SIGNALS AND SYSTEMS

AUDIO GRPAHS AND SPLICING DETECTION

PROJECT REPORT

Group members

Name	CMS
Muhammad Ahmed Mohsin	**333060**
Tariq Umer	**334943**
Hassan Rizwan	**335753**

1 TABLE OF CONTENTS

1 Introduction: 3
2 Flowchart: 4
3 Audio Spectrogram: 4
4 Fourier Transform: 5
5 FFT of Spectrogram: 6
6 Analyzing Different pieces of Spectrogram: 8
7 Using Convolutional Neural Networks pre trained models for Detection: 10
8 Adding Different Sub Classes: 11
9 Using MATLAB Audio Toolbox: 12
10 Enhancing Audio: 7
11 Detecting Splices: 14
12 Problems Faced: 15
13 Conclusion: 15
14 References: 15

2 TABLE OF FIGURES:

Figure 1 5
Figure 2 6
Figure 3 7
Figure 4 8
Figure 5 9
Figure 6 9
Figure 7 11
Figure 8 13
Figure 9 13

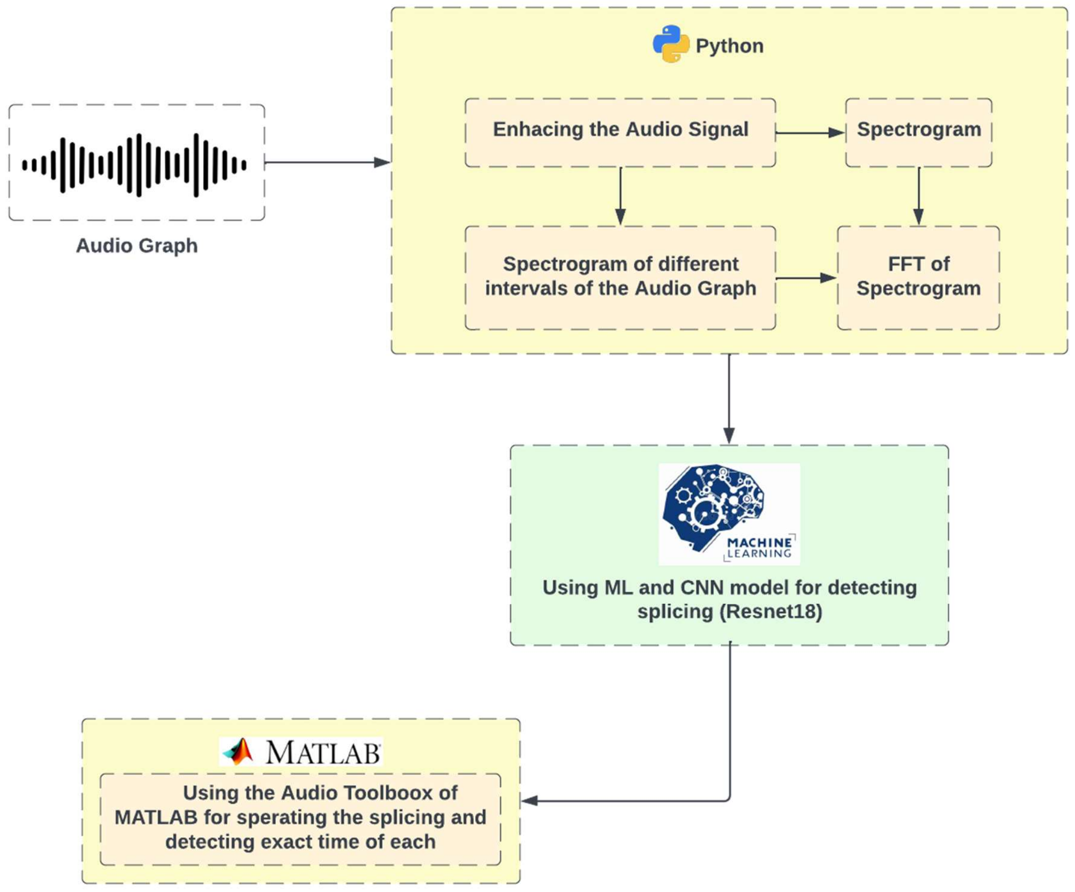
1 INTRODUCTION:

