IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.102 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 5, May 2024

DOI: 10.17148/IJARCCE.2024.13504

TuneDetect: Musical Mastermind for Musician

Archana Priyadarshini Rao[1], Ashik Raj E[2], Vaishnavi Naik[3], Manasa[4], Arpitha[5]

Assistant Professor, Information Science and Engineering, A J Institute of Engineering and Technology, Mangalore, India[1]

Student, Information Science and Engineering, A J Institute of Engineering and Technology, Mangalore, India[2]Student, Information Science and Engineering, A J Institute of Engineering and Technology, Mangalore, India[3]Student, Information Science and Engineering, A J Institute of Engineering and Technology, Mangalore, India[4]Student, Information Science and Engineering, A J Institute of Engineering and Technology, Mangalore, India[5]

Abstract: In this study, a system that captures the spirit of sound detective work is presented. Similar to a musician, it analyzes the sound by first removing certain elements, recognizing individual instruments, confirming the notes or sounds they produce. The finished product is like to a information of how many instruments are involved in a particular audio file. It can be a fresh score or text, with the detection of pitch feature. It is available for listening, and people can enjoy the music immensely. This project accomplishes a number of things really effectively for listening to polyphonic music performed by multiple instruments at once. The implementation of automated project that allows the users to identify the instruments involved in the audio files and detect the pitches out of the detected instruments. We first dealed with the analyzing of monophonic instruments sound which involves only one instrument and got the accuracy of 70 % for 40 Epoch. While dealing with polyphonic instrument sound there were many complexity faced during the dataset collection and testing phase. From previous survey there is work only done on monophonic instruments .Since there were no work done on polyphonic instruments sound, there was no dataset available. The merging of the monophonic instrument sound taken from different websites were done, almost 30,000 audios were collected .The one audio file contains varying number of instruments merged .Some contains four ,some five instruments. For training it is 2630 and testing 81 audio files. While the user can provide any of audio files from total of 2462 audio files. The accuracy decreased since we are dealing with very complex, overlapping datasets. Re-labelling would provide good results. We got 30% accuracy with 40 Epoch. The detection of Instruments was determined. The Other features includes pitch detection and synchronization. The real-time pitch detection using the Web Audio API and microphone input, along with the ability to analyze pre-recorded audio files is implemented. The synchronization process allows the user to upload an audio file, analyze it, generate synchronized MIDI data, and present the synchronized output back to the user for further interaction or analysis. The goal is to enhance user satisfaction by providing music that aligns with their instrument detection, pitch detection and synchronization.

Keywords: Instrument Detection, Pitch Detection, Synchronization, MIDI Synthesizer, Polyphonic, Monophonic.

I. INTRODUCTION

The project explores the field of polyphonic sound, which is the symphony of different melodies and tones produced by combining several instruments. Its objective is to strip this musical composition of all of its complexity in order to reduce it to a harmonic, comprehensive task. Finding the musical instruments in a given polyphonic audio input is the aim of the current investigation. A monophonic transmission, sometimes referred to as a "mono" signal, consists of a single audio source or line of music. Put another way, it transmits a single audio channel, which often translates to a single voice or instrument. A polyphonic signal, on the other hand, consists of multiple independent audio sources or musical lines playing simultaneously. There are multiple audio channels in these signals, and each one could represent a distinct voice or instrument. Despite years of research on musical instrument identification, no study has been conducted on musical instruments. There are many obstacles in the way of developing an automated system for pitch detection and instrument identification. Learning the instruments for music instruction was a difficult and time-consuming task in the past. Manually hearing the instruments played one by one is time consuming. The work provides a simple and robust method for overcoming some of the main challenges in the past when detecting musical instruments in polyphonic audio data. Nobody has seen any really important study on musical instruments, despite the years of research on musical instrument identification. In order to get over some of the major limitations encountered in the past, the study provides a straightforward and reliable method for identifying musical instruments in polyphonic audio signals. There are numerous applications for a system that can identify an instrument's pitch in polyphonic music. Pitch shifts, frequency modulations,

DOI: 10.17148/IJARCCE.2024.13504

and polyphonic music with many instruments are all supported by the pitched component. The technology includes a "unpitched component" in addition to pitched sounds. With the help of this unpitched component, the model can identify different drum kits, which are known for their non-pitched, percussion-like sounds. Because it involves managing problems with onset detection, note tracking, and temporal alignment, this process can be difficult. One of the most important things that can be done to advance the area is to create synchronization algorithms that can handle different musical nuances and complexities. The field of polyphonic pitch recognition and transcription has seen a great deal of study. Developing an automated system that automatically detects the number of instruments present in the polyphonic instrument sound is wonderous. The system is also able to detects the pitches out of the voice provided by the user. One of the primary challenges in implementing a pitch detection system is minimizing audio processing latency. Real-time pitch detection requires efficient signal processing techniques to analyze audio input without noticeable delay. Optimizing algorithms and minimizing computational overhead are essential to achieve low latency. When there's a lot of background noise, it can mess up our ability to accurately detect pitch. To make sure we're getting the right pitch even in noisy places, we need smart techniques to clean up the signal. It's like trying to listen to someone speaking in a crowded room — we need ways to filter out all the other sounds and focus on what they're saying. By using methods like looking at the frequency content of the sound and filtering out unwanted noise, we can make our pitch detection system work better even in noisy environments. The synchronization process is indeed very difficult task. It includes many challenges like audio files can vary significantly in terms of genre, instrumentation, recording quality, etc. Adapting the synchronization process to handle this variability robustly can be challenging. Audio analysis can be computationally intensive, especially for large audio files or complex analysis tasks. Ensuring efficient processing to handle various file sizes and complexities is essential.The project implements the synchronization process that allows the user to upload an audio file, analyze it, generate synchronized MIDI data, and present the synchronized output back to the user for further interaction or analysis.The MIDI data, which stands for Musical Instrument Digital Interface, is a standardized protocol that enables electronic musical instruments, computers, and other devices to communicate with each other. MIDI data represents musical events, such as notes, pitch, velocity, and timing, in a digital format, allowing for the interchange of musical information between different hardware and software systems. MIDI data is event-based, meaning it represents discrete musical events rather than continuous audio waveforms. These events include note-on and note-off messages, control changes, pitch bend, and others.MIDI messages encode various musical parameters, including pitch (note number), velocity (loudness), duration, and channel (which instrument or voice the event applies to).The goal is to enhance user satisfaction by providing music that aligns with their instrument detection, pitch detection and synchronization.

