Team Bassoon™

A subsidiary of the University of Michigan

To: Jeffery Fessler, Professor, University of Michigan Engineering ECE
Ella Alhuditi, Professor, University of Michigan Engineering TC

From: Adarsh Bharathwaj, Developer, Team Bassoon
Aiden Wenzel, Developer, Team Bassoon
Anthony Valencia, Developer, Team Bassoon
Emre Yavuz, Developer, Team Bassoon

Subject: Musical Learning Instrument Recognition Continuing Design Review

Date: Monday, April 8, 2024

Dist: Zachary Kerhoulas, Lab Coordinator, University of Michigan CSE & SMTD
Philip Derbesy, Professor, University of Michigan Engineering TC
Future Engineers of ENGR 100-436

1. Foreword

The development of our machine learning based instrument recognition system is underway. You have asked us for an update in the development of our project, with more specifications such as a system overview, a testing plan, and the next steps that we will take before the project's due date. This report contains all those details and specifications.

2. Summary

Machine Learning Instrument Recognition (MLIR) is a machine learning model that is designed to be able to identify what instruments are being played given an audio sample. The purpose of this project is to address the needs of sound engineers who want to know what instrument is being played in an audio sample for use in music recommendation algorithms. MLIR is designed to be a graphical user interface (GUI) and likely a command line interface (CLI) for users to input a sound file and receive a prediction on which instruments have been played. The interface will work where the user inputs a sound file which goes through preprocessing (via fast-fourier transform spectrums) and then the model which provides the necessary output. The current status of the project is that the machine learning model and GUI have started. Regarding testing, we are using a split dataset with the first part to train the model and the second part to test the model.

Our next steps are to test the model accordingly and connect the interfaces (GUI and/or CLI) to the model for delivery of the product.

3. Problem Statement

3.1 Problem Statement Summary

Sound engineers have a hard time identifying what musical instrument is being played when they are recommending music. Providing sound engineers with a method of identifying what musical instrument is being played will help sound engineers recommend music to their users, sort their music according to instrument being played and compute similarities between compositions. A possible solution should include ease of use through input and output of audio files, being able to identify three separate instruments in noise and multi-instrument tracks, and being quick to execute.

3.2 Problem Definition and Background

Automatic music transcription (AMT) converts musical signals into musical notation [1]. AMT is used by many audio engineers in post-production of music for organizing and identifying musical instruments in various tracks [2]. Musical instrument recognition is used for recommendation systems for music streaming services, computing similarities between compositions, and filtering music based off of instrument played [3].

Musical instrument recognition currently has a hard time identifying multiple instruments within an audio sample [2,3]. Creating a solution to this problem will create an easier method for identifying overlapping musical instruments and supporting sound engineers in music streaming services for recommendation.

3.3 User Needs

3.3.1 Recommendations

For sound engineers at streaming companies, they want to use the instruments present in one track to recommend tracks that are similar to the sounds of the original track. This is because they want their users to easily access music they enjoy. Recommendation systems are a popular way that streaming services share music with users.

3.3.2 Easy to Embed into Existing Project

Sound engineers want to easily embed the solution into their workflow so that it can be used efficiently. This is because the engineers have a developed system and do not want to remake their system to support our solution.

3.4 Functional Requirements

3.4.1 Quick Response Time

Since sound engineers will be using this solution in a real-time environment, the solution should be quick enough to respond within ten seconds. Users will lose patience and faith in the application if their wait time is longer than ten seconds [4].

3.4.2 Quick to Learn

Learning how to use the solution should be under five minutes. The typical attention span a human can hold is ten to fifteen minutes [5]. Since the solution will have little amount of features, we think it is reasonable that a person can learn within five minutes.

3.4.3 Can Accurately Identify Three Musical Instruments

The solution should be able to identify a variety of instruments inside of a given track. Due to the time constraints of this project we have selected three instruments that cover different instrument classifications.

4. Software overview and progress

4.1 Overview

Our software will operate like a desktop application. Both the frontend GUI and the middleware machine learning models are built on the popular, open-source Python language and it utilizes several Python libraries which will be discussed in the frontend and backend sections.

4.2 GUI

Our frontend GUI is built on the Tkinter python library. Tkinter is a Python binding to the Tk GUI toolkit. The Tk toolkit is a collection of widgets which are used in tandem to build intuitive, simple GUIs.