Audio Splicing is a technique of attaching different pieces of audio together to create a new edited audio. Fake audios have become a trend now, and with the increasing expertise of the editors it has become impossible to differentiate between fake and real audio without the use of proper tools. Also, in the field of Forensics Science, audio forensics is done to validate if a piece of audio is admissible in courts or other official venue. To validate audios different techniques has been used, one of them is using Fourier Transform to detect any splicing. In this project, for detecting splicing in each audio python libraries have been used to analyse the audio, the Machine Learning model have been run to classify different sections of the audio based on their frequencies. Then Fast Fourier Transform was taken, and spectrogram of the audio signal was made to further analyse the audio. And finally, audio toolbox of MATLAB was used to detect splicing in the audio, the number of splices combined, and their duration was recorded. The Machine learning library used with pre trained model was **opensoundscape (pytorch models).** The pretrained model was RESNET 18.

PyTorch

2 FLOWCHART:

The flowchart of our workflow is given as:


Audio Graph
Python
Enhacing the Audio Signal
Spectrogram
Spectrogram of different intervals of the Audio Graph
FFT of Spectrogram
MACHINE LEARNING
Using ML and CNN model for detecting splicing (Resnet18)
MATLAB
Using the Audio Toolboox of MATLAB for sperating the splicing and detecting exact time of each

3 AUDIO SPECTROGRAM:

We made audio spectrogram of our given audio signal by using python libraries such as **matplotlib, keras etc.** The code for the spectrogram of our audio signal in python 3.11.1 is given as:

```
# for data transformation
import numpy as np
# for visualizing the data
import matplotlib.pyplot as plt
```

```
# for opening the media file
import scipy.io.wavfile as wavfile
Fs, aud = wavfile.read('/content/Audio Semester Project.wav')
aud = aud[:,0] # select left channel only
first = aud[:int(Fs*43)] # trim the first 43 seconds

powerSpectrum, frequenciesFound, time, imageAxis = plt.specgram(first,
Fs=Fs)
plt.show()
```

The spectrogram for our audio signal obtained was as:

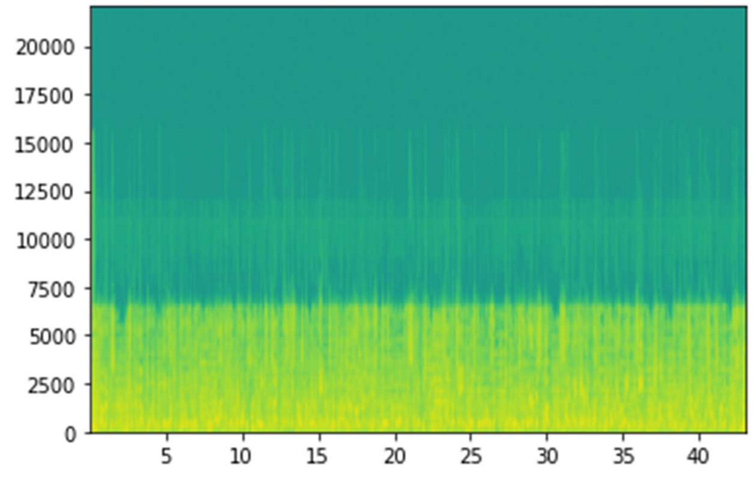


Figure 1

4 Fourier Transform:

Audio signals are composed of multiple single-frequency sound waves. When these waves are recorded, only the amplitudes captured. The amplitudes graphs do not give much information about the signal. Thus, a mathematical technique, Fourier Transformation is used to decompose the signal into its constituent frequencies. This new graph represents the amplitude of every frequency present in the signal.

