

50.038 Computational Data Science

F04 Piano Transcription

December 2020

1 Abstract

A multi-model audio analysis tool, done periodically over a span of music wav files. Utilizing this model would produce a visual transcription that is able to show what specific notes are being played in the audio file.

2 Introduction

Automatic Music Transcription (AMT) is a process of converting an acoustic music into any form of musical notation, which allows the preservation of music performances and the composers' pieces over a long period of time. However, due to the complexity and the diversity of music nature and musical features, this process has been a challenging task, even for musicians with many years of formal training [1, 6, 10, 13]. Furthermore, in human music transcription, there exists some problems, such as, each musician may transcribe the same musical piece differently due to differences in musical perceptions, differences in viewing the importance of different aspects of a musical piece, and differences in transcription speed when given different musical genres to transcribe [10]. These introduce inconsistency in the human transcription process and other methods are required to retrieve the musical information and transcribe important musical features, such as pitch, tempo, loudness, and duration [6].

In recent years, machine learning models have proved to bring many benefits in music research, from music generation [8] to music genre classification [11]. Hence, many research has been conducted to apply machine learning models to the music transcription process [6, 16, 15], and it was identified that the two crucial stages in AMT is pitch detection and note tracking [14]. Pitch detection is to estimate the pitch that is played at each time frame, and note tracking is to be able to identify the onset and offset time of a pitch. Convolutional Neural

Network (CNN) is one of the most popular models that is used in AMT, and is widely used in the two stages [1, 16]. This paper explores the use of CNN model in monophonic music transcription, focusing mainly on the note tracking stage, and compare it with a state-of-the-art [5] model - CREPE (Convolutional Representation for Pitch Estimation), which targets the pitch detection stage [9].

3 Motivation

It was noticed that in the music industry, talent and perfect pitch are what the musician is born with. Although it might be possible to train the ear to detect certain notes and melodies, it is mostly people with innate talent that are able to transcribe intricate songs into notes or sheet music with ease. This project aims to remove this limitation using machine learning model to help people with a strong passion for music to be able to play any song they desire even when the sheet music is not available to them.

4 Project Overview

This project utilizes the forms of machine learning algorithms, Convolutional Neural Network (CNN) and Long Short-Term Memory (LSTM), in an attempt to resolve the problems in AMT.

5 Dataset and Collection

MIDI Aligned Piano Sounds (MAPS) dataset is used for training and testing the two models [4]. The dataset consists of 29910 piano recordings in .wav file format, categorised into nine folders based on the different instruments used, and their recording conditions. Within each folder, the piano sound files are further categorised based on the audio characteristics into four sets – ISOL, RAND, UCHO and MUS set. The description of the four categories are shown in Table 1

Table 1: The description of the four set of piano sounds in MAPS dataset

Set	Description
ISOL	Isolated notes and monophonic sounds
RAND	Random Chords
UCHO	Usual Chords
MUS	Pieces of Music

The MAPS recordings are sampled with a sampling frequency of 44kHz, and each recording is paired with a .txt file, which records the onset and offset time for all the notes that are played with its corresponding midi number. The .txt file will be used as a ground truth to evaluate the performance of the models. In our initial model - CNN only model, only the ISOL file is used for training, whereas the entire dataset is used to train our final model - CNN + LSTM model. The dataset is preprocessed first before training, which will be explained in the later section.

6 Data Pre-processing

6.1 Librosa

The initial model involves using librosa [12]. Using Fast Fourier Transform (FFT), the audio signal input was converted from time domain to frequency domain, where the magnitude is the contribution of each frequency to the overall sound. Subsequently, the magnitude was mapped to relative frequency using linspace (function that gives a number of evenly spaced numbers in an interval. (freq interval to be considered is 0 to sr, no of evenly spaced values = len(mag))). The power spectrum obtained is shown in Figure 1.

