Search

Home

Library

AI Music Transcriber

CPS 5881 - Independent Study:

AI Musical Audio to MIDI Transcriber

By: Alexander Fisher and Kevin Parra-Olmedo

Table of contents

01

Our vision

What, how, and why?

02

Inspirations

Existing software and other research papers

03

Our implementation

Model architecture, training, and app development

04

Results

What we learned and project demo

Search

Home

Library

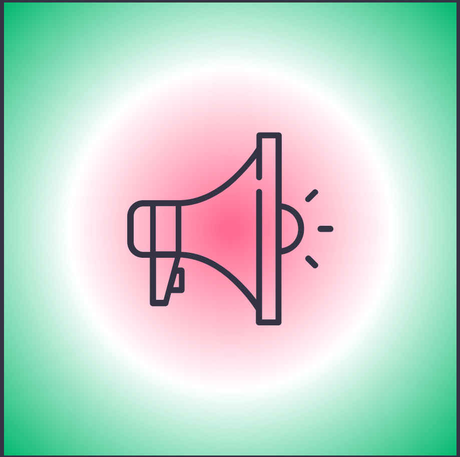
AI Music Transcriber

01

Our vision

What, how, and *why?*

What?



Our vision was to create an application that:

- Accepts a recording of raw musical audio as input
- Outputs a transcription of that raw musical audio in MIDI format

This isn’t as simple as it sounds. Music is a complex subject, and a song can be broken down into a long list of constituents.

How?

Our plan to realize this vision:

- Look at existing solutions to this problem (if any) for inspiration
- Implement our own solution using the initial research as a foundation
- We were most interested in including AI as a core backbone for our software solution
    - Complex problems require complex solutions

Why?

Music transcription is a tedious process, but the outcome is well worth it. What if it could be automated, though?

- Anyone interested in music could:
  - Break down any song's composition
  - Learn how to play a song, no matter the popularity of it

Search

Home

Library

AI Music Transcriber

02

Inspirations

Existing software and other research papers

Literature 1

Automatic Music Transcription: An Overview

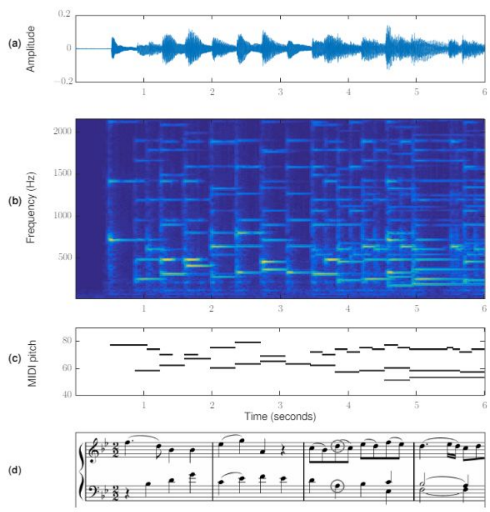


Figure 1: Different forms of musical representation

- Challenges of Automatic Music Transcription (AMT)
  - Polyphonic music contains mixture of multiple simultaneous sources with:
    - different pitch
    - loudness
    - timbre
  - These aspects make it more difficult to predict what individual notes are being played

E. Benetos, S. Dixon, Z. Duan and S. Ewert, "Automatic Music Transcription: An Overview," in IEEE Signal Processing Magazine, vol. 36, no. 1, pp. 20-30, Jan. 2019, doi: 10.1109/MSP.2018.2869928.
https://www.researchgate.net/publication/330068609_Automatic_Music_Transcription_An_Overview

Literature 2

Music transcription using a convolutional neural network

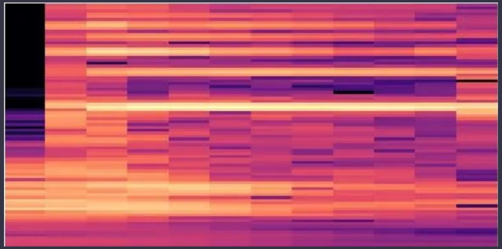


Figure 2: Q-transform spectrogram (audio squashed to logarithmic scale)

- CNN architecture
- Split audio files into samples of ⅛ second intervals
- Spectrogram for each sample was run through a CNN image classification network
- Drawbacks:
  - Loss of fidelity for note timing, length, and song BPM due to 1/8th second intervals

Verma, D. (2020, January 10). Music transcription using a convolutional neural network. Medium. https://medium.com/@dhruvverma/music-transcription-using-a-convolutional-neural-network-b115968829f4