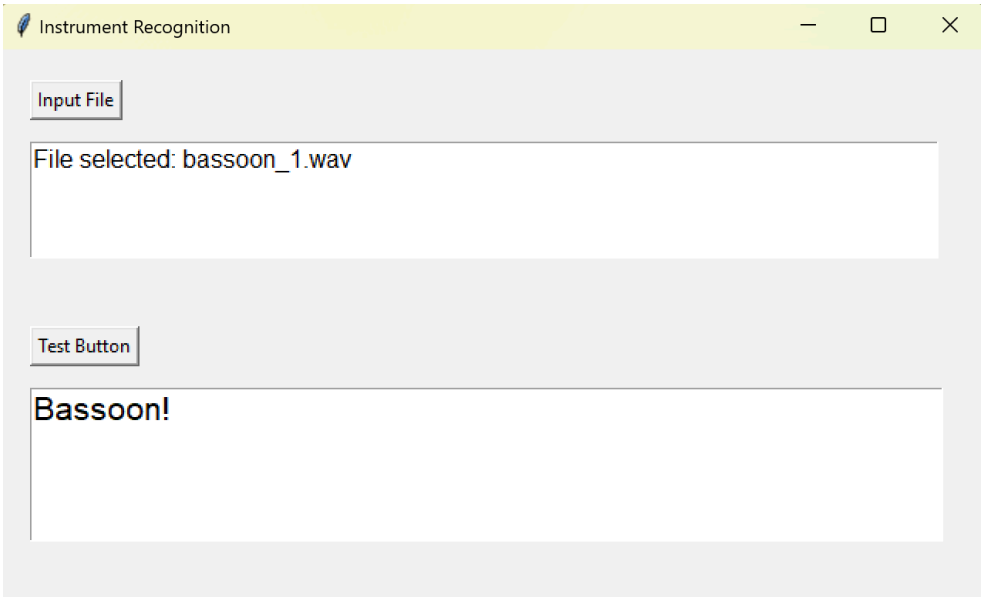

Figure 1: App GUI

Our GUI (see *Figure 1*) contains 4 total widgets: an input file button, an input file textbox, a test button, and the final output textbox. The input file button, when clicked, prompts the user to select a file by opening their operating system's native file explorer. The input file text box tells the user which file they selected and informs them if they have chosen a file which is not in the WAV format. The test button is linked to a function which, when clicked, preprocesses the audio file, sends it to a machine learning model for prediction, and outputs the name of the predicted instrument. The final textbox simply outputs the name of the predicted instrument to the screen.

4.3 Machine Learning Software

Our software will consist of three stages. The first stage is a preprocessing stage where we process the .WAV file into machine readable data. The second stage is the training stage, in which each of the machine learning models is inputted with the preprocessed data to begin training with. The final stage is the prediction stage where the .WAV file data is processed and then passed into a pre-trained machine learning model in which the predicted instrument is returned as a string.

In the preprocessing stage, we read in the .WAV file into a one dimensional array making sure to downsample to 22.05kHz. Then chunk the sample into 1 second long chunks. Then, using the Librosa library, every chunk is turned into a mel-spectrogram, which is a spectrogram of the melody scale (mel-scale) that more accurately reflects human's discernment of pitches [6]. The mel-spectrogram is then decibel scaled, resulting in a 2 dimensional matrix. Additionally, multiple audio tracks are combined into one audio track to create multi-instrument samples for training.

In the training stage, we take the preprocessed data and train our three machine learning models: the multilayer perceptron (MLP), convolutional neural network (CNN), and convolutional recurrent neural network (CRNN). For the MLP model, the preprocessed data is flattened into a vector and then trained with each track. For the CNN and CRNN models, the preprocessed data train directly on the outputted matrices from the preprocessing stage. Finally, after training, we create testing benchmarks (see *Section 5.1.2*) to analyze the performance of each model. After the performance is benchmarked, the most accurate model will be used in the application.

In the prediction stage, each model has been trained and the user specifies a .WAV file for each model to read in. Once this occurs, each model will output an instrument that best fits their expectations of the data. This prediction will be directly printed into the GUI for the user to view.

4.4 Installation

To install and use the app, the user must first install python, clone the app's github repository, and install necessary dependencies and their corresponding versions (see *Appendix A*).

5. Testing plan

To ensure that our product is functioning, we have developed a method of testing to see firstly, how each component works independently and secondly, how the components work as a full product.

5.1 Individual Tests

5.1.1 User Interface Testing

For the testing of the GUI we are making sure that it works by confirming that buttons correspond to a given function. This means that when a button is selected, there should be an output put to the output window of the screen.