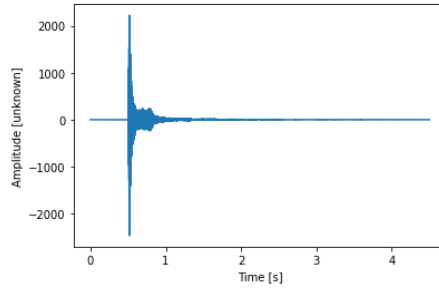


Figure 1: Librosa Power Spectrum

To understand how the frequency contributed to the overall sound throughout time, the short time Fourier transform (STFT) is used with the following parameter definitions:

```
#FFT window size
n_fft = 2048
#number of frames of audio between STFT column
hop_length = 512

#STFT
stft = librosa.core.stft(signal, hop_length=hop_length, n_fft=n_fft)
```

The output of the STFT can be visualized through a spectrogram, a heatmap with frequency on the y-axis, time on the x-axis and colour of the heatmap as the amplitude of the frequency, as shown in Figure 2.

```
spectrogram = np.abs(stft)
```

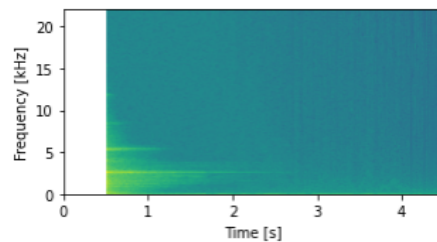


Figure 2: Spectrogram HeatMap

To linearize the STFT, logarithm is applied to the spectrogram and the output is shown in Figure 3.

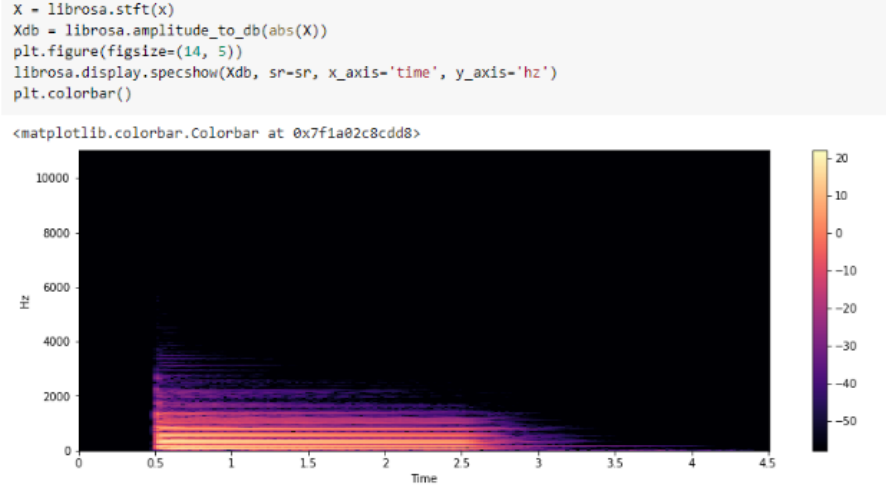


Figure 3: Logarithm Spectrogram

Another method explored to classify the training .wav files to their respective label notes is chroma feature. While the pitch can be obtained easily using chroma feature, information about the octave is not captured. Figure 4 shows some sample outputs from librosa's chroma feature.

With reference to Figure 5, M65 has a predicted note of F, and M63 has a predicted note of D, which are both correct. However, the tool is less accurate for notes towards the ends of the spectrum. Hence, a model is required to obtain higher accuracies.