Literature 3

A Lightweight Instrument-Agnostic Model for Polyphonic Note Transcription and Multipitch Estimation

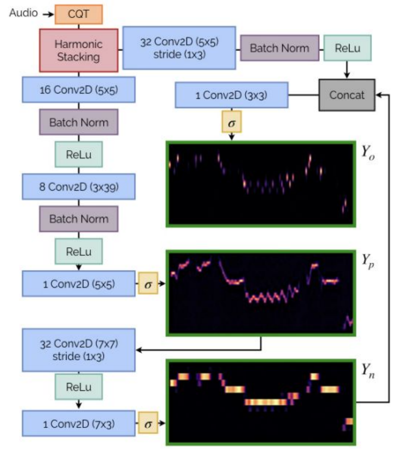


Fig. 1. The NMP architecture. The matrix posteriorgram outputs Y_o, Y_p, and Y_n are outlined in green. σ indicates a sigmoid activation.

- Simple CNN-based architecture
- Audio split into intervals of 2 seconds each
- Input: Harmonic-stacked CQT (~HCQT) (Constant Q Transform)
- Outputs:
  - Yo: Posteriorgram detects that the onset of a note is taking place (one bin per semitone)
  - Yp: Posteriorgram include 3 bins per semitone to more accurately capture small variations in pitch, like vibrato (acts as bottleneck layer)
  - Yn: Posteriorgram detects whether a note is active during the time frame using Yp, one bin per semitone

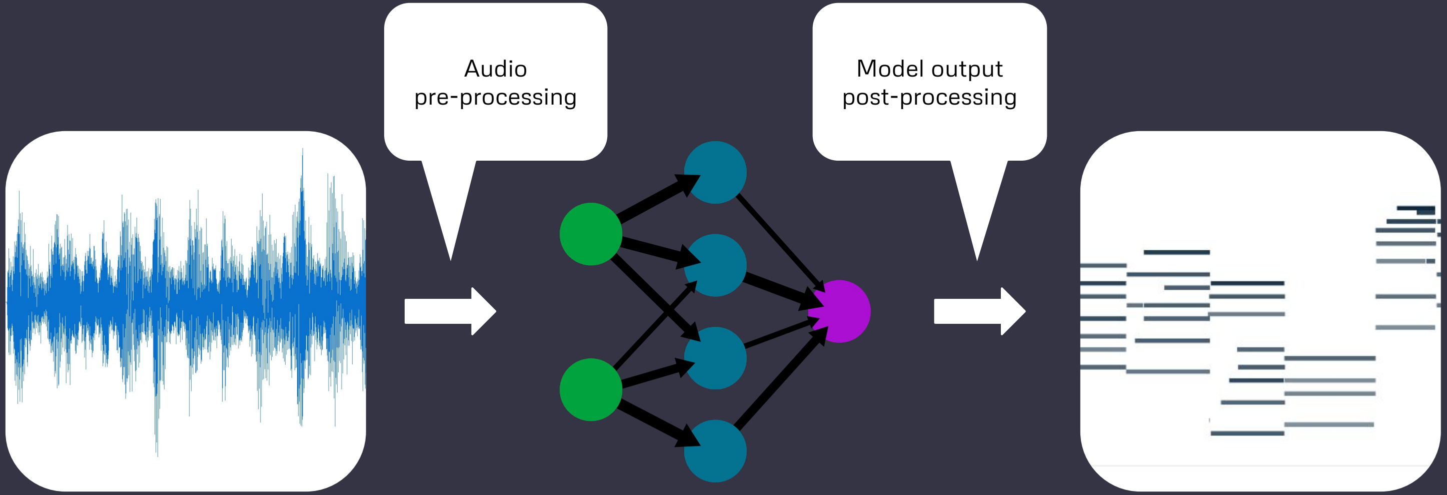
R. M. Bittner, J. J. Bosch, D. Rubinstein, G. Meseguer-Brocal and S. Ewert, "A Lightweight Instrument-Agnostic Model for Polyphonic Note Transcription and Multipitch Estimation," ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Singapore, Singapore, 2022, pp. 781-785, doi: 10.1109/ICASSP43922.2022.9746549.

