MLIR: Machine Learning Instrument Recognition

Submitted April 29, 2024

Team Bassoon™

A subsidiary of the University of Michigan

bassoon@umich.edu
Ann Arbor, MI 48109

Adarsh Bharathwaj
Tony Valencia
Aiden Wenzel
Emre Yavuz
Product Engineers

Submitted to the University of Michigan for numeric academic feedback

Table of Contents

Executive Summary

Machine Learning Instrument Recognition (MLIR) is a machine learning model interface 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 instruments are being played so that they sort easily to create recommendation systems. MLIR was also designed to be easy to embed into existing software so that sound engineers do not have to rebuild their codebases around its interface.

The functional requirements of MLIR include a quick response time of at most 10 seconds for a 3 minute long song, an under 5 minute learning time for the interface, and the ability to recognize an oboe, trumpet, and violin with at least 60% accuracy for studio recordings.

For basic use, MLIR is designed to be a graphical user interface (GUI) for users to input a sound file and receive a prediction on which instruments are playing in the file. The interface will work where the user inputs a sound file which goes through a Mel-Spectrogram and then processed to fit into a machine learning model, which uses feature recognition from previous Mel-Spectrograms to identify oboes, trumpets, and violins.

To find the best machine learning models we trained a Multi-Layer Perceptron (MLP), Convolutional Neural Network (CNN), and a Convolutional Recurrent Neural Network (CRNN).

Regarding testing, we are using a three part split dataset of 80/10/10 with the first part to train the model, the second part to validate the model while training, and the third part to test the model. The accuracy of models for training and validation are all above 60% which is a measure of total correct predictions and total predictions made. Each of the models has also been tested for accuracy using a confusion matrix. The CNN had the highest accuracy, followed by the CRNN, which both had accuracies above 60% for each instrument, and lastly the MLP which was under the 60% threshold for the trumpet.

Introduction

Automatic music transcription (AMT) converts musical signals into musical notation [2]. AMT is used by many audio engineers in post-production of music for organizing and identifying musical instruments in various tracks [3]. Musical instrument recognition is used for recommendation systems for music streaming services, computing similarities between compositions, and filtering music based off of instrument played [4].

Since machine learning is still in its infancy, we want to see where of the many use cases it can be applied to. The biggest part of this project is finding and training the best machine learning models for the problem statement. After reading some papers we decided to use the Multi-Layer Perceptron (MLP), Convolutional Neural Network (CNN), and Convolutional Recurrent Neural Network (CRNN) based on the recommendations of researchers Siddharth Gururani, Cameron Summers, and Alexander Lerch [1].

Problem Statement

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.

Problem Definition:

Musical instrument recognition currently has a hard time identifying multiple instruments within an audio sample [3,4]. 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 softwares.

User Needs:

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.

Easily Embeddable:

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.

Functional Requirements

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 [5].

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 [6]. Since the solution will have little amount of features, we think it is reasonable that a person can learn within five minutes.

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. For our baseline accuracy, we wanted each model to predict instruments correctly at least 60% of the time. This measurement took into account that only 42.5% to 60% of the time, humans were able to correctly identify instruments [7].

System Overview

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. In Figure 1, there is a simplified flowchart of how the user interacts with the system as a whole.