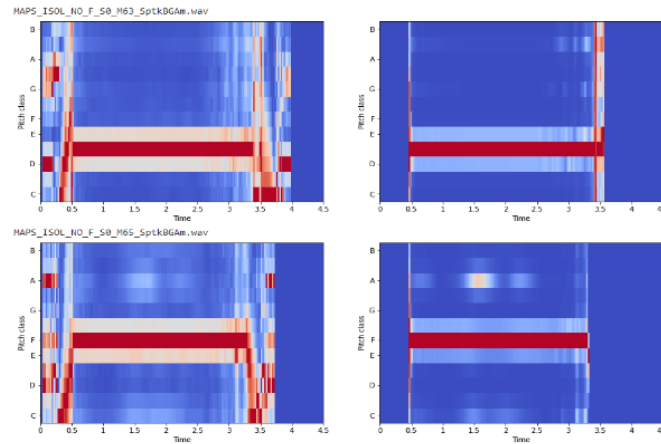


Figure 4: Librosa's Chroma Feature





































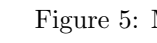




MIDI number		Note name	Keyboard	Frequency Hz		Period ms	
21	22	A0		27.500		36.36	
23		B0		30.868	29.135	32.40	34.32
24	25	C1		32.703		30.58	
26	27	D1		36.708	34.648	27.24	28.86
28		E1		41.203	38.891	24.27	25.71
29	30	F1		43.654		22.91	
31	32	G1		48.999	46.249	20.41	21.62
33	34	A1		55.000	51.913	18.18	19.26
35		B1		61.735	58.270	16.20	17.16
36	37	C2		65.406		15.29	
38	39	D2		73.416	69.296	13.62	14.29
40		E2		82.407	77.782	12.13	12.86
41	42	F2		87.307		11.45	
43	44	G2		97.999	92.499	10.20	10.81
45	46	A2		110.00	103.83	9.091	9.631
47		B2		123.47	116.54	8.099	8.581
48	49	C3		130.81		7.645	
50	51	D3		146.83	138.59	6.811	7.216
52		E3		164.81	155.56	6.068	6.428
53	54	F3		174.61		5.727	
55	56	G3		196.00	185.00	5.102	5.405
57	58	A3		220.00	207.65	4.545	4.816
59		B3		246.94	233.08	4.050	4.290
60	61	C4		261.63		3.822	
62	63	D4		293.67	277.18	3.405	3.608
64		E4		329.63	311.13	3.034	3.214
65	66	F4		349.23		2.863	
67	68	G4		392.00	369.99	2.551	2.703
69	70	A4		440.00	415.30	2.273	2.408
71		B4		493.88	466.16	2.025	2.145
72	73	C5		523.25		1.910	
74	75	D5		587.33	554.37	1.703	1.804
76		E5		659.26	622.25	1.517	1.607
77	78	F5		698.46		1.432	
79	80	G5		783.99	739.99	1.276	1.351
81	82	A5		880.00	830.61	1.136	1.204
83		B5		987.77	932.33	1.012	1.073
84	85	C6		1046.5		0.9556	
86	87	D6		1174.7	1108.7	0.8513	0.9020
88		E6		1318.5	1244.5	0.7584	0.8034
89	90	F6		1396.9		0.7159	
91	92	G6		1568.0	1480.0	0.6378	0.6757
93	94	A6		1760.0	1661.2	0.5682	0.6020
95		B6		1975.5	1864.7	0.5062	0.5363
96	97	C7		2093.0		0.4778	
98	99	D7		2349.3	2217.5	0.4257	0.4510
100		E7		2637.0	2489.0	0.3792	0.4018
101	102	F7		2793.0		0.3580	
103	104	G7		3136.0	2960.0	0.3189	0.3378
105	106	A7		3520.0	3322.4	0.2841	0.3010
107		B7		3951.1	3729.3	0.2531	0.2681
108		C8		4186.0		0.2389	

Figure 5: MIDI Sequence

6.2 nnAudio

To achieve better accuracies, preprocessing and normalisation on the spectrogram were done to provide a better input representation for the CNN model [2]. The Mel spectrogram was chosen as the preprocessing step because it is proven to provide a high accuracy using a compact spectrogram representation, thus reducing the computational speed [2].

nnAudio [3], an audio processing toolbox which provides on-the-fly audio to spectrogram conversion using the GPU, and consists of the Mel spectrogram function, is used in our model as it is able to generate the spectrogram representations around 100 times faster than traditional libraries [2].

The setup for the Mel spectrogram is as follows:

```
self.spec_layer = Spectrogram.MelSpectrogram(  
    sr=44100,  
    n_mels=156,  
    hop_length=441,  
    pad_mode='constant',  
    fmin=20, fmax=4200,  
    trainable_mel=False,  
    trainable_STFT=False)  
    .to(device)
```

The sampling rate was kept at 44100 Hz in accordance to the MAPS dataset. A hop_length of 441 meant that a timestep of 10ms can easily be obtained. fmin and fmax relates to the lowest and highest frequency of the keys of a piano respectively.

7 CNN architecture

The preliminary model was first trained using a CNN architecture. The Mel-Spectrogram layer is first obtained from the library nnAudio. The output from the Mel-Spectrogram layer is passed through a log function for the purpose of making training more efficient. After which, 2D Convolution is applied across both frequency and time domains with an accompanying ReLU Activation function. The output is subsequently passed through a Linear layer. The output of this fully connected layer is finally passed through a sigmoid activation function for classification.

