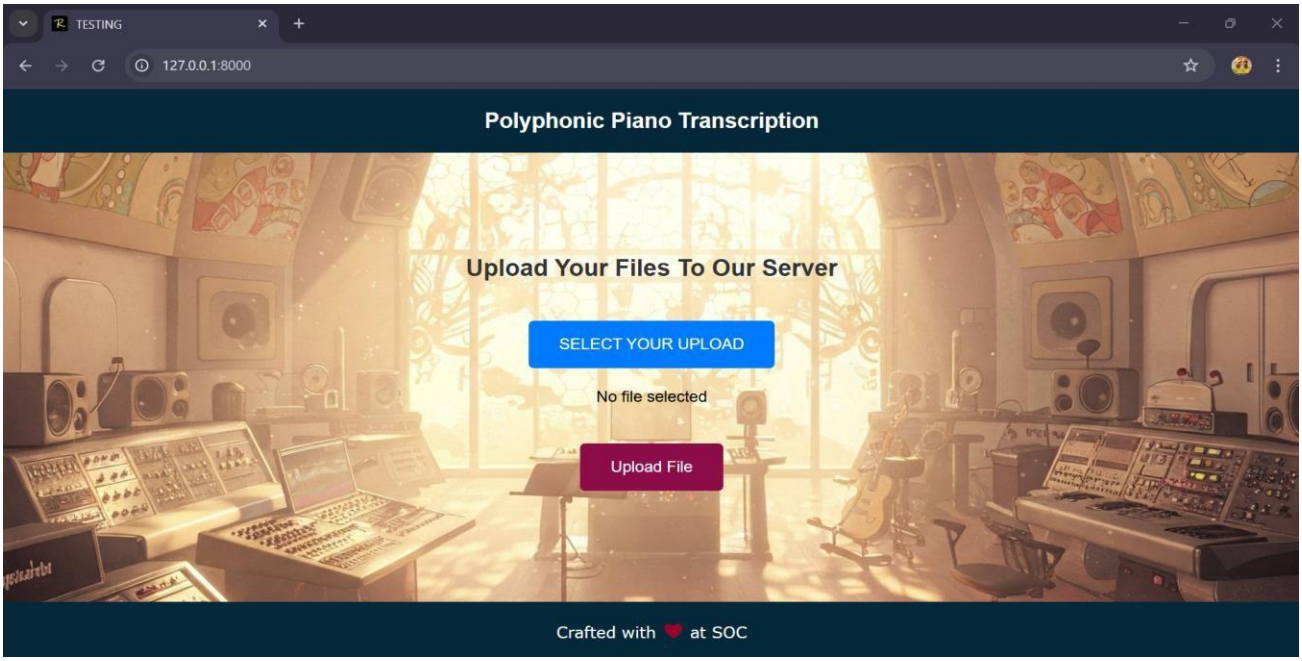

Figure 4.4 UI of Web Application

```
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
May 04, 2025 - 16:48:28
Django version 5.1.1, using settings 'mini.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.

[04/May/2025 16:48:34] "GET / HTTP/1.1" 200 2709
[04/May/2025 16:48:34] "GET /static/css/page1.css HTTP/1.1" 200 3781
[04/May/2025 16:48:35] "GET /static/images/gstudio.jpeg HTTP/1.1" 200 993700
[04/May/2025 16:48:36] "GET /static/icons/favicon.ico HTTP/1.1" 200 15406
[04/May/2025 16:48:49] "POST /upload/ HTTP/1.1" 200 1616
[04/May/2025 16:48:49] "GET /static/css/page2.css HTTP/1.1" 200 2350
[04/May/2025 16:49:04] "POST /upload/ HTTP/1.1" 200 1616
[04/May/2025 16:49:33] "GET / HTTP/1.1" 200 2709
[04/May/2025 16:49:42] "POST /upload/ HTTP/1.1" 200 1616
```

Figure 4.5 Logs of Terminal during Deployment

4.2 Music Sheet Generation

Prerequisites

1. **Music21 -** Python library for analyzing and manipulating musical elements.
2. **LilyPond** - music engraving tool that transforms text-based notation into professional-quality sheet music.
3. **Music21's elements -** Note, Chord, Stream, Tempo, Environment and Converter.