II. MOTIVATION

It takes a lot of effort to manually listen to and identify different instruments, especially for people who are just starting to play an instrument. Similarly, dealing with polyphonic audio recordings presents many technological problems for music producers. Producers and students can save time and effort by automating the recognition and identification of instruments. Because of this automation, the process is streamlined and musicians are free to concentrate more on creative elements rather than technical details. As a result, creative and emotive musical compositions may result from this. Increasing productivity, encouraging creativity, accelerating learning, and supporting continuous technological developments in the domains of music production and education are the objectives of creating an instrument identification system. Automated pitch detection and synchronization streamline the learning process for musicians, enabling them to progress faster and achieve their musical goals with greater efficiency. When musicians don't have to worry about the technical stuff like identifying instruments or getting everything perfectly synced up, they can focus completely on being creative. This means they can come up with new and exciting music without being held back by the nitty-gritty details. Automated tools make sure everything sounds just right, leaving musicians free to let their emotions shine through in their compositions.

III. OBJECTIVES

Instrument Detection : Creating a model that can efficiently detect the number of instruments present in the both monophonic and polyphonic instruments sound.

Pitch Detection: Using sophisticated pitch identification techniques to precisely determine the pitch of every note played by instruments that have been identified.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.102 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 5, May 2024

DOI: 10.17148/IJARCCE.2024.13504

Synchronization: The system needs to handle issues with harmonics and overlapping frequencies to properly separate and analyse the sounds of each instrument in the musical landscape.

Intuitive user Interface: Providing an intuitive user interface so that people can communicate with the system without any difficulty.

IV. SYSTEM DESIGN

Systems design is the process of defining the architecture, modules, interfaces, and data for a system to satisfy specified requirements

A. Architectural Diagram

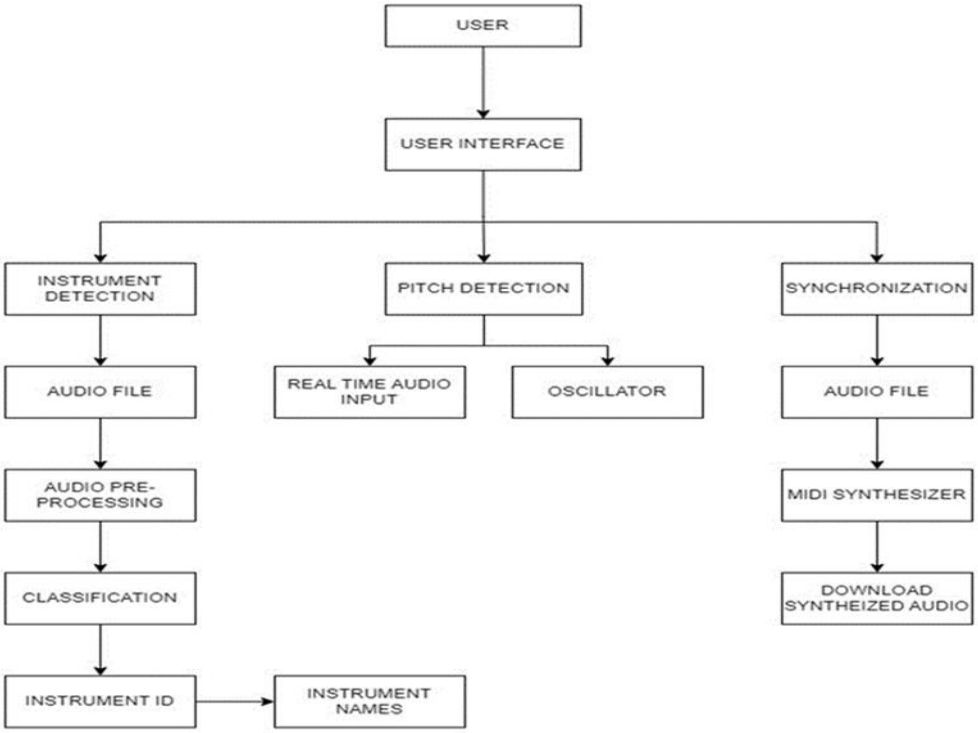

Fig. 1 Architectural Diagram

Above Figure 1 shows the architecture of proposed system Initially, the system receives Polyphonic Instrument audio files via a user interface. In order to extract additional characteristics from the audio signals, the audio signal is preprocessed, which includes segmentation, the removal of noisy data, and the conversion of low dimensional space input into high dimensional space, extract Mel-frequency cepstral coefficients (MFCCs) and spectrograms from audio files. MFCCs and spectrograms are common features used in audio processing tasks. Spectrograms obtained from audio files are converted into images.The transfer learning with the DenseNet201 architecture pre-trained on ImageNet for feature extraction. Data augmentation is applied using ImageDataGenerator from Keras to enhance model generalization. Augmentation techniques such as rotation, Zooming and shifting are utilized. The classification aims at identification of Instruments from the polyphonic instrument sound. The output of the system is the instruments names from both monophonic and polyphonic instruments audio. The system is also able to detect the pitches from real time audio or voice of a person. There are two buttons at the interface which allows the user to record the voice and oscillator option that is able to generates a sound, and when deactivated, it stops generating the sound. The system is able to detect the real time pitch out of the vocals provided. The input audio file that has different frequencies are synchronized and midi data is provided.

B. Flow Chart

A data flow diagram (DFD) is a visual representation of how data flows through a system. It is made up of processes,

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.102 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 5, May 2024

DOI: 10.17148/IJARCCE.2024.13504

data storage, data flows, and external entities. In this Tune detect detection project, a data flow diagram could represent the flow of data from input sources (such as audio / feeds from the interface) through the various preprocessed dataset through the instrument detection ,pitch detection to the output, which includes the instruments names in particular polyphonic instrument file, the detected pitches in score or graph form. External entities may include the user who interacts with the system via the user interface, as well as any external systems or devices that are integrated with Instrument detection system.The data flow diagram would help visualize how data moves through the system, allowing you to better understand its functionality and identify potential areas for optimization or improvement. In conclusion, flowcharts are essential for improving process clarity and comprehension, encouraging efficient communication, and supporting decision-making and problem-solving in a variety of fields. They are a priceless resource for individuals and groups looking to optimize and enhance their operational workflows because of their adaptability and accessibility. provides a clear and visual representation of a process. It promotes understanding and communication amongst team members. Additionally, it locates process bottlenecks and inefficiencies, which helps with process analysis and improvement.