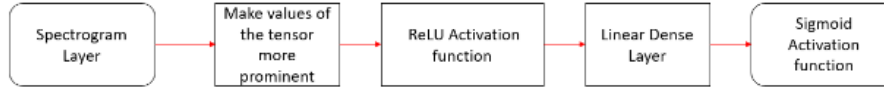


Figure 6: CNN Layer

8 CNN-LSTM architecture

Building on the CNN only architecture, the convolution across time domain is replaced with LSTM. In training the model, a batch size of 1024 is passed through the spectrogram as input through a 2D Convolution layer across frequency domain which does feature extraction and downsampling of the Mel spectrogram. After reshaping the output of the CNN blocks of size 3008, it is passed through a LSTM layer with hidden size of 1024, and this output is passed through a dense layer with an output size of 88. The final output is obtained by passing through a sigmoid activation function.

```

class Model(torch.nn.Module):
    def __init__(self):
        super(Model, self).__init__()
        self.spec_layer = Spectrogram.MelSpectrogram(sr=44100, n_mels=156, hop_length=441,
                                                    window='hann', center=True, pad_mode='constant',
                                                    fmin=20, fmax=4200, norm=1,
                                                    trainable_mel=False, trainable_STFT=False).to(device)

        k_out = 64
        self.CNN_freq_kernel_size = (64,1)
        self.CNN_freq_kernel_stride = (2,1)
        self.CNN_freq = nn.Conv2d(1, k_out,
                                   kernel_size=self.CNN_freq_kernel_size, stride=self.CNN_freq_kernel_stride)
        self.lstm = torch.nn.LSTM(input_size=3008, hidden_size=1024, bias=False, dropout=0.2, batch_first=True)
        self.linear = torch.nn.Linear(1024, 88, bias=False)

    def forward(self, x):
        z = self.spec_layer(x)[:,:,:-1]
        z2 = torch.log(z+1e-30)
        z3 = torch.relu(self.CNN_freq(z2.unsqueeze(1)))
        z3 = z3.contiguous().view(z3.shape[0], z3.shape[1]*z3.shape[2], z3.shape[3])
        z3 = torch.transpose(z3, 1, 2)
        z4, states = self.lstm(z3)
        y = self.linear(z4)
        return torch.sigmoid(y)
  
```

Figure 7: CNN-LSTM Layer

9 Model Adjustments

To further improve the model accuracy, some ablation studies were conducted for the model.

9.1 Adjusting Batch Size and Window Size

Comparing Figure 8 and Figure 9, while larger batch sizes are known to have greater generalization errors, the increase in window size has a greater effect on the accuracy of the output. The model with a longer window size has longer duration on the notes as well as more defined probability values, with red as the highest probability.

However, the longer the window sizes, the slower the training speed as more data will be trained on. With the understanding that larger batch sizes also have faster computational speeds, the final batch size and window size used for training the final model are shown in Table 2 along with the other hyperparameters of the final model.

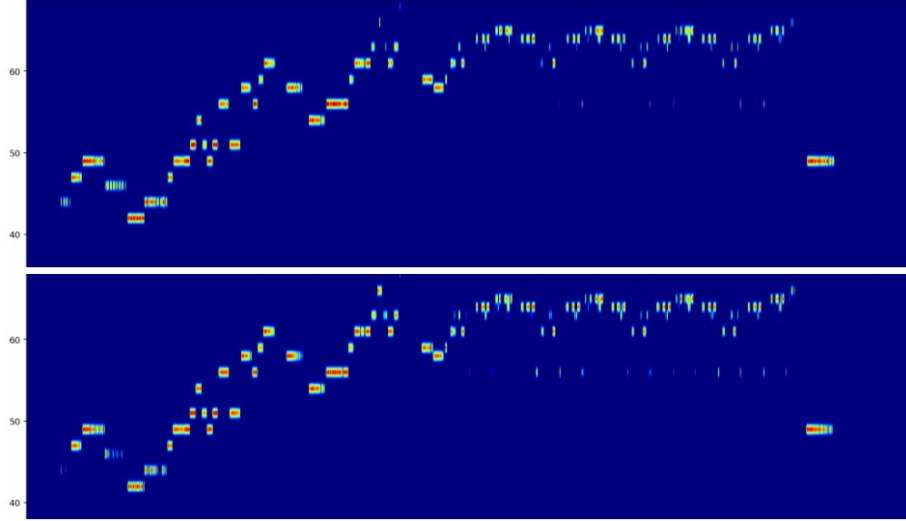


Figure 8: Visualization of output from model trained with batch size 32, window size 5 after 12 epochs, evaluated with window size 10 (top) and window size 5 (bottom)