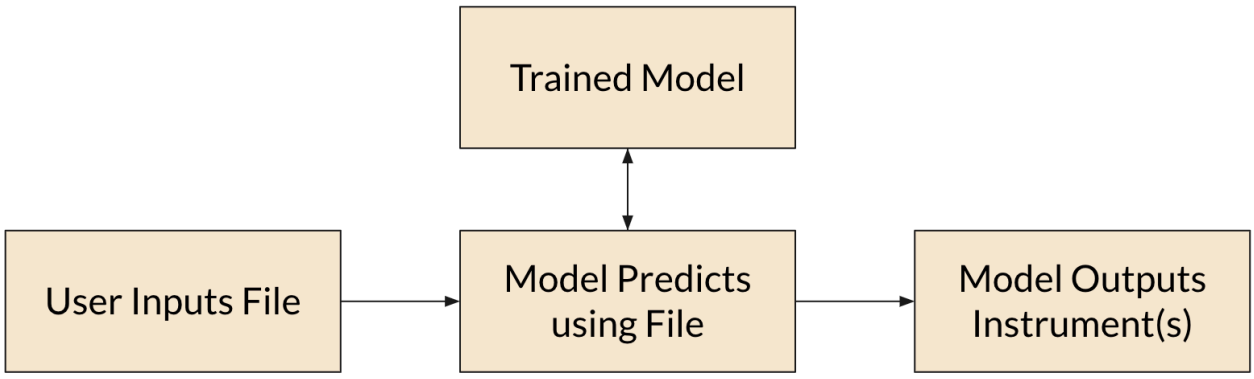

Figure 1: MLIR System Pipeline

Graphical User Interface (GUI):

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

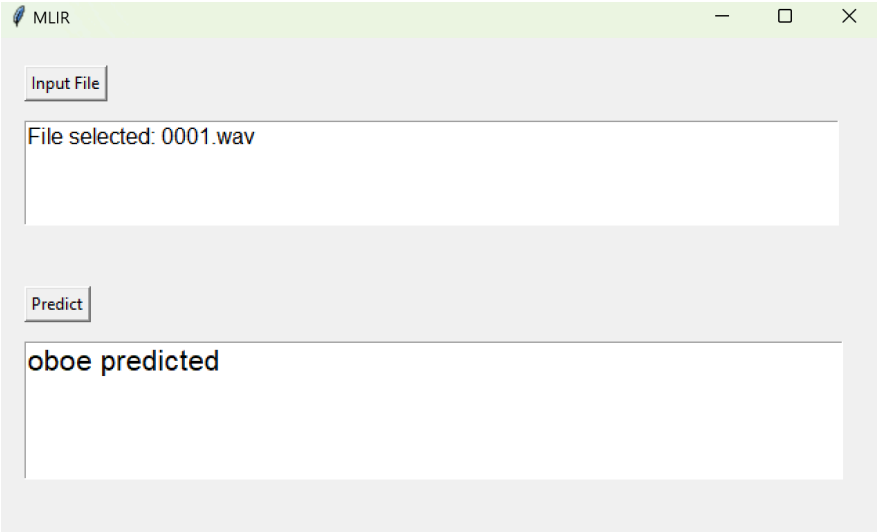

Figure 2: App GUI

Our GUI (see Figure 2) 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 that is not a 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 stores the name of the predicted instrument. The final textbox simply outputs the name of the predicted instrument to the screen.

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. We then chunk the sample into 1 second long chunks. Using the Librosa library, each chunk is then turned into a mel-spectrogram, which is a spectrogram of the melody scale (mel-scale) that more accurately reflects human's discernment of pitches [8]. 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 better 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, which consists of 4 hidden layers with 256 hidden units with each of them, the preprocessed data is first flattened into a vector and then trained with each track. For the CNN and CRNN models (see Appendix B for the architecture), the preprocessed data train directly on the output matrices from the preprocessing stage. The data was passed through multiple layers within each of these models to form pattern recognition. Finally, after training, we create testing benchmarks (see Testing) to analyze the performance of each model. After the performance is benchmarked, the most accurate model will be used in the application.

These three models were chosen since Deep Neural Networks (DNNs), a subset of machine learning that has multiple layers for learning [14], have consistently outperformed other forms of machine learning and instrument- and music-related activities, such as music transcription or tagging [7]. As such, the MLP model is the simplest form of a DNN and was chosen as a baseline for predictions and testing. The CNN and CRNN models were chosen because of their strong ability to recognize patterns with "non-numeric" data (such as music or images) [12]. In addition, the CRNN model, which has been commonly used in music-based machine learning, has the added advantage of being recurrent, which provides increased accuracy in time-series data due to how each of the model's layers interact with one another [13].