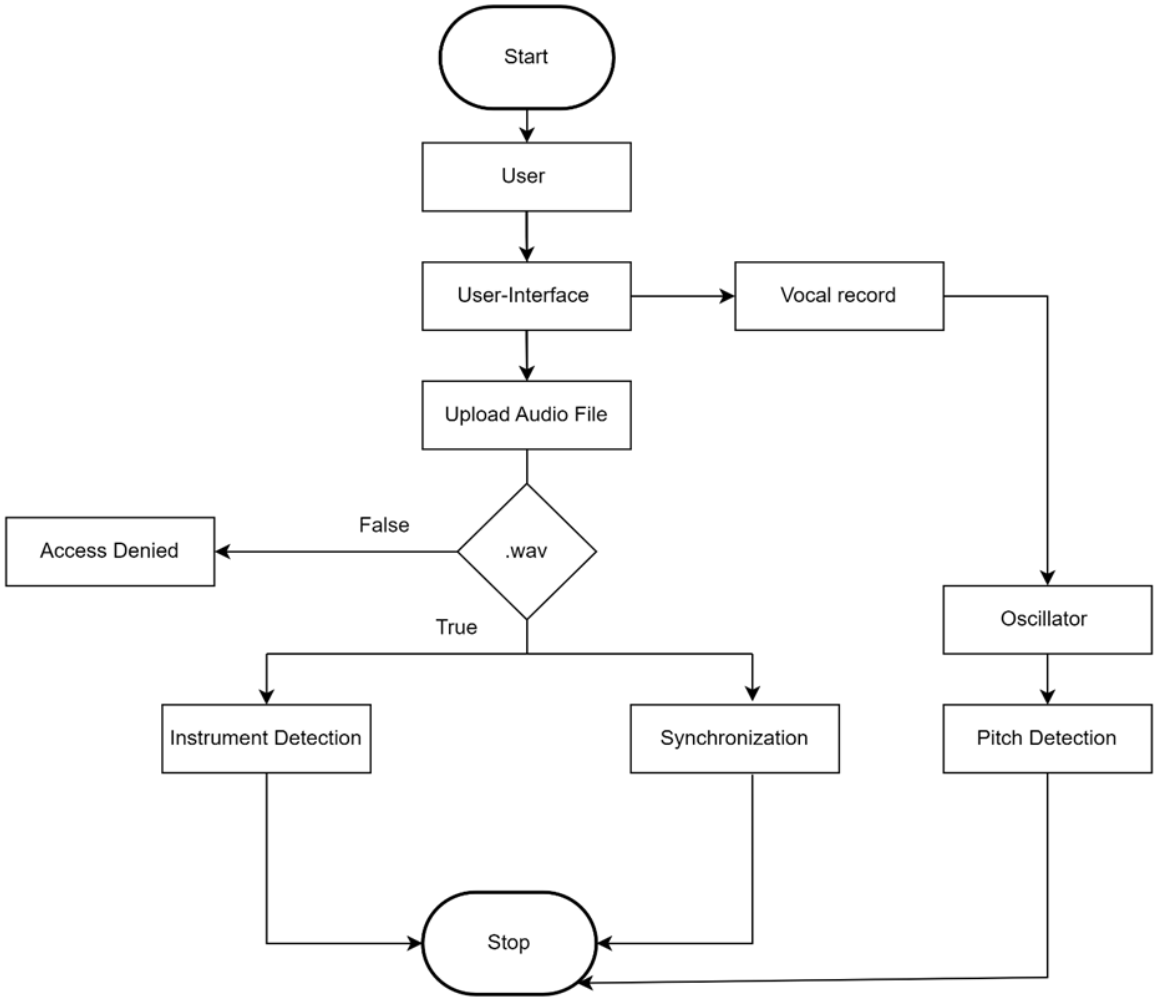

Fig. 2 Flow Chart

Initially the process starts with user interacting with the interface by uploading the monophonic and polyphonic audio files from the system. The audio file is provided from the user to the system once the upload option is clicked. After the audio file should be of format.wav. The other format is not supported. The message is sent back to user as access denied. After the audio is uploaded ,then clicking on to the classify option redirects to the detection process. From the audio file provided the instrument is detected .The number of instruments present in the audio files are detected and the names are

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.102 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 5, May 2024

DOI: 10.17148/IJARCCE.2024.13504

provided also the audio is played. There is integration of another user interface for pitch detection.The user can input live voice and the system is able to detect the pitch out of the vocals.There is stop and start button for the starting and ending live audio input process.The user is also given with the option called oscillator , this button toggles the oscillator on and off. When activated, it generates a sound, and when deactivated, it stops generating the sound.The system is able to detect the real time pitch out of the vocals provided.The synchronization process allows the user to upload an audio file, analyze it, generate synchronized MIDI data, and present the synchronized output back to the user for further interaction or analysis. The goal is to enhance user satisfaction by providing music that aligns with their instrument detection, pitch detection and synchronization.

C. Use Case

A use case diagram depicts the interactions between actors (users or external systems) and the system being considered. It demonstrates the various ways in which users can interact with the system to accomplish specific goals or tasks. A use case diagram for the Tune Detect: instrument and pitch detection project could depict the system's various functionalities or features, as well as how different actors interact with them. A use case diagram is a visual representation that depicts the interactions between actors (users or external systems) and a system under consideration, showcasing the various functionalities or behaviors that the system offers. It provides a high-level overview of the system's features and how users interact with them, helping stakeholders understand the system's purpose and scope. Use case diagrams are valuable tools in requirements analysis, system design, and communication among stakeholders, facilitating discussions about system.

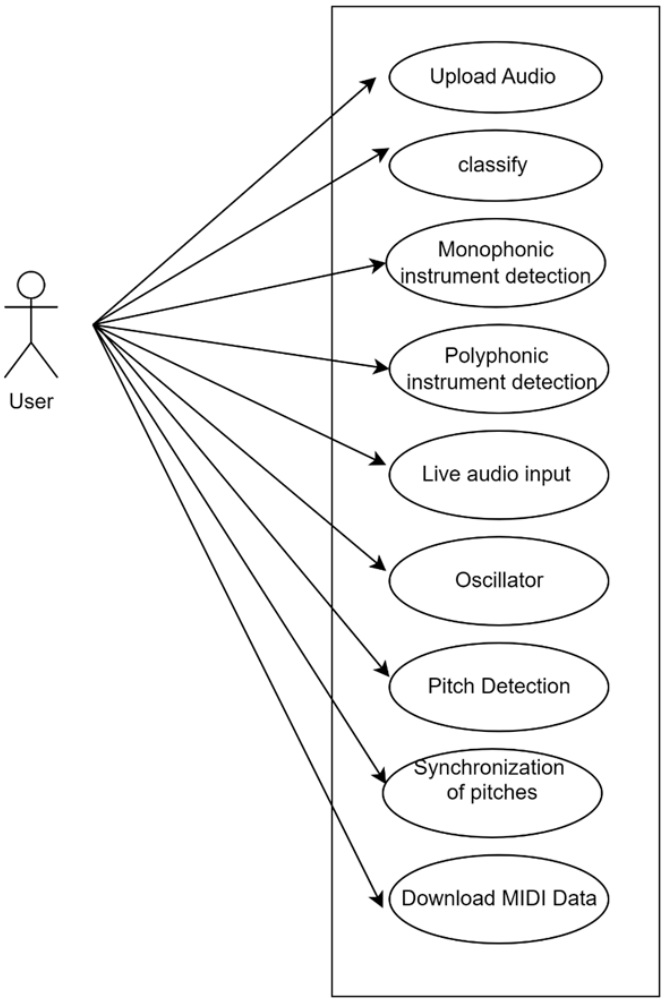

Fig. 3 Use Case

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.102 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 5, May 2024

DOI: 10.17148/IJARCCE.2024.13504

Figure 3 shows the use case diagram for the project The system provides the service to the user .The user is able to interact with the system through intuitive user interface where the user can upload the monophonic or polyphonic audio files from the system and click on to classify for the classification process. Once the classification is done the system provides the names of all the instruments present in the audio files and the user also can deal with real time pitch detection by inputing their voice by oning microphone They can start and stop the process and also use oscillator option that is able to generates a sound, and when deactivated, it stops generating the sound. The system is able to detect the real time pitch out of the vocals provided. The system detects the pitches out of the audio provided. The systems allows the user to start and stop record accordingly. It also provides the option of oscillator and user can hear the oscillating sound. The synchronization process allows the user to upload an audio file, analyze it, generate synchronized MIDI data, and present the synchronized output back to the user for further interaction or analysis. The goal is to enhance user satisfaction by providing music that aligns with their instrument detection, pitch detection and synchronization.