Sheet Generation Process

- MIDI or other audio formats can be processed by music21 identifying musical elements like pitch/notes and rhythm. You can analyze a MIDI file, extract notes, and group them into chords by detecting their onset times
- After analyzing the musical data, it can be converted into LilyPond notation. This conversion translates the data into a .ly file, which LilyPond can process and engrave as sheet music.
- After generating the .ly file, you compile it using LilyPond, which produces PNG file format of the music notation. This makes it possible to visualize the music created in music21.
- While processing an audio file, specifying the correct file path by ensuring that music21 can correctly read and analyze it. Extracting chords from the file involves checking which pitches share the same onset time and grouping them together.
- While making Environment Augmentation, specify the correct folder path for the LilyPond application lilypond.exe

```
midi_chords = []
for element in midi_stream.flatten().notes:
    if isinstance(element, chord.Chord):
        pitches = [p.midi for p in element.pitches]
        onset = round(float(element.offset),2)
        duration = round(float(element.quarterLength),2)

        # Debugging statements
        print(f'Chord Pitches: {pitches}, Onset: {onset}, Duration: {duration}')

        if duration > 0:  # Ensure you only add chords with positive duration
            midi_chords.append([pitches, onset, duration])

# Check how many chords were found
print(f'Total chords extracted: {len(midi_chords)}')

# Create a music21 score structure
score = stream.Score()
part = stream.Part()
part.insert(0, tempo.MetronomeMark(number=120))
score.metadata = metadata.Metadata(title="Music Sheets Score", composer="The USE
```

Figure 4.6 Chord Generation Code

```
Chord Pitches: [55, 48], Onset: 4.0, Duration: 4.0
Chord Pitches: [55, 47], Onset: 8.0, Duration: 4.0
Chord Pitches: [52, 55], Onset: 12.0, Duration: 2.0
Chord Pitches: [55, 48], Onset: 20.0, Duration: 4.0
Chord Pitches: [55, 47], Onset: 24.0, Duration: 4.0
Chord Pitches: [55, 52], Onset: 28.0, Duration: 2.0
Chord Pitches: [55, 53], Onset: 32.0, Duration: 1.5
Chord Pitches: [52, 55], Onset: 33.5, Duration: 0.5
Chord Pitches: [52, 55], Onset: 34.0, Duration: 2.0
Chord Pitches: [55, 47], Onset: 36.0, Duration: 2.0
Chord Pitches: [48, 55], Onset: 38.0, Duration: 2.0
Chord Pitches: [55, 47], Onset: 40.0, Duration: 2.0
Chord Pitches: [48, 55], Onset: 42.0, Duration: 2.0
Chord Pitches: [55, 47], Onset: 44.0, Duration: 2.0
Chord Pitches: [48, 55], Onset: 52.0, Duration: 4.0
Chord Pitches: [47, 55], Onset: 56.0, Duration: 4.0
Chord Pitches: [52, 55], Onset: 60.0, Duration: 2.0
Chord Pitches: [53, 55], Onset: 64.0, Duration: 1.5
```

```
Chord Pitches: [48, 50, 53], Onset: 29.0, Duration: 0.75
Chord Pitches: [48, 52, 55], Onset: 30.0, Duration: 2.75
Chord Pitches: [48, 53, 57], Onset: 33.0, Duration: 2.75
Chord Pitches: [48, 52, 55, 59], Onset: 36.0, Duration: 2.75
Chord Pitches: [60, 69], Onset: 39.0, Duration: 0.75
Chord Pitches: [60, 67], Onset: 40.0, Duration: 0.75
Chord Pitches: [60, 65, 67], Onset: 41.0, Duration: 0.75
Chord Pitches: [60, 64], Onset: 43.0, Duration: 1.75
...
Chord Pitches: [52, 60], Onset: 816.0, Duration: 0.25
Chord Pitches: [60, 72], Onset: 816.0, Duration: 0.75
Total chords extracted: 499
Total chords added to part: 499
```