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, the most accurate model will output an instrument that best fits its expectations of the data. This prediction will be directly printed into the GUI for the user to view.

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).

Testing

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.

Individual Tests:

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.

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 analyzing the results of our three models: a MLP, a CNN, and a CRNN. 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, training/validation accuracies and multilabel confusion matrices.

Training and Validation Accuracies:
When training the models, we calculated the training and validation accuracy by accumulating each correct prediction and the total predictions for each respective datasets. We then plotted each accuracy over time for the individual models. Naturally, as the model becomes more accustomed to our training set, the accuracy and validation should increase which indicates that our model is learning. We chose to train 20 epochs, or iterations of data, because, after tha point, the model becomes susceptible to overfitting, in which our model only becomes accurate on the data we provided it, and can't make conclusions about other, new data [16]. Once training was complete, we collected the accuracies and plotted them:

For the MLP (see Figure 3), both accuracies increased logarithmically and stopped at 93% accuracy for both the training and validation datasets.

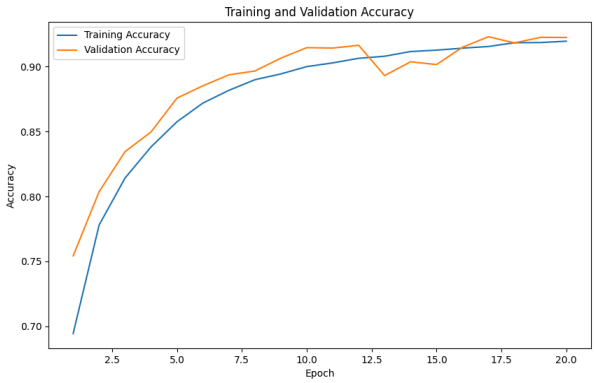

Figure 3: MLP Validation accuracy graph

For the CNN (see Figure 4), accuracies also increased logarithmically and finally stopped at around 92% accuracy for both the training and validation datasets.

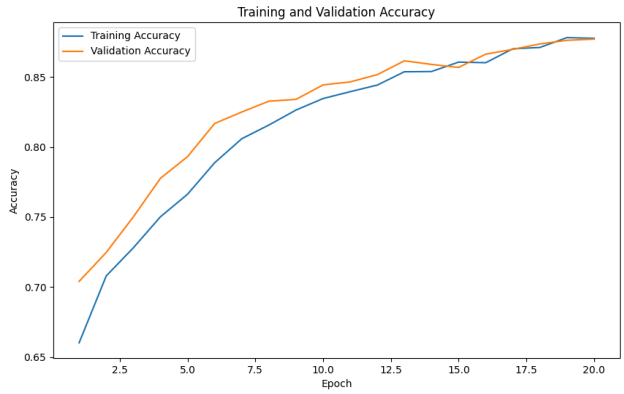

Figure 4: CNN Validation accuracy graph

The CRNN's accuracies (see Figure 5) followed a linear trajectory stopping at 89% for the training set and at 88% for validation of the model.

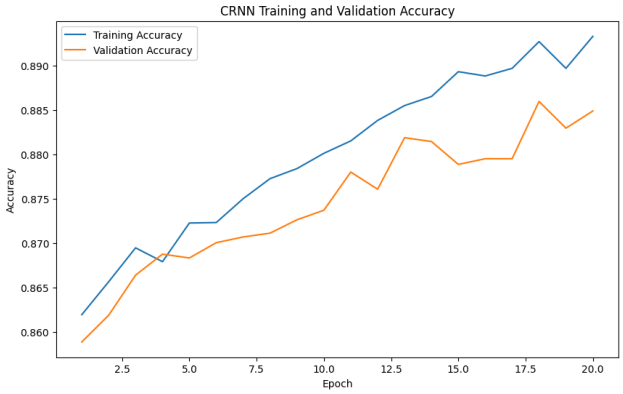

Figure 5: CRNN Validation accuracy graph

Based on the results of the training and validation accuracies, every model performed well, with the CNN and MLP models performing the best, at around 92% accuracy.