5.1.2 Model Testing

For the machine learning model we are specifically looking at accuracy that the model we are using can return the instrument for an inputted audio file. To do this we are using three different models: a multilayer perceptron (MLP), a convolutional recurrent neural network (CRNN), and a convolutional neural network (CNN). The reason for the use of these specific models is because they are the three broadest classes of deep neural networks (DNN). In industry, DNNs are especially helpful because they can identify features in raw or minimally unprocessed data [7].

With the advantages of the DNNs we will be looking at which class of DNNs gives the best accuracy. Our evaluation process of the accuracy includes the use of two methods, Label

Ranking Average Precision (LRAP) and Area Under Receiver Operating Characteristic Curves (AUC-ROC).

Label Ranking Average Precision

In short, LRAP averages a ground truth ranking vs. the model's confidence rankings for our set of instruments from an input audio sample [8].

```
>>> import numpy as np
>>> from sklearn.metrics import label_ranking_average_precision_score
>>> y_true = np.array([[1, 0, 0], [0, 0, 1]])
>>> y_score = np.array([[0.75, 0.5, 1], [1, 0.2, 0.1]])
>>> label_ranking_average_precision_score(y_true, y_score)
0.416…
```

Figure 2. Python demonstration of scikit-learn's LRAP function [9].

Given the ground truths of y_true and the scores from the model of y_scores, scikit-learn's LRAP function takes the ratio of the ground truth–y_true– to each respective rank (the index when ordered from highest to lowest) of the prediction from the model–y_true. After taking each ratio, the found values are averaged up to find the label ranking average precision (LRAP) of the model.

Area Under Receiver Operating Characteristics Curves

AUC-ROC, on the other hand, graphs the rates of true and false positives at different classification thresholds and finds the probability that the model ranks a random positive higher than a random negative [10].

An ROC graph is made from calculating and plotting the true positive rate and false positive rate along the y and x axis respectively.

$$TPR = \frac{TP}{TP+FN} \quad (1)$$

Where TPR is true positive rate, TP is the number of true positives counted, and FN is the number of false positives counted at a given classification threshold.

$$FPR = \frac{FP}{FP+TN} \quad (2)$$

Where FPR is the false positive rate, FP is the number of false positives counted, and TN is the number of true falses.

When Graphed the ROC looks like Figure 3.

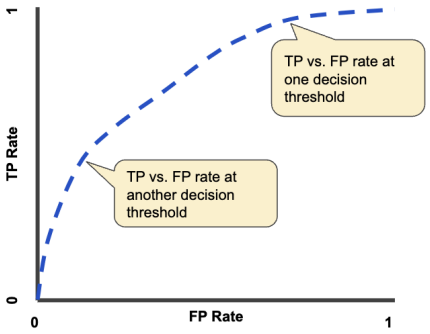

Figure 3. ROC Graph of FPR vs. TPR at different classification thresholds [11]

The next measure AUC refers to the Area under Figure 3.

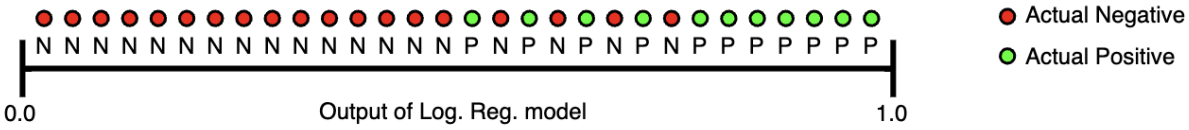

Figure 4. Rankings of a Logistic Regression model's rankings [11]

Given the definition of ranking a random positive higher than a random negative, the AUC represents the probability that a positive (green point) example is to the right of a negative (red point) example in Figure 4 [11].

5.2 Combined Tests

Testing the combined system will include taking audio files and running them through the whole system. We will then compare the interfaces' output to the output of the model under the same conditions to confirm that the proper values are being passed through and not corrupted by the user interfaces.

6. Next Steps

As of now, our progress on the project is smoother than expected. Right now, we have a working, bare-bones GUI with an input file button and text boxes that output relevant information. Additionally, we have completed functions to load, process, and store data to train our three machine learning models. Perhaps most excitingly, the MLP model has been trained and is showing promising results when predicting.