Figure 4.7 Chord Generation

```
us = environment.UserSettings()
us['lilypondPath'] = 'E:\\lilypond-2.24.4\\bin\\lilypond.exe'

# Load the MIDI file
midi_file_path = 'E:\\lilypond-2.24.4\\Sheets\\example1.mid'
midi_stream = converter.parse(midi_file_path)
for onset, ch in chords_to_insert:
    part.insert(onset, ch)

# Verify the number of chords added to the part
print(f'Total chords added to part: {len(part.notes)}')

# Append part to score if there are any notes
if len(part.notes) > 0:
    score.append(part)

# Save and show sheet music
score.write('lilypond') # Save as PNG
score.show()
```

Figure 4.8 LilyPond Environment Augmentation Code

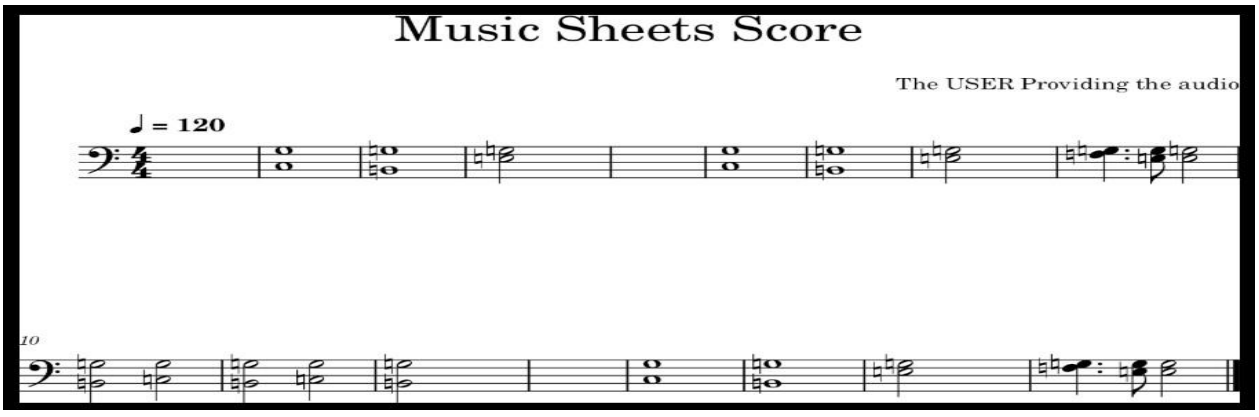

Figure 4.9 Music Sheet Generation

5. RESULTS AND DISCUSSIONS

5.1 Evaluation Results:

Metric Category	Precision	Recall	F1-score
Frame-level Metrics	0.9118	0.8124	**0.8588**
Note-level Metrics (Onset)	0.9009	0.8183	**0.8508**
Note w/ Offset-level Metrics	0.9011	0.8186	**0.8510**