Multilabel Confusion Matrices:

A confusion matrix is a table used for classification algorithms by comparing the actual versus predicted outputs. A sample is shown in Figure 6 where TP indicates a true positive, FP indicates a false positive, FN indicates a false negative, and TN indicates a true negative. For an ideal model, both false positives and false negatives will be close to 0.

	Positive (1)	Negative (0)
Positive (1)	TP	FP
Negative (0)	FN	TN

Figure 6: Arbitrary confusion matrix
Source: Adapted from [17]

We can calculate accuracy for the training dataset by using the following equation,

$$accuracy = \frac{TP + TN}{TP + FP + FN + TN} \quad (1)$$

where TP is the number of true positives, TN is the number of true negatives, FP is the number of false positives, and FN the number of false negatives in a given number of predictions.

In MLIR, we specifically used a multilabel confusion matrix because we wanted to identify multiple instruments playing in a track. Our models returned true or false values according to if an instrument is inside the track or not. This means that we will get three separate binary confusion matrices for each of the instruments for a given track. Our testing data set is 98 tracks which contains oboes, trumpets, violins, and a random mix of the instruments–this is the denominator of Equation 1. We also added a heat map where darker colors indicate a higher number and therefore denser matrix value. Although it is still important to consider the numbers when looking at the matrix.

For the MLP model (see Figure 7), there were mixed results with identifying a true positive with oboes but strong results with a true negative, mixed results toward finding trumpets in general due to the close values, and strong true positives for the violin but insignificant true negatives. The accuracies for the oboe, trumpet, and violin on the training set are 72%, 51%, and 70% respectively.

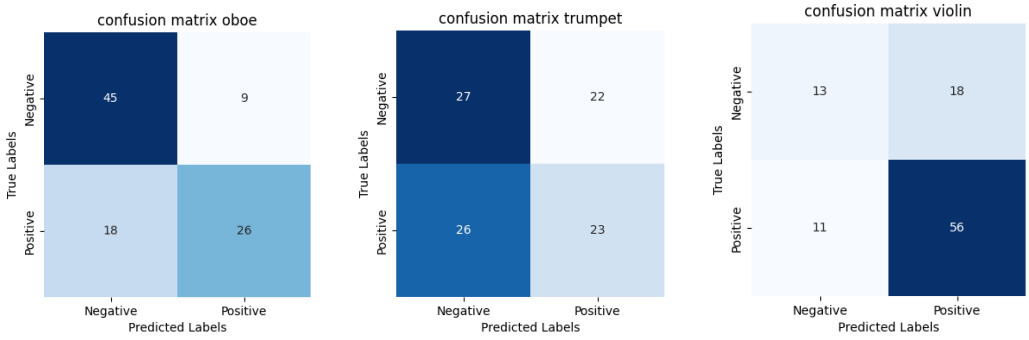

Figure 7: Multilabel confusion matrix for MLP

For our CNN model (see Figure 8), we noticed that it performed well at identifying the oboe, while it had a positive bias towards the trumpet, and a positive bias towards the violin. The accuracies for the oboe, trumpet, and violin using the CNN in the training dataset are 87%, 61%, and 70% respectively.