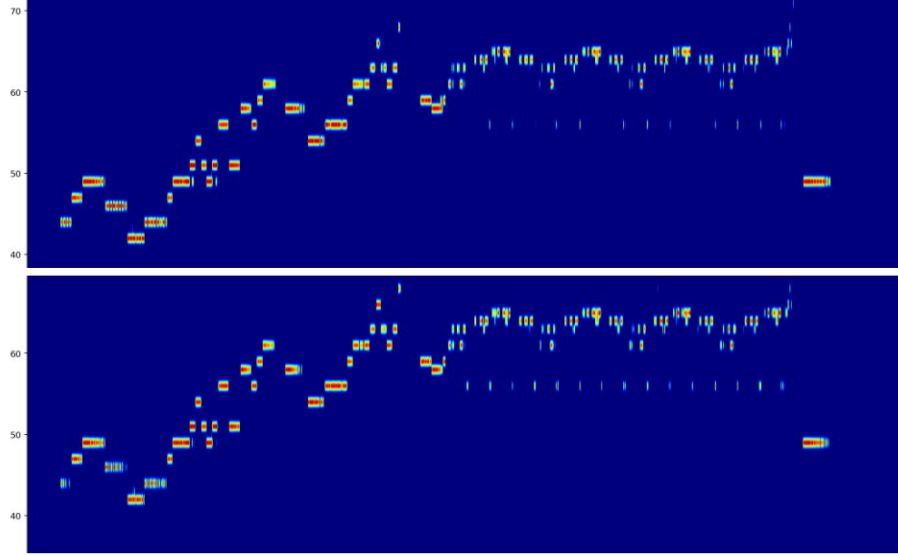


Figure 9: Visualization of output from model trained with batch size 16, window size 10 after 12 epochs, evaluated with window size 10 (top) and window size 5 (bottom)

Table 2: Hyperparameters of Final Model

Layer	Hyperparameter	Value
-	Batch Size	1024
	Window Size	40
	Learning Rate	0.00001
	Epochs	100
Conv2D	Output Layers	64
	Kernel Size	(64, 1)
	Kernel Stride	(2, 1)
LSTM	Hidden Layers	1024
	Dropout	0.2

10 Output Postprocessing

10.1 Adjusting Window Size

By adjusting the window size of the test data, while the window size of the training data is fixed, it is also observed that this affects the output from the test data.

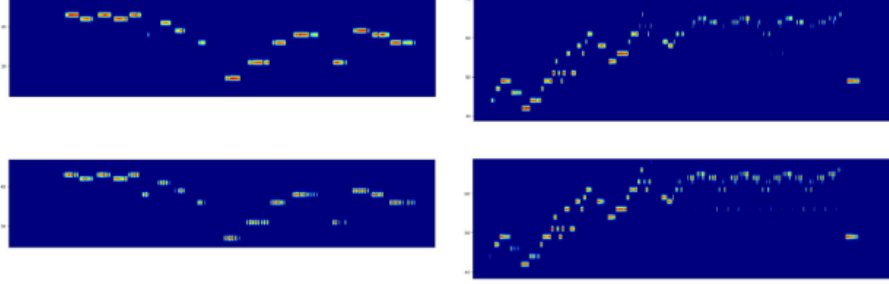


Figure 10: Visualization of MIDI notes with different window size. (Top Left) Fur Elise with window size 10, (Top Right) Lunatic Princess with window size 10, (Bottom Left) Fur Elise with window size 5, (Bottom Right) Lunatic Princess with window size 5

As seen from Figure 10 above, the results are generated from the exact same model, with the top row having a test window size of 10, bottom row having a test window size of 5, and with Fur Elise (left) and lunatic princess (right) as the inputs. It is observed that in the images of the top row, the notes are more joined together. However, overall, there is also more loss of notes, especially the shorter ones, in the top row.

10.2 Combining Window Sizes

With the above observation, an attempt to combine the outputs of different window sizes, where the maximum value at each timestep and Musical Instrument Digital Interface (MIDI) note across the windows sizes is taken, was made. The rationale behind is to get the best of both worlds where larger window sizes help to keep longer notes joined together and small window sizes help to detect the shorter notes.

However, this resulted in some unwanted joining of notes and additional noise, as seen from Figure 11.