Although we have seen a great deal of progress, there is more work to be finished. While we have the scaffolding for the CNN and CRNN, we still need to fully implement them and alter the loading functions so that they convert audio data into a form that the networks can use. After that

is done, we need to train our models with the database of audio samples we have. However, this shouldn't be too difficult and should only take a little time and computing power. Once the models are trained, they should then be tested to determine which model is going to be the most accurate. Finally, after the model has been chosen, we need to link the model to the front end GUI so that, once a file is selected, and a button is pushed, the audio data is processed, sent through the network, and the predicted instrument is displayed to the screen.

References

[1] E. Benetos, S. Dixon, D. Giannoulis, H. Kirchhoff, and A. Klapuri, "Automatic music transcription: challenges and future directions," *Journal of Intelligent Information Systems*, vol. 41, no. 3, pp. 407–434, Jul. 2013, doi: https://doi.org/10.1007/s10844-013-0258-3.

[2] M. Blaszke and B. Kostek, "Musical Instrument Identification Using Deep Learning Approach," Sensors, vol. 22, no. 8, p. 3033, Apr. 2022, doi: https://doi.org/10.3390/s22083033.

[3] D. Szeliga, P. Tarasiuk, B. Stasiak, and P. S. Szczepaniak, "Musical Instrument Recognition with a Convolutional Neural Network and Staged Training," *Procedia Computer Science*, vol. 207, pp. 2493–2502, 2022, doi: https://doi.org/10.1016/j.procs.2022.09.307.

[4] R. B. Miller, "Response time in man-computer conversational transactions," *Proceedings of the December 9-11, 1968, fall joint computer conference, part I on - AFIPS '68 (Fall, part I)*, 1968, doi: https://doi.org/10.1145/1476589.1476628.

[5] N. Bradbury, "Attention span during lectures: 8 seconds, 10 minutes, or more?," *Advances in Physiology Education*, vol. 40, no. 4, pp. 509–513, Nov. 2016, doi: https://doi.org/10.1152/advan.00109.2016.

[6] S. S. Stevens, J. Volkmann, and E. B. Newman, "A scale for the measurement of the psychological magnitude pitch," The Journal of the Acoustical Society of America, vol. 8, no. 3, pp. 185–190, Jan. 1937. doi:10.1121/1.1915893

[7] Siddharth Gururani, C. Summers, and A. Lerch, "Instrument Activity Detection in Polyphonic Music using Deep Neural Networks.," pp. 569–576, Jan. 2018.

[8] Robert E Schapire and Yoram Singer. Boostexter: A boosting-based system for text categorization. Machine learning, 39(2-3):135–168, 2000.

[9] G. Varoquaux, L. Buitinck, G. Louppe, O. Grisel, F. Pedregosa, and A. Mueller, "Scikit-learn," *GetMobile: Mobile Computing and Communications*, vol. 19, no. 1, pp. 29–33, Jun. 2015, doi: https://doi.org/10.1145/2786984.2786995.

[10] Yoonchang Han, Jaehun Kim, Kyogu Lee, Yoonchang Han, Jaehun Kim, and Kyogu Lee. Deep convolutional neural networks for predominant instrument recognition in

polyphonic music. IEEE/ACM Transactions on Audio, Speech and Language Processing (TASLP), 25(1):208– 221, 2017.

[11] Google, “Classification: ROC Curve and AUC | Machine Learning,” *Google for Developers*. https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc#:~:text=AUC%3A%20Area%20Under%20the%20ROC

Appendix A: GitHub Installation Instructions

README.md file found at https://github.com/aiden-wenzel/ENGR_100_Project_3.

Instrument Recognition App

1 Installation

1.1 Install Python
To install Python, navigate to Python's official downloads page at:
https://www.python.org/downloads/.

Once you have completed the download process on Python's website, verify your installation by opening your operating system's terminal and typing `python --version`. If your installation was successful, you should see `Python <version>` outputted in the terminal.

1.2 Download Files

After downloading Python, either run `git clone
https://github.com/aiden-wenzel/ENGR_100_Project_3` if you have git or download the zipped source code by first pressing the green code button, then clicking download zipped. Make sure to extract the files if you downloaded the zip.

1.3 Install Required Packages

Once the files have been downloaded, open the terminal in the folder where `app.py` and `requirements.txt` are.

Then run the following command
`python -m pip install -r requirements.txt`. This command will install all the required python packages and allow you to run `app.py`.

1.4 Running the Application

In the terminal, make sure that you are in the `app.py` directory. Then run `python app.py`. This should run the file and then open the GUI window.