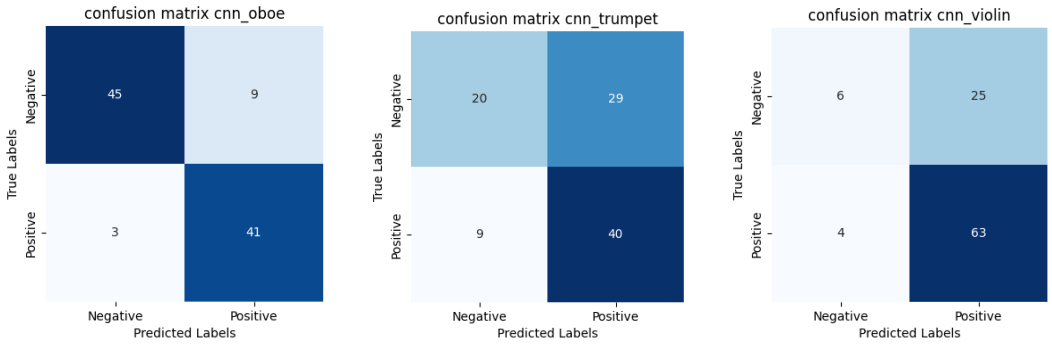

Figure 8: Multilabel confusion matrix for CNN

For the CRNN model, we noticed that it identified the oboe well, identified many true positives for trumpets but not necessarily an instance where there were no trumpets, and had a positive bias toward violins Figure 9. The accuracies for the oboe, trumpet, and violin using the CRNN in the training dataset are 79%, 70%, and 68% respectively.

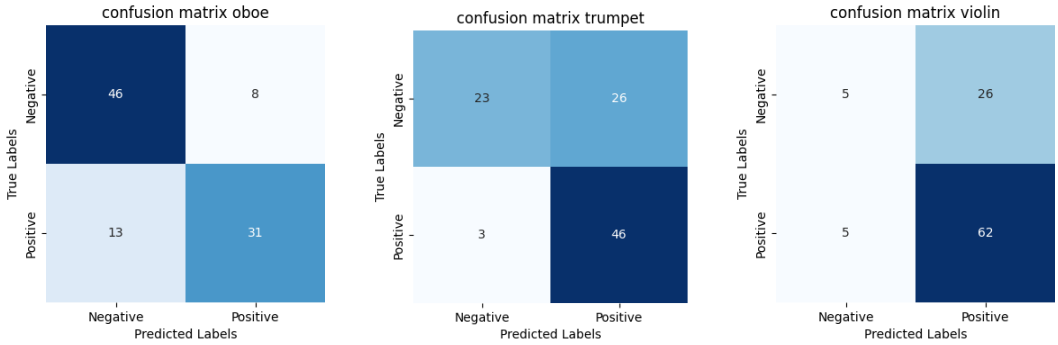

Figure 9: Multilabel confusion matrix for CRNN

With the multilabel confusion matrix tests, the MLP didn't perform well overall whereas the CNN and CRNN had performed with much higher accuracy.

Combined Tests:

When linking the machine learning models to the GUI, we chose to use the CNN due to its good performance in both the training/validation accuracies as well as its multilabel confusion matrix performance. After successfully linking the GUI with the model, we had found that the model has a strong positive bias towards the oboe, in which some violin samples were incorrectly predicted as being predominantly oboes.

Future Work

The development of MLIR was made to fit a small deadline so we could not manage to thoroughly cover everything to make our models the best that they can be. Future work includes: (1) Adding more instruments for our models to recognize, (2) Training our models to be more accurate through using more datasets and complex tracks, (3) Increasing the user experience by making a web application or command-line interface, (4) Explore different model architectures

and identify more key features of each instrument, (5) Optimizing metrics outside of accuracy, which are all expanded upon below:

1. To address more instruments we need to find more recordings and data of instruments to train our models. This would increase the range of MLIR and make it actually consumer grade.
2. Along with adding more instruments is using more datasets. Training our models to get better at identifying instruments with different signal-to-noise ratios will increase the range of MLIR and allow us to address problems such as having noise in songs.
3. A web service, command line application, or published library will reach our targeted user of sound engineers better because they will be using our solution with big batches of data.
4. There are potentially better model architectures that are better at identifying key features that are not identified by our existing models. Along with identifying key features is providing our models with more input data to distinguish instruments other than Mel Spectrograms.
5. MLIR's functional requirements looked specifically at the accuracies of the models and not their precision, recall, or F1 scores. Optimizing to get a better balance of all these metrics may offer better recognition of instruments.