Table 5.1 Evaluation Metrics of CR-GCN

5.2 Confusion Matrix for Frame Level Prediction

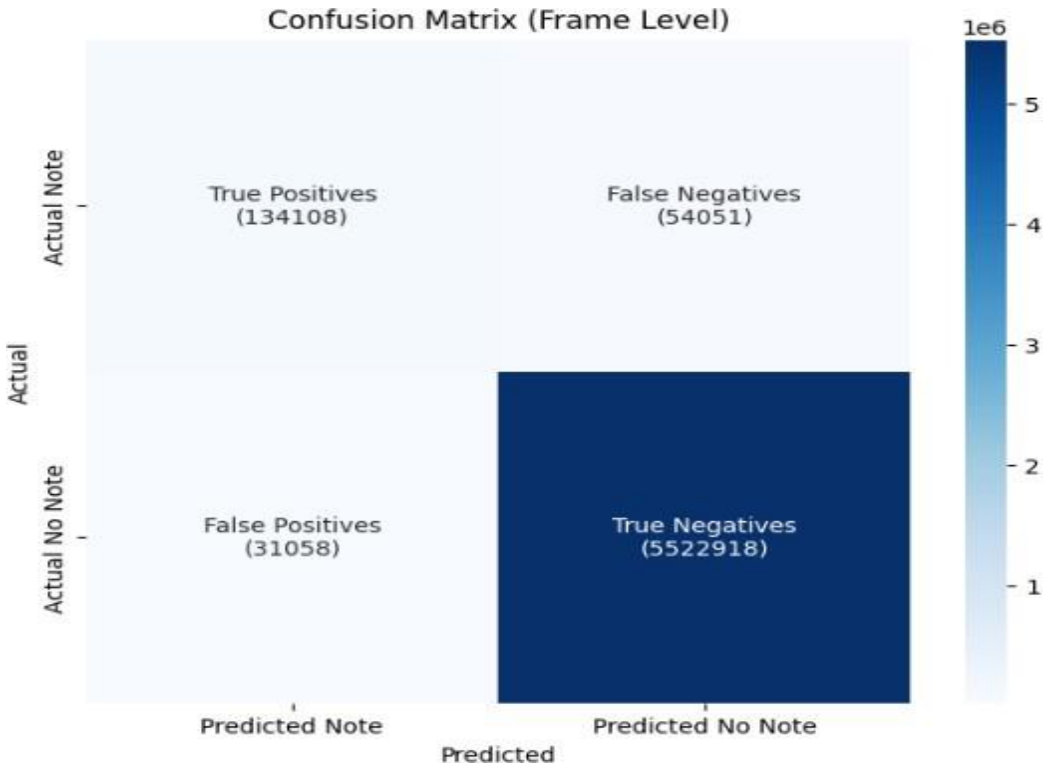

Figure 5.1 Confusion Matrix for Frame level Predictions

5.3 Training vs Validation Loss

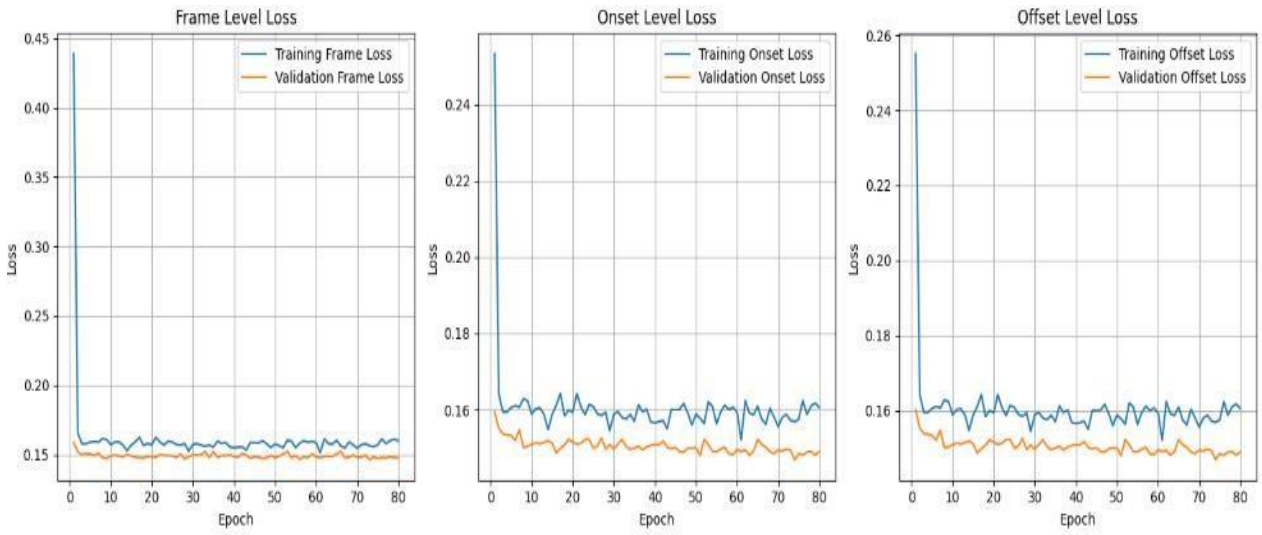

Figure 5.2 Training and Validation loss vs Epochs across three different losses

The enhanced CR-GCN model was evaluated on the MAESTRO dataset, demonstrating solid performance in polyphonic piano transcription. At the frame level, the model achieved a precision of 0.9118, a recall of 0.8124, and an F1-score of 0.8588, compared to the original CR-GCN's frame-level F1-score of 0.9277 on the same dataset. For note-level metrics, the model recorded an F1-score of 0.8508 for onset-only and 0.8510 for note with offset, against the base paper's 0.9588 and 0.8318, respectively. The frame-level confusion matrix showed 1,341,108 true positives and 5,522,918 true negatives, with 31,058 false positives and 540,051 false negatives, indicating effective detection but challenges with missed notes. The training and validation loss curves for frame, onset, and offset levels declined steadily, stabilizing after around 20 epochs with minimal divergence, suggesting robust convergence and limited overfitting.