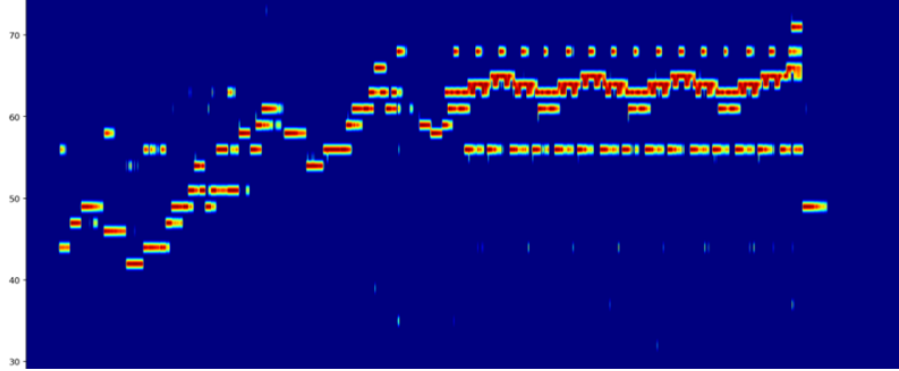


Figure 11: Visualization of output with test window sizes from 1 to 40 and activation at 0.5

Despite that, with the fact that a note that has a high probability at a certain timestep across different windows means that the model has a high confidence for that prediction, by increasing the activation, a final output that is better than the raw output of the model can be obtained, where the incorrectly joined notes are separated again, and the additional noise are removed, as seen from Figure 12.

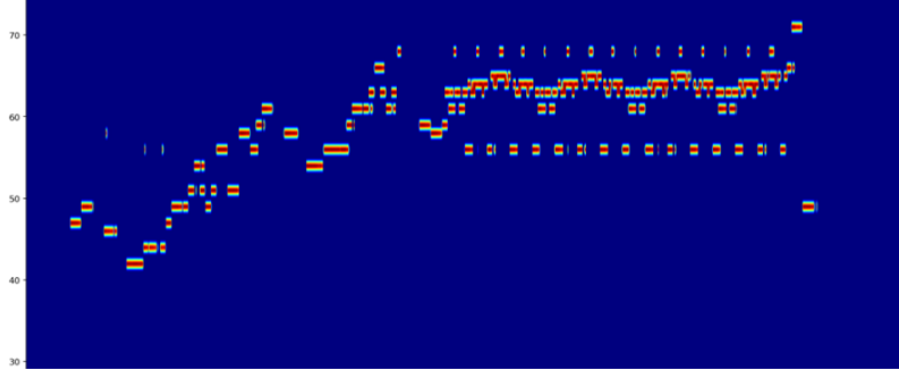


Figure 12: Visualization of output with test window sizes from 1 to 40 and activation at 0.9

11 Front-end User Interface

The model is pre-loaded using PyTorch .pt model. The web application can either be hosted locally or on an AWS EC2 XLarge 32GB Disk Storage In-

stance. The front-end code can be found at <https://github.com/tengfong/F04Musician>

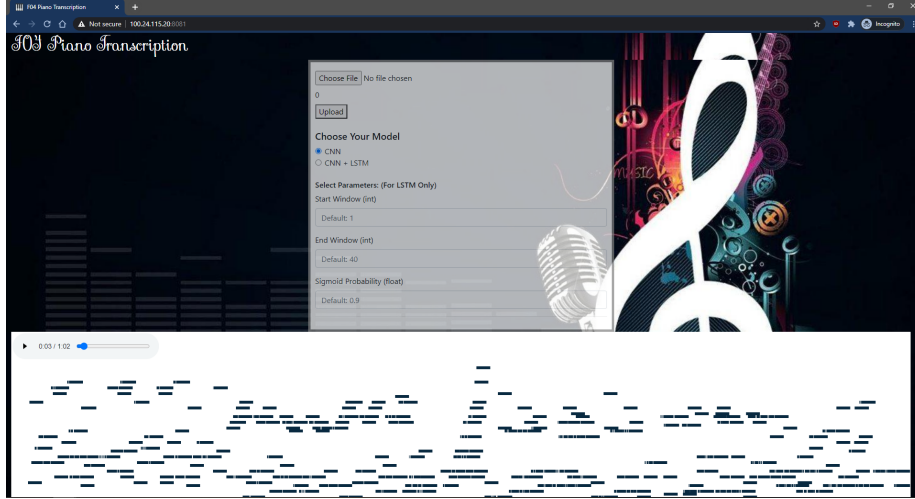


Figure 13: Front-end User Interface

12 Evaluation

12.1 CREPE Model for Evaluation

The two models are compared against the current state of the art in monophonic pitch estimation - Convolutional Representation for Pitch Estimation (CREPE) model to provide a rough estimation of the performance of the two models.

The CREPE pitch estimation model is able to generate predictions with time-domain audio signal as its inputs. For each prediction that is made on the user-selected timestep, the algorithm also outputs the confidence value, which measures the probability of detecting any audio signal during that time frame.