Conclusions and Recommendations

The goal of this project was to create an accurate and easy-to-use method in which sound engineers can input a music file and obtain which instruments were inside the given file. After researching and experimenting, we decided to use a multilayer-perceptron (MLP), convolutional neural network (CNN), and convolutional recurrent neural network (CRNN) to recognize and output instruments within a file.

To test each model, we calculated accuracies for our training, validation, and testing datasets. The CNN and CRNN had accuracies for each instrument over 60%. The MLP had 50% accuracy for the trumpet in the training dataset but was over the threshold 60% with every other instrument and dataset. The best overall performance, in terms of mean accuracy for all instruments, came from the CNN. Each of the models were tuned to identify oboes but had a hard time identifying trumpets and violins, typically with biases to predict positive or negative all the time. We did not test the user interface but we made an effort to make the interface easy to understand with information on how to install and use the application, and a simple GUI.

To make MLIR better at recognizing musical instruments, we would make implementations such as adding more instruments to recognize, data to train on, and user interfaces. We can also train on different model architectures and optimize other performance metrics.

If you have any questions, concerns, design recommendations, or improvements, please email bassoon@umich.edu.

References

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

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] S. K. Zieliński, H. Lee, P. Antoniuk, and O. Dadan, “A comparison of human against machine-classification of spatial audio scenes in binaural recordings of Music,” Applied Sciences, vol. 10, no. 17, p. 5956, Aug. 2020. doi:10.3390/app10175956

[8] 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

[9] 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

[12] M. M. Taye, “Theoretical understanding of Convolutional Neural Network: Concepts, architectures, applications, Future Directions,” Computation, vol. 11, no. 3, p. 52, Mar. 2023. doi:10.3390/computation11030052

[13] N. Buhl, “Time series predictions with RNNS,” Time Series Predictions with RNNs, https://encord.com/blog/time-series-predictions-with-recurrent-neural-networks/ (accessed Apr. 28, 2024).

[14] C. Joo, H. Kwon, J. Kim, H. Cho, and J. Lee, “Machine-learning-based optimization of operating conditions of naphtha cracking furnace to maximize plant profit,” Computer Aided Chemical Engineering, pp. 1397–1402, Jun. 2023. doi:10.1016/b978-0-443-15274-0.50222-5

[16] X. Ying, “An overview of overfitting and its solutions,” Journal of Physics: Conference Series, vol. 1168, p. 022022, Feb. 2019. doi:10.1088/1742-6596/1168/2/022022

[17] S. Narkhede, “Understanding confusion matrix,” Medium, https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62 (accessed Apr. 19, 2024).

Appendix A. GitHub Installation Instructions from the "README.md"

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 `requirements.txt` is.

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 `main.py`.

1.4 Running the Application

In the terminal, type `cd src`. After that type, `python main.py`. This should run the main script and open the GUI.

Appendix B. Model Architecture for the Convolutional Neural Network and Convolutional Recurrent Neural Network.

CNN Architecture

CNN
Conv2D $k = 3 \times 3, d = 64$ MP $(p = 2, 2)$
Conv2D $k = 3 \times 3, d = 128$ MP $(p = 2, 2)$
Conv2D $k = 3 \times 3, d = 256$ MP $(p = 3, 3)$
Conv2D $k = 3 \times 3, d = 640$ MP $(p = 3, 3)$
FC $(h = 128)$
FC $(h = 18)$
Legend: Conv2D - 2D Convolutional Layer MP - 2D Max-Pooling FC - Fully-Connected Layer

CRNN Architecture

CRNN
Conv2D $k = 3 \times 3, d = 64$ MP $(p = 2, 2)$
Conv2D $k = 3 \times 3, d = 128$ MP $(p = 2, 2)$
Conv2D $k = 3 \times 3, d = 256$ MP $(p = 2, 2)$
Conv2D $k = 3 \times 3, d = 256$ MP $(p = 2, 2)$
GRU $(h = 256)$
FC $(h = 18)$
Legend: Conv2D - 2D Convolutional Layer MP - 2D Max-Pooling GRU - Gated Recurrent Unit FC - Fully-Connected Layer