V. IMPLEMENTATION

The execution of the "Tune Detect" project integrates advanced technologies like Convolutional Neural Networks (CNNs), signal processing, and user-friendly interfaces. This integration enables musicians to efficiently analyze, tune, and synchronize audio recordings. The system's core functionality includes instrument detection, pitch analysis, and precise synchronization, empowering musicians to refine performances with precision. Moreover, an intuitive user interface facilitates seamless interaction, enabling users to upload recordings and visualize analysis results effortlessly. Administrators access a comprehensive interface to manage the database, organize recordings, and facilitate collaboration. This integration ensures a streamlined and effective music analysis and synchronization experience for musicians.

Code Implemented

```
import os
from tqdm import tqdm
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.io import wavfile
import librosa
import librosa.display
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense,Dropout,Activation
from tensorflow.keras.optimizers import Adam
from basic_pitch.inference import predict
from basic_pitch import ICASSP_2022_MODEL_PATH
```

Fig. 4 Importing Required Packages

Figure 4 depicts the setup of a project focused on musical instrument classification and synchronization. The Python code establishes a neural network model using TensorFlow and Keras for instrument classification. Key libraries are imported for numerical computations, data manipulation, and building neural networks, alongside the 'basic_pitch' library for synchronization purposes. The model architecture comprises sequential layers, including convolutional layers for feature extraction, pooling layers for dimensionality reduction, and dense layers for classification. Following compilation with appropriate loss functions, optimizers, and metrics, the model is trained on prepared training data. Additionally, data augmentation techniques may be applied to diversify training samples. Subsequently, the model's performance is evaluated on testing data to assess its accuracy in classifying musical instruments, while synchronization tasks are handled using the 'basic_pitch' library.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.102 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 5, May 2024

DOI: 10.17148/IJARCCE.2024.13504

```
test=[]
test_labels=[]
for i in range(len(test_df)):
    try:
        label=test_df.loc[i,'Class']
        file=test_df.loc[i,'FileName']
        path=os.path.join(test_dir,file)
        y, sr = librosa.load(path)
        mel_spectrogram = librosa.feature.melspectrogram(y=y, sr=sr, n_mels=128)
        log_mel_spectrogram = librosa.power_to_db(mel_spectrogram, ref=np.max)
        img=log_mel_spectrogram
        img=cv2.resize(np.array(img),dsize=(128,128))
        X=cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        #X = (X - X.min(axis=0)) / (X.max(axis=0) - X.min(axis=0))
        test+=[X]
        test_labels+=[tnormal_mapping[label]]
    except:
        print('except',i,file)
        continue
```

Fig. 5 Pre-Processing Dataset Images

Figure 5 illustrates the supplied code excerpt, which orchestrates the preprocessing and feature extraction of audio data for machine learning endeavors. It meticulously navigates through each entry in the dataset, diligently loading the audio files via Librosa and processing them to compute mel-spectrograms. These spectrograms undergo logarithmic scaling to refine their perceptual relevance. Subsequently, the images are meticulously resized to a uniform 128x128 pixel size and adeptly converted to RGB format. The processed spectrograms are then systematically appended to a list alongside their corresponding numerical labels. The code demonstrates astute error handling, gracefully managing any encountered issues during the preprocessing routine. This methodical approach lays the groundwork for robust feature extraction, thereby facilitating subsequent machine learning tasks with audio data.

```
datagen = ImageDataGenerator(
            horizontal_flip=False,vertical_flip=False,
            rotation_range=0,zoom_range=0,
            width_shift_range=0.2,height_shift_range=0.2,
            shear_range=0.1,fill_mode="nearest")
```

Fig. 6 Splitting dataset and training

Figure 6 presents the code snippet which instantiates an ImageDataGenerator class from the Keras library, a pivotal tool for augmenting image data in deep learning applications. The instantiation encompasses an array of augmentation parameters meticulously tailored to refine the training data. These parameters encompass options like horizontal and vertical flips, rotation, zoom, and shift ranges, fine-tuning the extent of random transformations imparted to images during training. By harnessing these augmentation methodologies, the model fortifies its resilience and mitigates the risk of overfitting, culminating in heightened performance across image classification tasks. The schematic depicts a harmonious synergy between the user and the system, facilitating an adaptive and nimble music recommendation process.

```
X_train,X_test,y_train,y_test=
            train_test_split(X,y,test_size=0.2,random_state=42)
```

Fig. 7 Splitting dataset

Figure 7 illustrates the provided code snippet, showcasing the process of partitioning a dataset into training and testing subsets utilizing the train_test_split function from the scikit-learn library. By designating a test size of 0.2, 20% of the data is segregated for testing purposes, while the remaining 80% is allocated for training. The incorporation of a specific random state, set to 42, ensures reproducibility of the split across different executions. It is imperative to acknowledge

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.102 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 5, May 2024

DOI: 10.17148/IJARCCE.2024.13504

that this code snippet exclusively handles the division of data and does not encompass subsequent stages of model training, hyperparameter optimization, or model evaluation, all of which constitute integral facets of the machine learning workflow. Furthermore, it's worth noting that a prevalent practice in machine learning involves further partitioning the data into training, validation, and testing subsets to effectively train, validate, and assess the generalization capability of the model on unseen data.

```
model=Sequential()
model.add(Dense(100,input_shape=(40,)))
model.add(Activation('relu'))
model.add(Dropout(0.3))

model.add(Dense(200))
model.add(Activation('relu'))
model.add(Dropout(0.3))

model.add(Dense(200))
model.add(Activation('relu'))
model.add(Dropout(0.3))

model.add(Dense(100))
model.add(Activation('relu'))
model.add(Dropout(0.3))

model.add(Dense(num_labels))
model.add(Activation('softmax'))
```

Fig. 8 Creating the Model

Figure 8 showcases the code segment dedicated to constructing a neural network model using the Sequential API from Keras. The model architecture comprises multiple dense layers interconnected sequentially. Each layer is meticulously designed to process the input data and extract relevant features essential for effective learning. The incorporation of activation functions, such as ReLU (Rectified Linear Unit), facilitates non-linearity within the model, enhancing its capability to capture complex patterns in the data. Furthermore, dropout layers are strategically inserted to mitigate the risk of overfitting by randomly deactivating a fraction of neurons during training. This regularization technique aids in generalizing the model's learned representations to unseen data. The final layer employs the softmax activation function to produce output probabilities across multiple classes, making the model suitable for classification tasks.