To make better comparison between our models and the CREPE model, the timestep selected for the CREPE model was also set to be 10ms for each prediction. Different confidence intervals were tested on the training dataset, as shown in Figure 12.1, and the confidence level of 0.85 was chosen after comparing it with audio sound with different characteristics, as shown in Table 3. Simple post processing process was done on the CREPE prediction output, the single stray notes were removed by setting its frequency to be the same as the frequency of the previous time frame.

Table 3: Different Playing Styles Recorded in ISOL File and Their Descriptions

Playing Style	Description
NO	2-seconds long notes played normally
ST	Staccato
RE	Repeated notes, faster and faster, from about 1.4 to 13.5 notes per second
CH	Chromatic ascending and descending scales
TR	Trills, faster and faster, up to half tone or to one tone, from about 2,8 to 32 notes per second

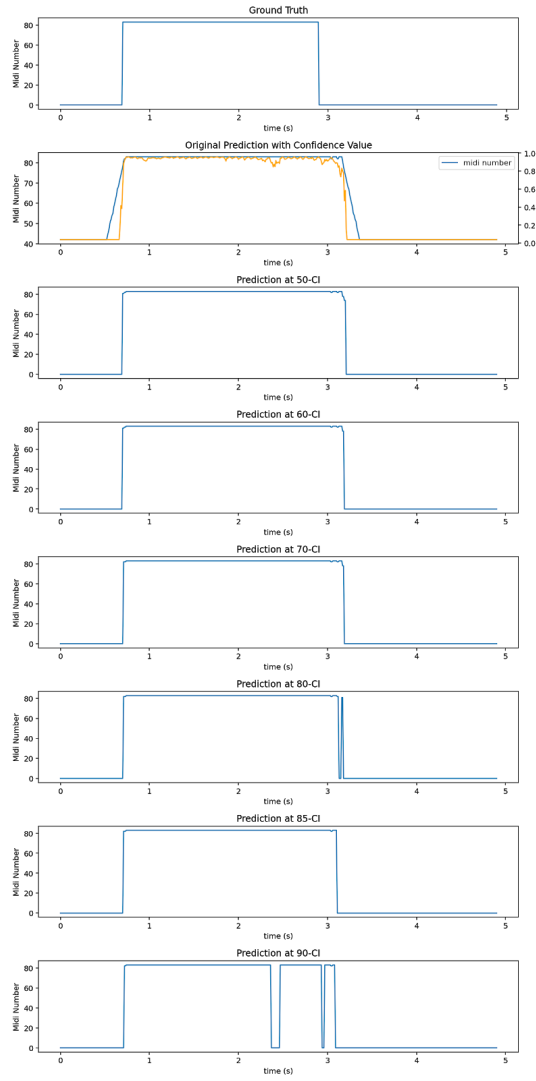


Figure 14: CREPE Model Prediction Result at Different Confidence Intervals

12.2 Evaluation Metric

As a huge portion of the MAPS recordings contains long waiting time between the start of the recording to the onset time of the first note appears, and between the offset time of the last note to the end of the recordings, weights were added to the accuracy score to highlight the correct prediction made on the non-zero midi notes (true positive, TP), and not neglecting the correct prediction made on the zero midi notes (true negative, TN). The weighted accuracy score was calculated using the formula below:

$$\text{Weighted Accuracy} = \frac{(\alpha \times TP) + (\beta \times TN)}{(\alpha \times (TP + FN)) + (\beta \times (TN + FP))}$$

where:

α = weight for non-zero midi number ground truth

β = weight for zero midi number ground truth

TP = True Positive

TN = True Negative

FP = False Positive

FN = False Negative

α score of 1 and β score of 0.5 were chosen after multiple trials with different α and β combinations. The same testing dataset was used to get the average weighted accuracy score of the three model, which the scores are shown in Table 4.