Summary of Literature

Generalized methodology:

- **Input:** Audio waveform
- Computation of a time-frequency representation (spectrogram, posteriorgram, etc.)
- Estimation of output using CNN image classification with the computed visual graph representations
- **Output:** Representation of pitches over time (piano roll)

Summary of Literature (Visual Representation)



Audio pre-processing

Model output post-processing

Raw audio waveform

Neural network

MIDI Piano Roll

Search

Home

Library

AI Music Transcriber

03

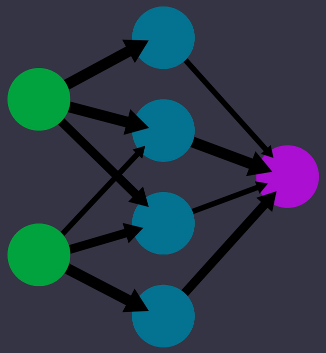
Our implementation

Model architecture, training, and app development

https://github.com/alexfishy12/audio_to_midi_vst

Model Architecture

Spotify®

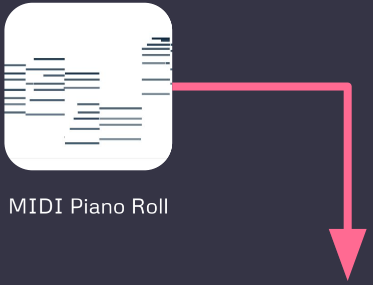


We chose to use Spotify's Basic Pitch model architecture as a baseline, and to potentially improve it in some way.

- Forked their repository on Github, and did a code analysis on what they provided
- Their public codebase was missing code for the following:
  - Data processing
    - Ground truth data
    - Part of input data
  - Model training
  - Model evaluation

https://github.com/spotify/basic-pitch

Data processing



MIDI Piano Roll

```
[[0,1,0],
 [1,0,0],
 [0,0,1]]
```

Binary Matrix (Notes x Time)

Input

- Raw audio was split into intervals of 2 seconds
- Sample rate was made uniform and audio was windowed for correct model input shape

Output validation

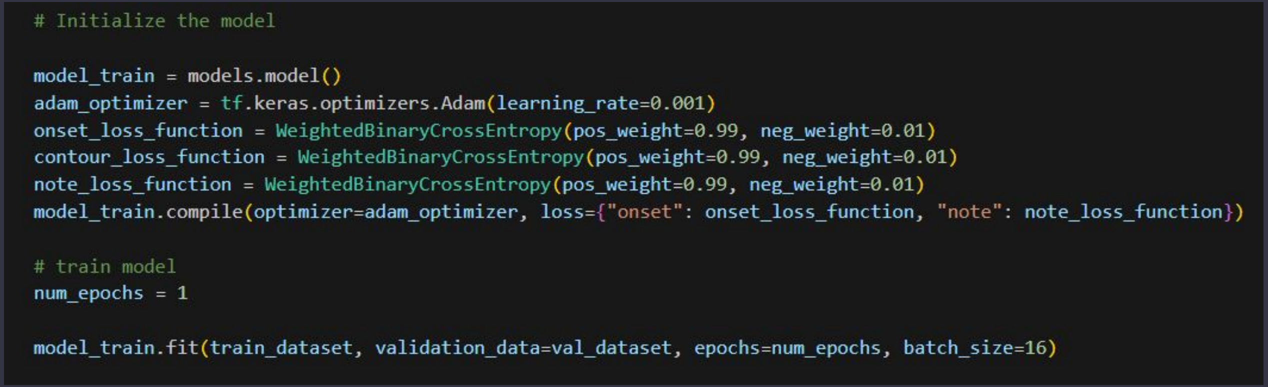
- Wrote a function to extract binary matrices of note onsets from ground truth MIDI files

Data Storage Pitfall

- A problem we encountered when trying to train our model was data storage
- Couldn’t store entire pre-processed dataset (~100GB) in memory, had to figure out ways to fetch data and pre-process on-the-fly as needed during the training loop
- Eventually learned how to use tensorflow’s “dataset.from_generator()” function that allowed us to pull data into memory for training, as needed
    - An alternative method that also worked for us was saving the pre-processed data as a binary file, but this was not a scalable solution

Model Training

- Weighted binary cross-entropy loss function
  - Our loss function had the model validate with a heavy class imbalance, where non-note prediction correctness contributed to the loss 1%, where actual note prediction correctness contributed to 99% of the loss



```
# Initialize the model

model_train = models.model()
adam_optimizer = tf.keras.optimizers.Adam(learning_rate=0.001)
onset_loss_function = WeightedBinaryCrossEntropy(pos_weight=0.99, neg_weight=0.01)
contour_loss_function = WeightedBinaryCrossEntropy(pos_weight=0.99, neg_weight=0.01)
note_loss_function = WeightedBinaryCrossEntropy(pos_weight=0.99, neg_weight=0.01)
model_train.compile(optimizer=adam_optimizer, loss={"onset": onset_loss_function, "note": note_loss_function})

# train model
num_epochs = 1

model_train.fit(train_dataset, validation_data=val_dataset, epochs=num_epochs, batch_size=16)
```