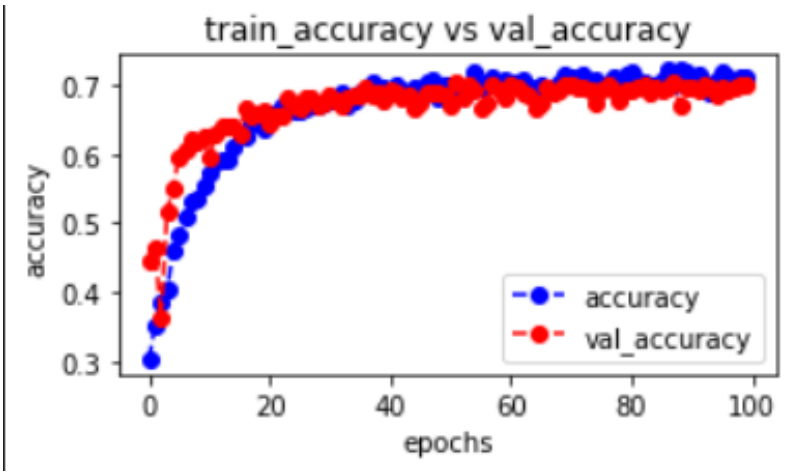

Fig. 9 Accuracy Graph

Figure 9 shows the graph illustrates the training and validation accuracy of a machine learning model over multiple epochs. The training accuracy steadily increases with each epoch, indicating that the model is effectively learning from the training data. Meanwhile, the validation accuracy, though slightly lower than the training accuracy, follows a similar increasing trend, indicating that the model is generalizing well to unseen data. The relatively close alignment of the two

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.102 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 5, May 2024

DOI: 10.17148/IJARCCE.2024.13504

curves suggests that the model is performing robustly without overfitting, achieving high accuracy on both the training and validation datasets.

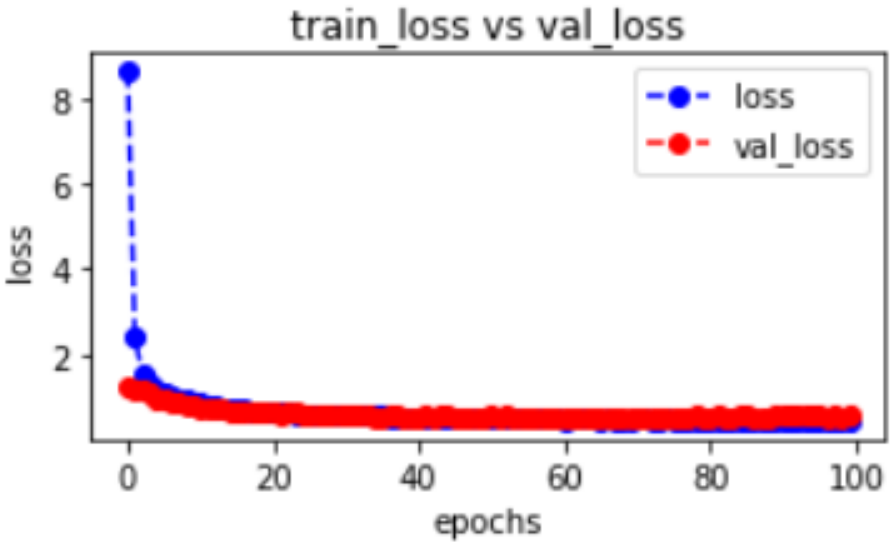

Fig 10 Training and Validation Loss

Figure 10 shows the line graph shows how well a machine learning model is learning from the data by comparing training loss and validation loss. Training loss measures how well the model performs on the data it is learning from, while validation loss shows how well the model performs on separate, unseen data. The graph's x-axis represents the number of times the model goes through the training data (epochs), and the y-axis shows the loss values. Ideally, both losses should decrease over time, but if validation loss starts increasing, it suggests the model is overfitting—it is learning the training data too closely and not generalizing well to new data. To simplify the model, add more training data, or use data augmentation to make the model more robust.

```
function toggleLiveInput() {
    if (isPlaying) {
        //stop playing and return
        sourceNode.stop( 0 );
        sourceNode = null;
        analyser = null;
        isPlaying = false;
        if (!window.cancelAnimationFrame)
            window.cancelAnimationFrame = window.webkitCancelAnimationFrame;
        window.cancelAnimationFrame( rafID );
    }
    getUserMedia(
        {
            "audio": {
                "mandatory": {
                    "googEchoCancellation": "false",
                    "googAutoGainControl": "false",
                    "googNoiseSuppression": "false",
                    "googHighpassFilter": "false"
                },
                "optional": []
            },
        }, gotStream);
}
```

Fig 11 Pitch Detection for Live Input

In Figure 11, the JavaScript code snippet delineates the implementation of the toggleLiveInput function, aimed at managing live audio input toggling within a web application. The function commences by evaluating the current state of audio playback (isPlaying). If audio playback is ongoing, the function halts the playback, releasing associated resources

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.102 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 5, May 2024

DOI: 10.17148/IJARCCE.2024.13504

and terminating the audio processing. This includes stopping the audio source node (sourceNode), nullifying the analyser node (analyser), and setting the isPlaying flag to false. Additionally, it checks for browser compatibility with the cancelAnimationFrame function and invokes it to cancel any pending animation frame requests (rafID). Upon completion of these steps, the function awaits a user's consent to access audio input via the getUserMedia method. The method is configured with specific audio constraints, such as disabling echo cancellation, automatic gain control, noise suppression, and high-pass filtering. Upon successfully obtaining user permission, the gotStream callback function is invoked to handle the incoming audio stream.

```
def uploadmusic():
    if 'file' not in request.files:
        return redirect(request.url)

    file = request.files['file']

    if file.filename == '':
        return redirect(request.url)

    if file:
        file_path = os.path.join('static', file.filename)
        file.save(file_path)

        # Call the predict function
        model_output, midi_data, note_events = predict(file_path)

        # Save the MIDI data
        output_midi_path = os.path.join('static', 'output_midi.mid')
        midi_data.write(output_midi_path)

        return render_template('sync.html', midi_file=output_midi_path)

    return redirect(request.url)
```