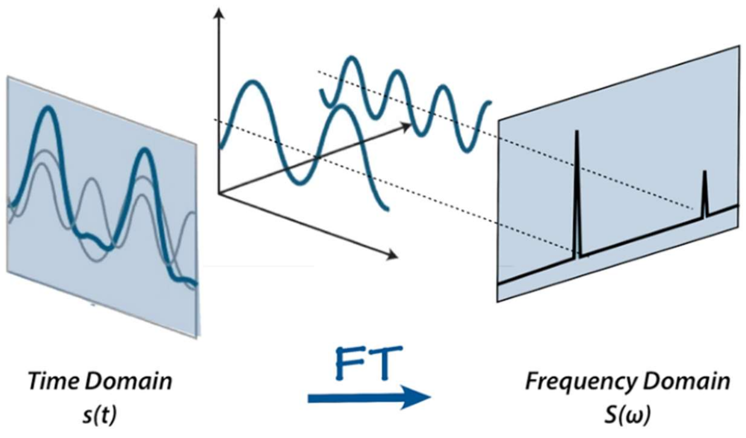


Figure 2

5 FFT OF SPECTROGRAM:

It is a mathematical algorithm that calculates the Discrete Fourier Transform (DFT) of a given signal. The difference between FT and FFT is that FT takes a continuous signal as input while FFT takes a discrete. Just like FT, FFT also converts a signal from time domain to frequency domain. The FFT for our audio signal is given as:

We applied Fourier transform on our audio signal as the code in python is given as:

```
print(f"How many samples does this audio object have? {len(audio_object
.samples)}")
print(f"What is the sampling rate? {audio_object.sample_rate}")
audio_object = Audio.from_file(audio_filename)
# calculate the fft
fft_spectrum, frequencies = trimmed.spectrum()

#plot settings
from matplotlib import pyplot as plt
plt.rcParams['figure.figsize']=[15,5] #for big visuals
%config InlineBackend.figure_format = 'retina'

# plot
plt.plot(frequencies,fft_spectrum)
plt.ylabel('Fast Fourier Transform (V**2/Hz)')
plt.xlabel('Frequency (Hz)')
```

The spectrogram obtained from the code is shown as:

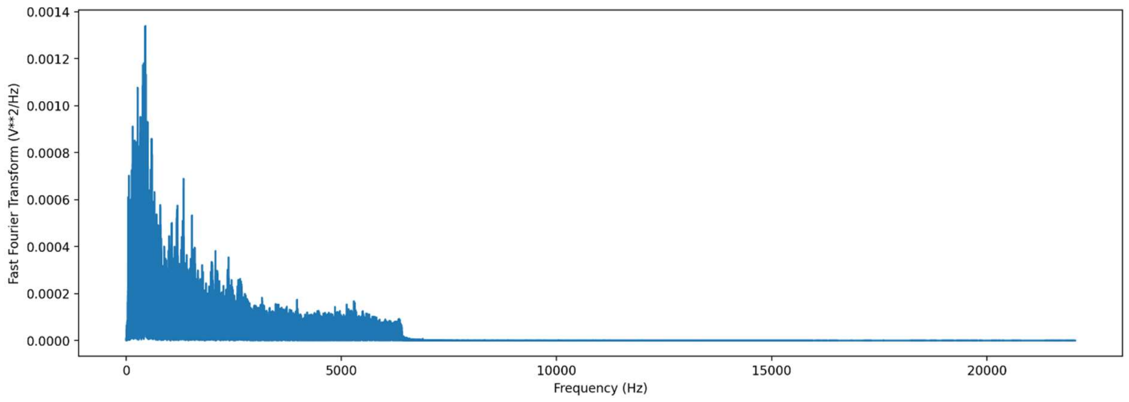
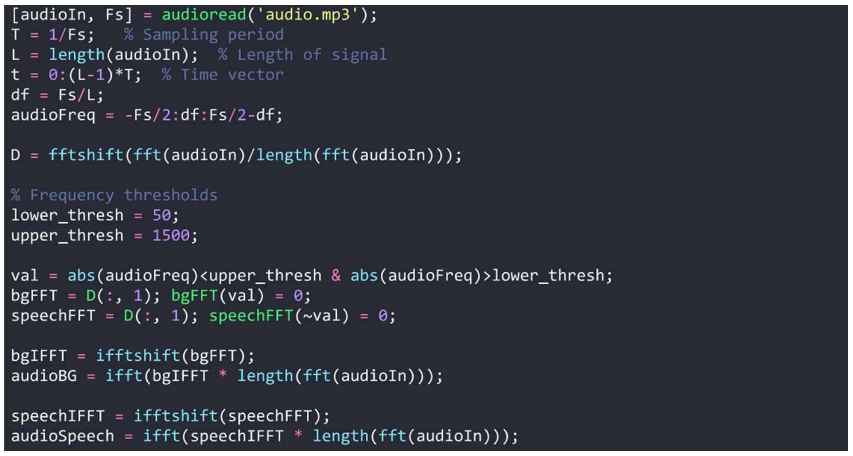


Figure 3

This plot is showing strong frequencies in the range of 0-1000Hz. This is the typical range of frequencies for human voice. This indicate that our audio has a human voice in it. We can also see frequencies on the range of 1000-6000Hz, these must be the frequencies of our background music in the audio.