While the proposed model showed competitive performance, it underperformed compared to the original CR-GCN on the MAESTRO dataset, particularly in frame-level (F1: 0.8588 vs. 0.9277) and note-level onset (F1: 0.8508 vs. 0.9588) metrics, possibly due to differences in dataset preprocessing or model optimization strategies. However, the note with offset F1-score (0.8510 vs. 0.8318) indicates a slight improvement in capturing note durations, likely benefiting from refined temporal modeling with the bidirectional LSTM, as noted in the base paper's emphasis on modeling note interdependencies. The high false negatives (540,051) highlight difficulties in detecting overlapping or quieter notes, a challenge also observed in the base paper's random chord experiments where CR-GCN struggled with generalization. The stable loss curves align with the base paper's findings on training stability, though the slight divergence after 100 epochs suggests potential overfitting risks, which could be mitigated with further regularization. Future work should aim to address these gaps to improve performance on the MAESTRO dataset. Enhancing the adjacency matrix construction with dynamic thresholding or incorporating attention mechanisms within the GCN, as an extension of the base paper's approach, could better capture note interdependencies and reduce false negatives. Additionally, applying advanced data augmentation techniques, such as synthetic noise injection, may improve detection of quieter notes, addressing the challenges noted in the base paper's high-polyphony experiments. Finally, optimizing the model's hyperparameters, such as the learning rate

or threshold τ, could further boost performance, ensuring the CR-GCN model remains competitive across diverse polyphonic transcription scenarios.

5.4 Comparative Analysis

Model	**Frame-Level F1 Score**	**Dataset**
Enhanced CR-GCN (Ours)	**0.8588**	**MAESTRO v2**
CR-GCN (Xiao et al., 2023 Base paper)	**0.9277**	**MAESTRO v2**
Onsets and Frames (Reimplemented)	0.7894	MAESTRO
Transition-Aware	0.8761	MAESTRO
Autoregressive Multi-State Note Model	0.8412	MAESTRO
High-Resolution Piano Transcription	0.9115	MAESTRO
Onsets and Frames (Original)	0.7800	MAPS
CNN-Transformer	0.8200	MAESTRO v2
Spectral-Temporal Model	0.7650	MAPS
Joint Onset-Offset Model	0.7950	MAPS
Acoustic Model with Data Augmentation	0.8800	MAESTRO

Table 5.2 Comparative Analysis on AMT Models

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. These findings contribute to the growing body of MIR research by offering insights into the practical challenges of AMT systems. 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.

7. REFERENCES

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

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] S. Block, M. Schedl, Polyphonic piano note transcription with recurrent neural networks, in: 2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE, 2012, pp. 121–124.

[7] E. Benetos, S. Dixon, D. Giannoulis, H. Kirchhoff, A. Klapuri, Automatic music transcription: challenges and future directions, J Intell Inf Syst 41 (3) (2013) 407–434.

[8] J.P. Bello, L. Daudet, M.B. Sandler, Automatic piano transcription using frequency and time-domain information, IEEE Trans Audio Speech Lang Process 14 (6) (2006) 2242–2251.

[9] S. Sigtia, E. Benetos, S. Cherla, T. Weyde, A. Garcez, S. Dixon, Rnn-based music language models for improving automatic music transcription, in: 15th International Society for Music Information Retrieval Conference, 2014.

[10] S. Sigtia, E. Benetos, S. Dixon, An end-to-end neural network for polyphonic piano music transcription, IEEE/ACM Trans Audio Speech Lang Process 24 (5) (2016) 927–939.

[11] C. Hawthorne, E. Elsen, J. Song, A. Roberts, Onsets and frames: Dual-objective piano transcription, in: Proceedings of the 19th International Society for Music Information Retrieval Conference, ISMIR 2018, Paris, France, 2018, 2018, pp. 50–57.

[12] Q. Wang, R. Zhou, Y. Yan, Polyphonic piano transcription with a note-based music language model, Applied Sciences 8 (3) (2018) 470.