Fig 12 Synchronization

Figure 12 showcases the Python function uploadmusic(), which is crafted to manage music file uploads within a web application. The function initiates by verifying the presence of a file in the request. If none is found, it redirects the user to the current URL to prompt for file upload. Upon receiving a file, it ensures that the filename is not empty. Subsequently, the uploaded file is stored in a designated directory within the application for further processing. The function then triggers a predictive model to generate outputs based on the uploaded music file. These outputs typically include model predictions, MIDI data, and note events. The MIDI data produced by the model is then saved to another file within the application directory. Finally, the function renders an HTML template (sync.html), passing the path to the generated MIDI file for display or additional interaction on the user interface

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.102 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 5, May 2024

DOI: 10.17148/IJARCCE.2024.13504

VI. RESULTS

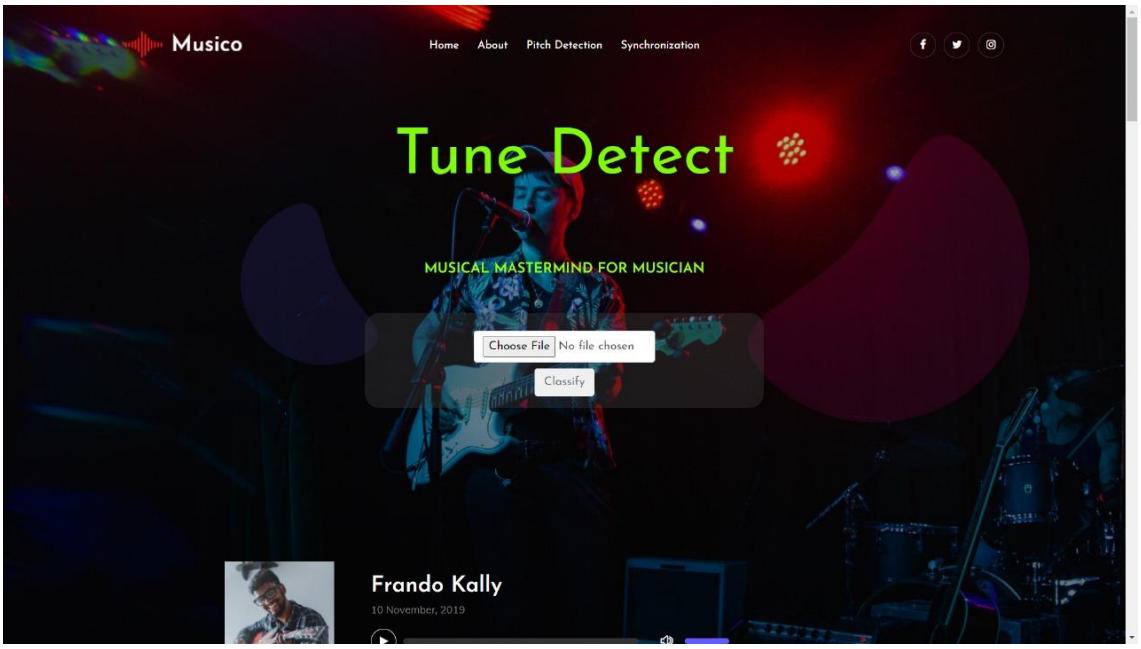

Figure 13:Instrument Detection

Figure 13 shows how through the user interface the user can interact with the system by uploading and classifying functions that allows the user to detect the instrument present it.

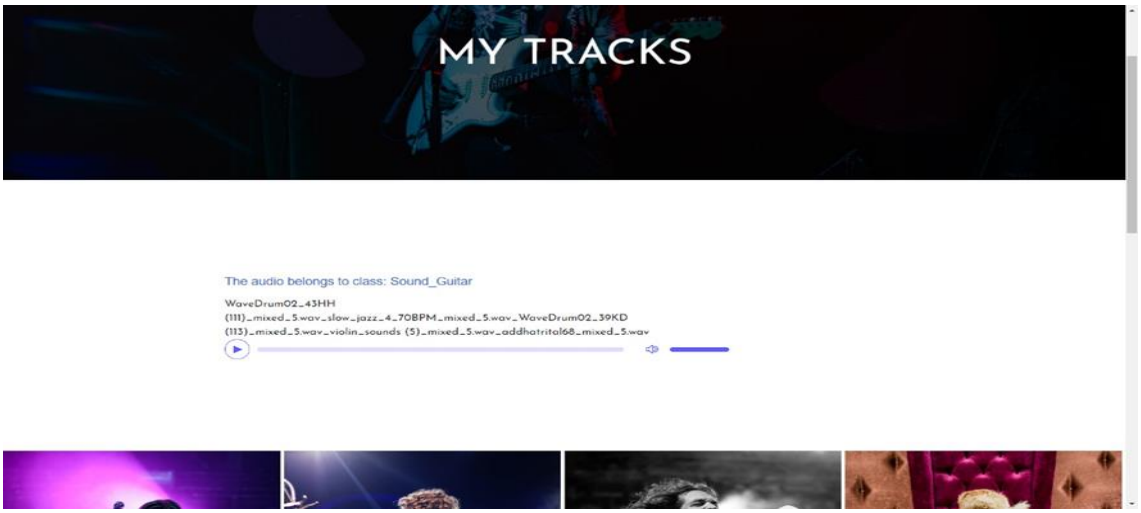

Figure 14: UserInterface for Monophonic instrument Detection

Figure 14 shows how through the user interface the user can interact with the system by uploading and classifying functions that allows the user to detect the instrument present it

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.102 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 5, May 2024