6 ENHANCING AUDIO:

We enhanced the audio by using audio pre-processing libraries present in MATLAB. The background noise and music was removed in order to filter out the false effects of the background noise and detect splicing more effectively. The code for MATLAB pre-processing is as shown:



```
[audioIn, Fs] = audioread('audio.mp3');
T = 1/Fs;   % Sampling period
L = length(audioIn);  % Length of signal
t = 0:(L-1)*T;  % Time vector
df = Fs/L;
audioFreq = -Fs/2:df:Fs/2-df;

D = fftshift(fft(audioIn)/length(fft(audioIn)));

% Frequency thresholds
lower_thresh = 50;
upper_thresh = 1500;

val = abs(audioFreq)<upper_thresh & abs(audioFreq)>lower_thresh;
bgFFT = D(:, 1); bgFFT(val) = 0;
speechFFT = D(:, 1); speechFFT(~val) = 0;

bgIFFT = ifftshift(bgFFT);
audioBG = ifft(bgIFFT * length(fft(audioIn)));

speechIFFT = ifftshift(speechFFT);
audioSpeech = ifft(speechIFFT * length(fft(audioIn)));
```

7 ANALYZING DIFFERENT PIECES OF SPECTROGRAM:

While taking FFT of the audio signal we lost task of the time information of the signal. i.e., we cannot that what was spoken first. Thus, to see the frequency value along with time, another type of graph is used, that is called spectrogram.

In spectrogram, x-axis represent the time and the y-axis represent the frequencies, whereas the colour represents the magnitude (amplitude) of the observed frequency at that time. The code for spectrogram for our signal in python is given as:

```
audio_object = Audio.from_file(audio_filename)
spectrogram_object = Spectrogram.from_audio(audio_object)
spec = Spectrogram.from_audio(Audio.from_file(audio_filename))
print(f'All times: {spec.times[0:40]}')
print(f'All frequencies frequencies: {spec.frequencies[0:40]}')
audio_object = Audio.from_file(audio_filename)
spectrogram_object = Spectrogram.from_audio(audio_object)
spectrogram_object.plot()
```

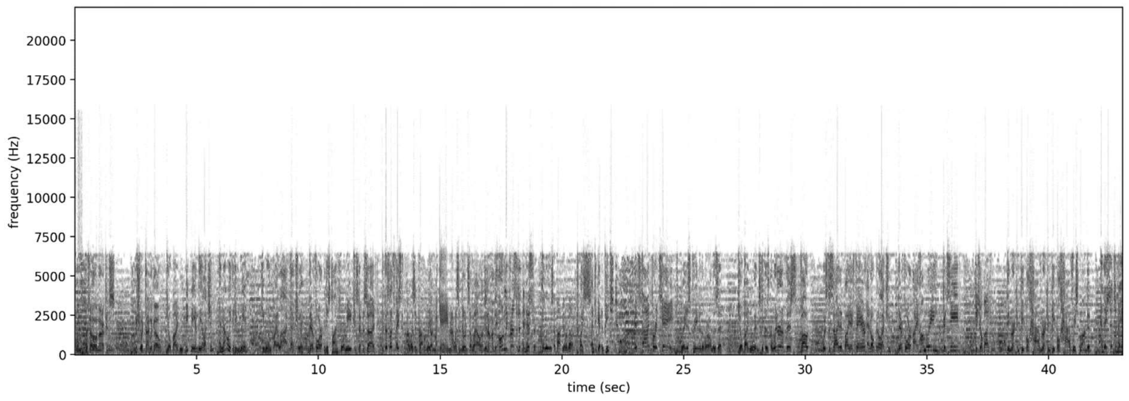


Figure 4

This spectrogram also represents that amplitude of the frequencies 0-1000Hz is very high. Meaning a human voice is present along the while audio. 1000-6000Hz are also Dark that represent high amplitude, these must be for the background music.