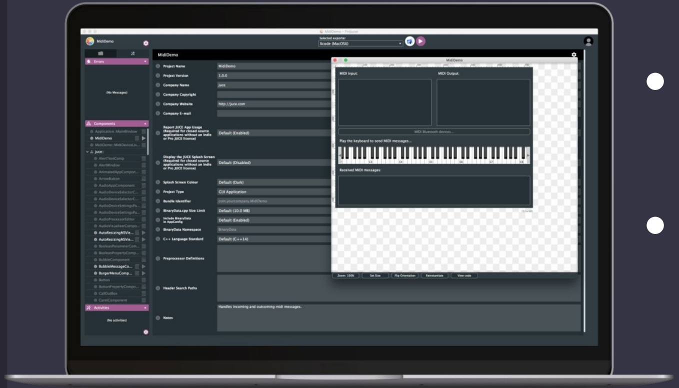
Model Evaluation

- Model evaluation was primarily done by ear, listening to the model's produced MIDI output and judge its accuracy to how the original audio sounds
  - We trained a model on ~127 songs from the MAESTRO dataset, and it produces decent results
- A shortcoming of our training pipeline is that the more data we train the model with, the worse it gets. This could be potentially because of the heavy class imbalance, where a majority of notes in each frame of the audio throughout the songs are labeled as "0" in the binary matrices
  - 0 represents no note, no sound
  - Thus, as the model trains on more data, the fewer notes it predicts to be in the resulting MIDI output

VST integration

- On our first iteration of the project we decided to implement a Virtual Studio Technology (VST). A VST holds the capabilities of being opened in a Digital Audio Workstation (DAW).
- The concept underlying the VST involved integrating the AI model with the VST, enabling them to collaboratively convert an audio file, selected by the user, into Midi notation transcription.
- It is important to note that based on some research that was conducted, a VST is written and built in the coding language known C++.
    - This also meant obtaining tools that support C++ and have the capability to build the program.

JUCE Framework



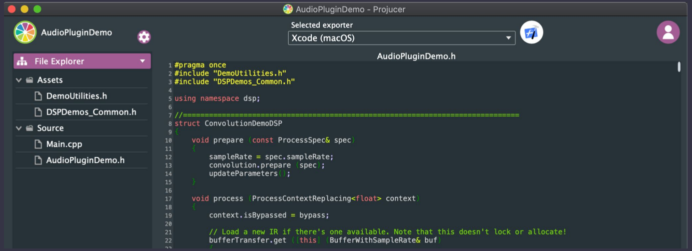
MidiDemo

Project Name: MidiDemo
Project Version: 1.0.0
Company Name: juce
Company Copyright:
Company Website: http://juce.com
Company E-mail:
Report JUCE App Usage (Required for closed source applications without an Indie or Pro JUCE license): Default (Enabled)
Display the JUCE Splash Screen (Required for closed source applications without an Indie or Pro JUCE license): Default (Disabled)
Splash Screen Colour: Default (Dark)
Project Type: GUI Application
Bundle Identifier: com.yourcompany.MidiDemo
BinaryData.cpp Size Limit: Default (10.0 MB)
Include BinaryData in AppConfig: Default (Enabled)
BinaryData Namespace: BinaryData
C++ Language Standard: Default (C++14)
Preprocessor Definitions:
Header Search Paths:
Notes: Handles incoming and outcoming midi messages.

- Upon further investigation the team discovered a framework known as JUCE that is commonly used to build VSTs.
- This framework came in the form of an application that allows users to capitalize the libraries and sample code provided by JUCE.
- Paired with the application known as Visual studio or Xcode, an IDE for Mac that supports C++, we began designing potential models for the VST

C++

- As stated before, the language used by the framework and the language needed to build such a program is C++
- JUCE provides sample C++ code that, when compiled, builds a very simple VST, This sample code were the bare minimum needed to build a fully functional VST.
- In order to set up the VST for our project purposes, we had to modify the existing code with our new code as well as utilizing the many libraries that JUCE had to offer to make the functions run properly.



```
#pragma once
#include "DemoUtilities.h"
#include "DSPDemos_Common.h"

using namespace dsp;

//==============================================================================
struct ConvolutionDemoDSP
{
    void prepare (const ProcessSpec& spec)
    {
        sampleRate = spec.sampleRate;
        convolution.prepare (spec);
        updateParameters();
    }

    void process (ProcessContextReplacing<float> context)
    {
        context.isBypassed = bypass;

        // Load a new IR if there's one available. Note that this doesn't lock or allocate!
        bufferTransfer.get ([this] (BufferWithSampleRate& buf)
```

Challenges

- Some challenges presented themselves during the implementation of the VST and JUCE.
- One of the challenges was that the team, although having had a general understanding of what needed to be done, miscalculated the scope of implementing a VST with the AI code
- The goal was once the VST was made, the AI model would be integrated into it to finalize the project and have a fully functional program.
- The unexpected challenge that arose from this was also that the AI model is written in Python, but in order to be integrated into the VST it had to be integrated in C++.