DOI: 10.17148/IJARCCE.2024.13504

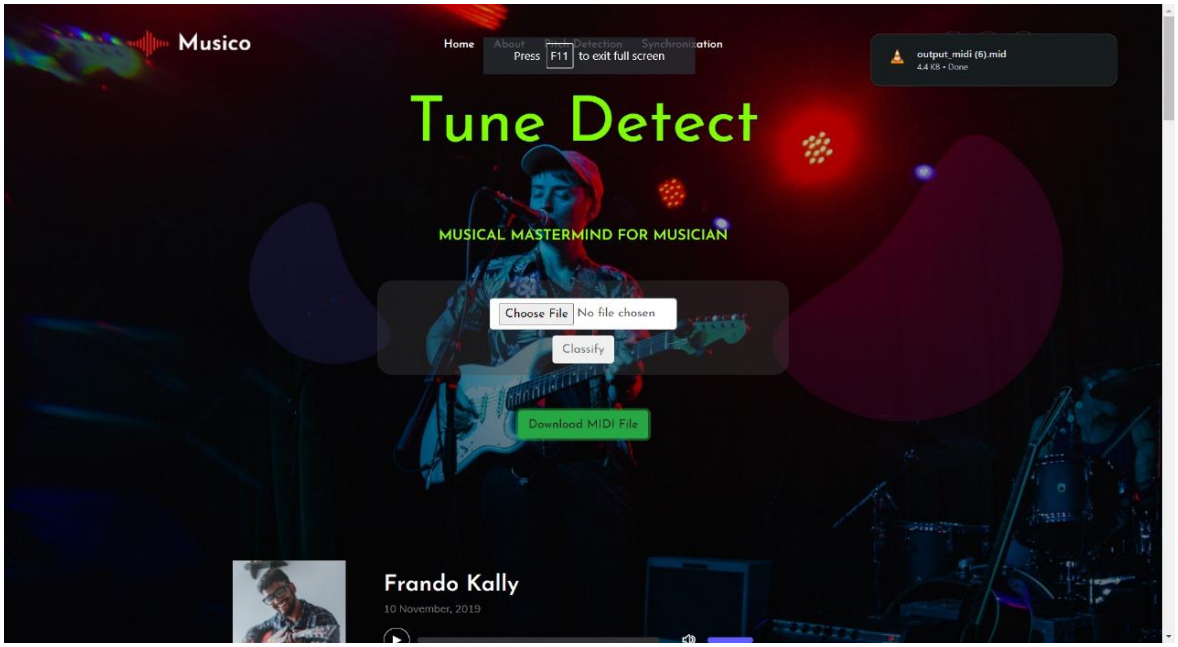

Figure 15: User Interface for polyphonic instrument Detection

Figure 15 shows how through the user interface the user can interact with the system by uploading and classifying functions that allows the user to detect the polyphonic instrument present it.

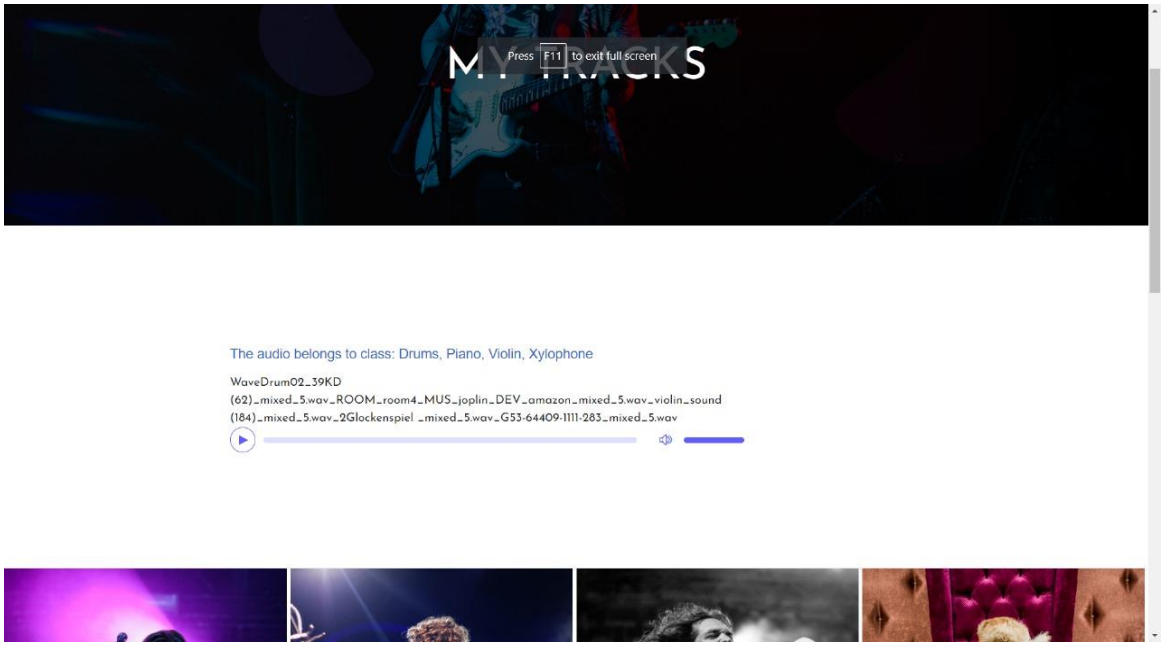

Figure 16:Polyphonic Instrument Detection

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.102 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 5, May 2024

DOI: 10.17148/IJARCCE.2024.13504

Figure 16 shows how through the user interface the user can interact with the system by uploading and classifying functions that allows the user to detect the polyphonic instrument present it.

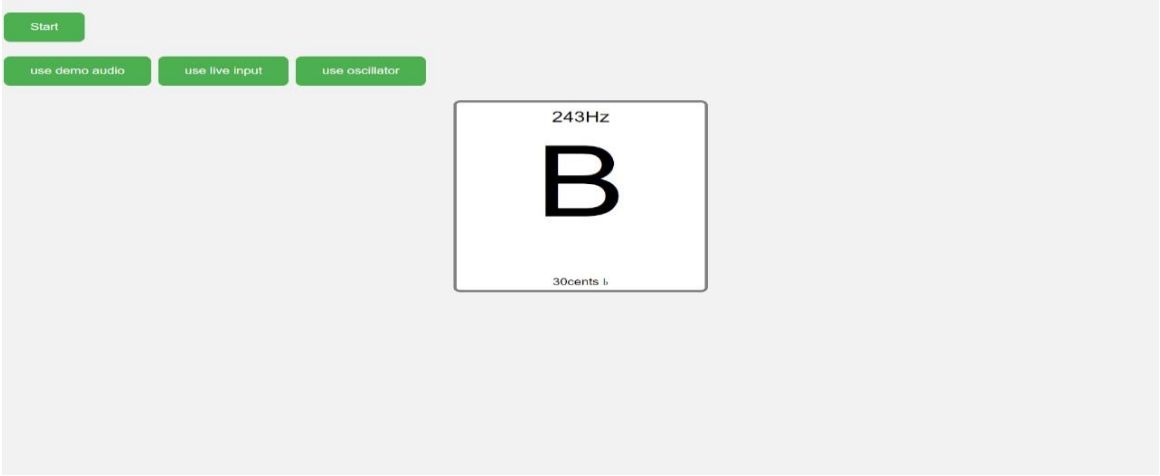

Figure 17: Real-time Pitch Detection

Figure 17 shows how the real-time Pitch Detection. The user can input their vocal using microphone and the system detects the pitches of their voice.