Table 4: Average Weighted Accuracy Score for Different Models by Different Playing Styles

Playing Style	Average Weighted Accuracy Score(%)		
	CNN	CNN + LSTM	CREPE
CH	16.23	37.77	33.80
NO	26.52	79.18	57.15
RE	31.96	69.06	46.95
ST	4.84	84.12	78.18
TR1	21.88	58.93	37.70
TR2	17.64	60.13	36.35
Average	19.845	53.68	48.36

Table 5: Comparison Table between Models Using Average Weighted Accuracy Score by Different Playing Styles

Playing Style	Comparing Average Weighted Accuracy Score between Models (10%)		
	CNN vs CREPE	CNN + LSTM vs CNN	CNN + LSTM vs CREPE
CH	-51.98	132.72	11.75
NO	-53.60	198.57	38.55
RE	-31.93	116.08	47.09
ST	-93.81	1638.02	7.60
TR1	-41.96	169.33	56.31
TR2	-51.47	240.87	65.41
Average	-58.96	226.86	34.14

13 Discussion

From Table 5, it is observed that the 2nd model - CNN + LSTM model has a significant improvement in performance when it is compared against its previous model - CNN model. It shows that by adding a LSTM layer, the training model is able to perform better as audio itself can be seen as a sequential data where learning front and back while retaining information from the previous layer helps with identifying the correct pitch and furthermore, LSTM takes into account on temporal dependence.

However, our two models are trained and tested with MAPS dataset, it introduces some biases when comparing the weighted accuracy score with the CREPE model, which is not trained using the MAPS dataset. Hence, the average weighted accuracy score when comparing against the CREPE model can only serve as a relative measurement of improvement between our 1st and 2nd model.

14 Future works and improvements

14.1 Polyphonic music transcription

Currently the model only works on monophonic music with some promising entities for polyphonic music transcription. This can be achieved by training it on MAESTRO data set, a data set that consist of polyphonic music data [7].

14.2 Transcription

The output for the prediction is currently a MIDI file and is presented as a piano roll. There can be an option to convert the MIDI file output into a proper piano sheet.

14.3 Improve Accuracy

It is possible to do ensemble model by training more models (3D Convolutional Layer) which can be then averaged out to obtain the best possible results.

References

- [1] M. Bereket. An ai approach to automatic natural music transcription. 2017.
- [2] K. W. Cheuk, K. Agres, and D. Herremans. The impact of audio input representations on neural network based music transcription. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–6, 2020.
- [3] K. W. Cheuk, K. Agres, and D. Herremans. The impact of audio input representations on neural network based music transcription. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–6, 2020.
- [4] V. Emiya, N. Bertin, B. David, and R. Badeau. Maps - a piano database for multipitch estimation and automatic transcription of music. page 11, 07 2010.
- [5] B. Gfeller, C. Frank, D. Roblek, M. Sharifi, M. Tagliasacchi, and M. Velimirovic. Spice: Self-supervised pitch estimation. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 28:1118–1128, 2020.
- [6] C. Hawthorne, A. Stasyuk, A. Roberts, I. Simon, C.-Z. A. Huang, S. Dieleman, E. Elsen, J. Engel, and D. Eck. Enabling factorized piano music modeling and generation with the MAESTRO dataset. In *International Conference on Learning Representations*, 2019.
- [7] C. Hawthorne, A. Stasyuk, A. Roberts, I. Simon, C.-Z. A. Huang, S. Dieleman, E. Elsen, J. Engel, and D. Eck. Enabling factorized piano music modeling and generation with the MAESTRO dataset. In *International Conference on Learning Representations*, 2019.
- [8] N. Hewahi, S. AlSaigal, and S. AlJanahi. Generation of music pieces using machine learning: long short-term memory neural networks approach. *Arab Journal of Basic and Applied Sciences*, 26(1):397–413, 2019.

- [9] J. W. Kim, J. Salamon, P. Li, and J. P. Bello. Crepe: A convolutional representation for pitch estimation, 2018.
- [10] A. Klapuri. *Introduction to Music Transcription*, pages 3–20. 01 2006.
- [11] B. Lansdown. *Machine Learning for Music Genre Classification*. PhD thesis, 09 2019.
- [12] B. McFee, C. Raffel, D. Liang, D. Ellis, M. McVicar, E. Battenberg, and O. Nieto. librosa: Audio and music signal analysis in python. 2015.
- [13] C. Roads. Research in music and artificial intelligence. *ACM Comput. Surv.*, 17(2):163–190, June 1985.
- [14] M. Román, A. Pertusa, and J. Calvo-Zaragoza. An end-to-end framework for audio-to-score music transcription on monophonic excerpts. 09 2018.
- [15] V. Sarnatskyi, V. Ovcharenko, M. Tkachenko, S. Stirenko, Y. Gordienko, and A. Rojbi. Music transcription by deep learning with data and ”artificial semantic” augmentation. *CoRR*, abs/1712.03228, 2017.
- [16] B. Sturm, J. Santos, O. Ben-Tal, and I. Korshunova. Music transcription modelling and composition using deep learning. 04 2016.