Resolution

- A collaborative decision was made amongst the team to switch gears and, instead of a VST program, we could build the program as an interactive web application.
- This decision was made so that we would still have a fully working program for the AI model to be interfaced with
- Web application structure:
  - Front-end: HTML, CSS, and javascript
  - Server: hosted on Kean's obi server
  - Back-end: written in python to interface with the model using a user-selection of saved weights (Spotify ICASSP 2022 or our model)
  - Utilized Docker to contain python dependencies

Search

Home

Library

AI Music Transcriber

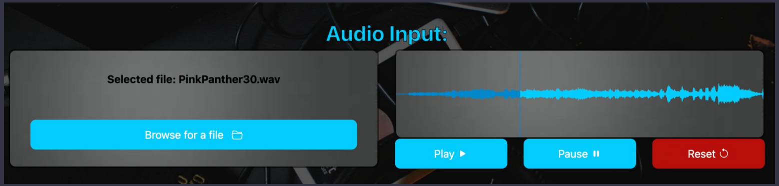
04

Results

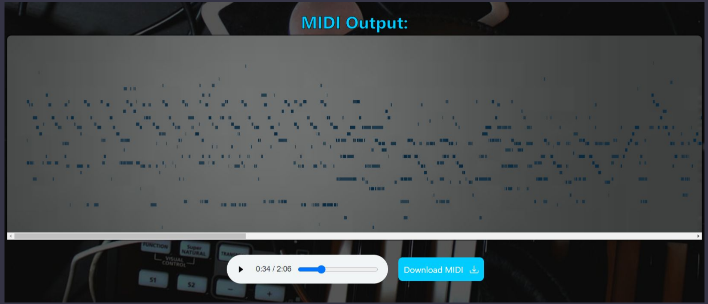
What we learned and project demo

Web Page

- Functionalities of the web page include:
  - Audio input function: this allows the user to input an audio file of their choosing whether it be a .WAV or an .mp3. The webpage will read the file and display a visual wave format of the audio and allow the user to play and pause the audio.
  - The webpage linked with the AI model processes the audio file, transcribing it to MIDI notation. Once this is done, it will display on the page, allowing the user to listen to the predicted MIDI transcription, as well as download it to place it into the DAW of their choice



Selected file: PinkPanther30.wav



http://www.alexjfisher.com/projects/audio_to_midi_transcriber/index.html

What we learned

- Fundamentals of C++ and the JUCE framework
- Fundamentals of Tensorflow machine learning framework
- AI model utilization to predict meaningful results from input audio data
- Process of programmatically generating MIDI data
- Process of building an AI-integrated web application
- How to host a website with a custom domain and containerized components

Search

Home

Library

Thanks!

Do you have any questions?

fisheral@kean.edu
parraolk@kean.edu
https://github.com/alexfishy12/audio_to_midi_vst

CREDITS: This presentation template was created by **Slidesgo**, and includes icons by **Flaticon** and infographics & images by **Freepik**

References

1. E. Benetos, S. Dixon, Z. Duan and S. Ewert, "Automatic Music Transcription: An Overview," in IEEE Signal Processing Magazine, vol. 36, no. 1, pp. 20-30, Jan. 2019, doi: 10.1109/MSP.2018.2869928. https://www.researchgate.net/publication/330068609_Automatic_Music_Transcription_An_Overview
2. R. M. Bittner, J. J. Bosch, D. Rubinstein, G. Meseguer-Brocal and S. Ewert, "A Lightweight Instrument-Agnostic Model for Polyphonic Note Transcription and Multipitch Estimation," ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Singapore, Singapore, 2022, pp. 781-785, doi: 10.1109/ICASSP43922.2022.9746549.
3. Verma, D. (2020, January 10). Music transcription using a convolutional neural network. Medium. https://medium.com/@dhruvverma/music-transcription-using-a-convolutional-neural-network-b115968829f4