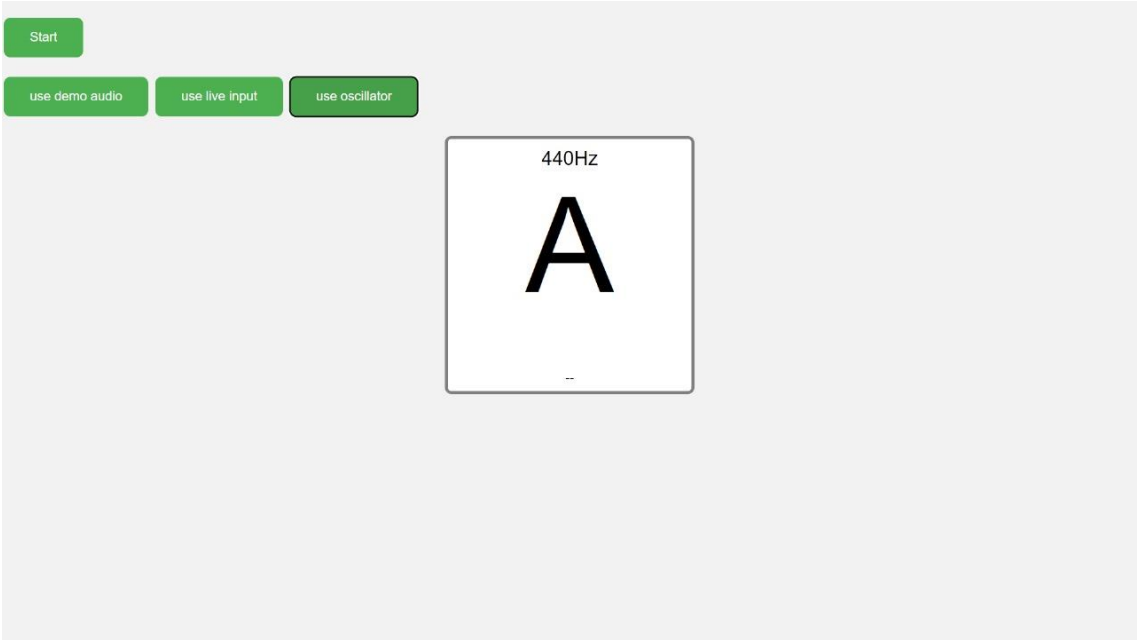

Figure 18: The Oscillator

Figure 18 shows how the oscillator feature works. The standard Ocillating sound of 440 Hz is heard which is used for the tuning of Musical Instruments.

VII. CONCLUSION

In conclusion, "Tune Detect: Musical Mastermind for Musicians" marks a significant milestone in leveraging artificial intelligence and machine learning to transform the music analysis and synchronization process. By seamlessly integrating instrument detection, pitch detection, and audio synchronization technologies, the project has successfully developed a comprehensive platform where musicians can efficiently analyze, tune, and synchronize their audio recordings. The utilization of advanced machine learning techniques, such as Convolutional Neural Networks (CNNs) and signal processing algorithms, ensures accurate instrument detection and precise pitch analysis, enabling musicians to fine-tune their performances with precision. Moreover, the project's user-centric design and iterative development process,

International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.102 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 5, May 2024
DOI: 10.17148/IJARCCE.2024.13504

informed by user feedback and industry insights, position Tune Detect as a forward-thinking solution in the realm of music production and editing tools. Its commitment to providing musicians with intuitive and powerful tools for audio analysis and synchronization has the potential to revolutionize the way musicians approach their craft.

Furthermore, the project's integration of user-friendly interfaces and real-time feedback mechanisms enhances usability and promotes seamless workflow integration for musicians. The availability of a dedicated mobile application expands accessibility, allowing musicians to utilize Tune Detect's features on the go, whether in the studio or on stage. Looking ahead, Tune Detect holds immense potential for future enhancements, including the implementation of advanced audio processing algorithms, integration of collaborative features for remote music production, and continued refinement of synchronization techniques. By embracing innovation and continuously evolving to meet the evolving needs of musicians, Tune Detect is poised to redefine the landscape of music production tools, offering musicians a comprehensive and intuitive platform for audio analysis and synchronization tailored to their creative needs.

REFERENCES

[1] Music Instrument Identification Using MFCC 2003 by Chih-Wen Weng, Cheng- Yuan Lin, Jyh-Shing Roger Jang.
[2] Music Instrument Identification Using MFCC 2017, India by Monica S. Nagawade and Varsha R. Ratnaparkhe
[3] Musical Instrument Classification using Support Vector Machine, Multi Layer Perceptron, and AdaBoost Classifiers 2017 by Swati D. Patil and Priti S. Sanjekar.
[4 Indian Instrument Identification from Polyphonic Audio using KNN Classifier 2019 by S.V .Chandan, Mohan R. Naik, Ashwini, and A. Vijay Krishna.
[5] Instrument Recognition In Polyphonic Music, 2005 by Slim ESSID, Gael RICHARD, and Bertrand .
[6] Pitch Detection In Polyphonic Music Using Instrument Tone Models 2007 by Yipeng Li and DeLiang Wang www.ijcrt.org © 20XX IJCRT | Volume X, Issue X Month Year | ISSN: 2320-2882 IJCRT1601009 International Journal of Creative Research Thoughts (IJCRT) www.ijcrt.org 7
[7] Automatic Transcription of Real-World Music Using Probabilistic Note Event Modeling 2005 by Matti P. Ryynanen and Anssi Klapuri.
[8] Pitch Detection Algorithm: Autocorrelation Method and AMDF ,2014 by Li Tan and Montri Karnjanadecha.
[9] Robust Feature Extraction Using Kernel PCA 2006 by Tetsuya Takiguchi and QYasuo Ariki
[10 Automatic Transcription Of Pitched And Unpitched Sounds From Polyphonic Music 2014 by Emmanouil Benetos, Sebastian Ewert, and Tillman Weyde
[11] Implementation of Pitch Detection Algorithms for Pathological Voices 2016 by Karishma Kolhatkar, Mahesh Kolte, and Jyoti Lele.
[12 Proposed combination of PCA and MFCC feature extraction in speech recognition system 2014 by Hoang Trang, Tran Hoang Loc, and Huynh Bui Hoang Nam.
[13] A Novel Approach for MFCC Feature Extraction 2010 by Md. Afzal Hossan, Sheeraz Memon, and Mark A Gregory
[14] Detecting Pitch Of Singing Voice In Polyphonic Audio 2005 by Yipeng Li and DeLiang Wang.
[15] Efficient Pitch Detection Techniques for Interactive Music 2001 by Patricio de la Cuadra, Aaron Master, Craig Sapp.
[16] Real Time Speech Classification and Pitch Detection 2000 by Jonathan a. Marks.
[17] A polyphonic pitch tracking embedded system for rapid instruments augmentation" 2018 by Rodrigo Schramm, Federico Visi, André Brasil, Marcelo Johann.
[18] Direct Magnitude Spectrum Analysis Algorithm for Tone Identification in Polyphonic Music Transcription 2015 by Marek Boháč, Jan Nouza.
[19] Extracting melody lines from complex audio2004 by Jana Eggink &amp Guy .J.Brown.
[20] RWC Music Database: Music Genre Database and Musical Instrument Sound" 2003 by Masataka Goto, Hiroki Hashiguchi, Takuichi Nishimura, Ryuichi Oka.
[21] Transcription and Separation of Drum Signals From Polyphonic Music 2008 by Olivier Gillet.
[22] Two-pitch tracking in co-channel speech using modified group delay functions 2017 by Rajeev Rajan , Hema A. Murthy.