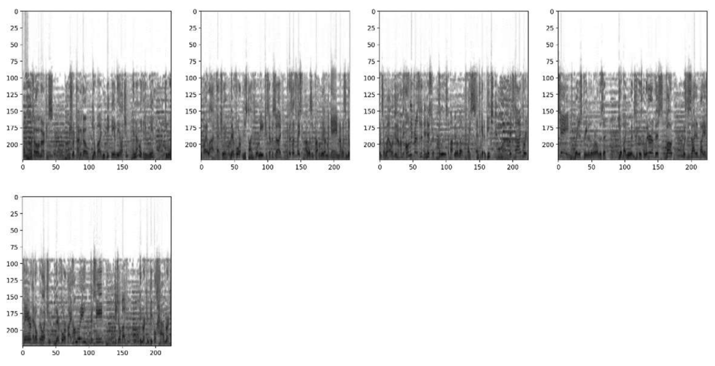


Figure 5

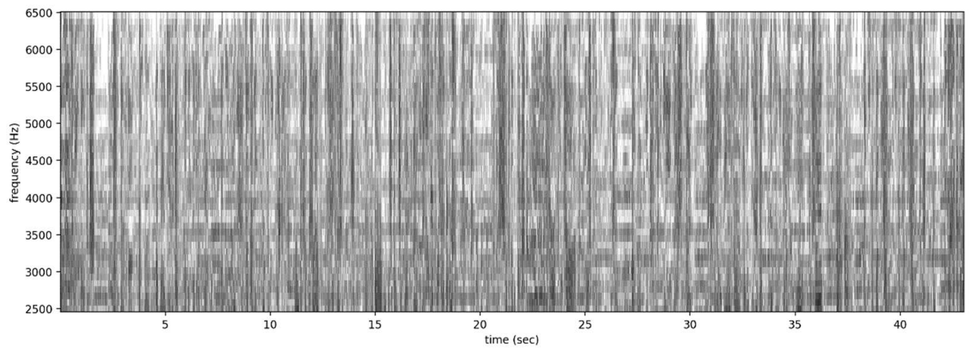


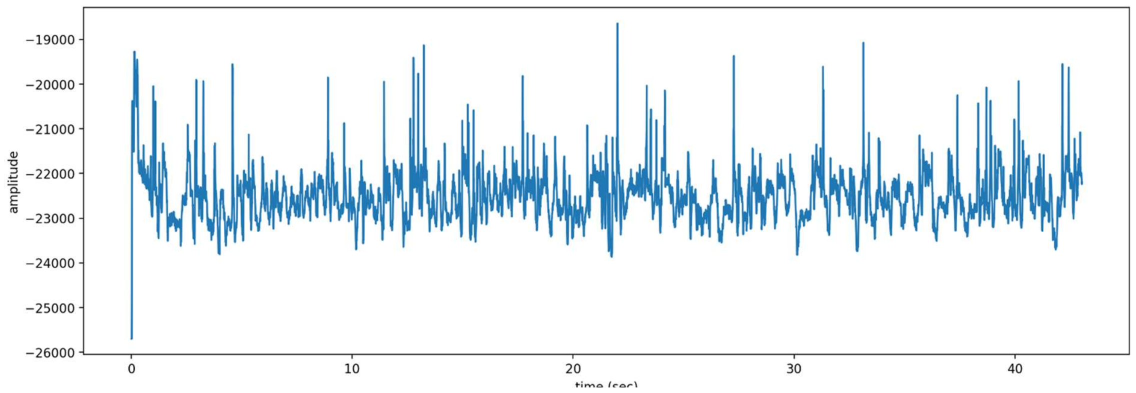
Figure 6

Now we calculate the amplitude of the signal to infer different peaks of the signal. Different splices might have different peaks, so we detect the audio peaks using the given python code:

```
# calculate amplitude signal
high_freq_amplitude = spec_trimmed.amplitude()

# plot
from matplotlib import pyplot as plt
plt.plot(spec_trimmed.times,high_freq_amplitude)
plt.xlabel('time (sec)')
plt.ylabel('amplitude')
plt.show()
```

The code gives the following peaks:



8 USING CONVOLUTIONAL NEURAL NETWORKS PRE TRAINED MODELS FOR DETECTION:

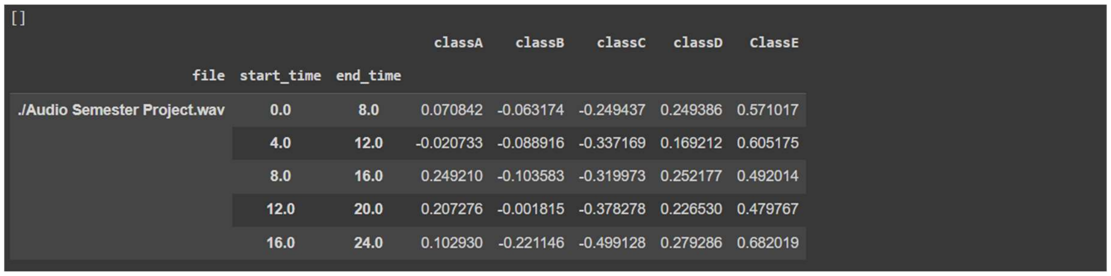
Now for detection of different splices we used pre-trained models using convolutional neural networks and imported them to our workspace to detect if different classes are present in our audio. First, we tested on 2 classes and then further increases the time interval to cater all the range of audios and detected splicing using them. The pre-trained models imported were from **opensource soundscape** , a preprocessing library in python which proves open source audio processing tools.

The code for Machine Learning algorithm is given as:

```
from opensoundscape.torch.models.cnn import load_model
import opensoundscape
# Other utilities and packages
import torch
from pathlib import Path
import numpy as np
import pandas as pd
from glob import glob
import subprocess
#set up plotting
from matplotlib import pyplot as plt
plt.rcParams['figure.figsize']=[40,2] #for large visuals
%config InlineBackend.figure_format = 'retina'
from opensoundscape.torch.models.cnn import CNN
```

```
CNN('resnet18',['classA','classB','classC','classD','ClassE'],8.0).save
('./temp.model')
```

The results are as shown:



```
[]
```

			classA	classB	classC	classD	ClassE
file	start_time	end_time					
./Audio Semester Project.wav	0.0	8.0	0.070842	-0.063174	-0.249437	0.249386	0.571017
	4.0	12.0	-0.020733	-0.088916	-0.337169	0.169212	0.605175
	8.0	16.0	0.249210	-0.103583	-0.319973	0.252177	0.492014
	12.0	20.0	0.207276	-0.001815	-0.378278	0.226530	0.479767
	16.0	24.0	0.102930	-0.221146	-0.499128	0.279286	0.682019

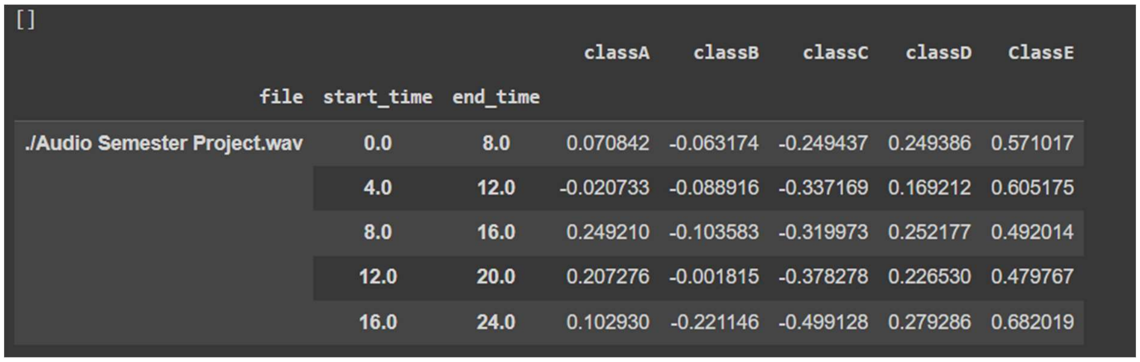
Figure 7

8.1 Adding Different Sub Classes:

By adding more different sub classes we obtained as:

```
from glob import glob
audio_files = glob('./*.wav') #match all .wav files in the current dire
ctory
audio_files
scores, _, _ = model.predict(audio_files, overlap_fraction=0.5)
scores.head()
```

The results for the ML model are as shown:



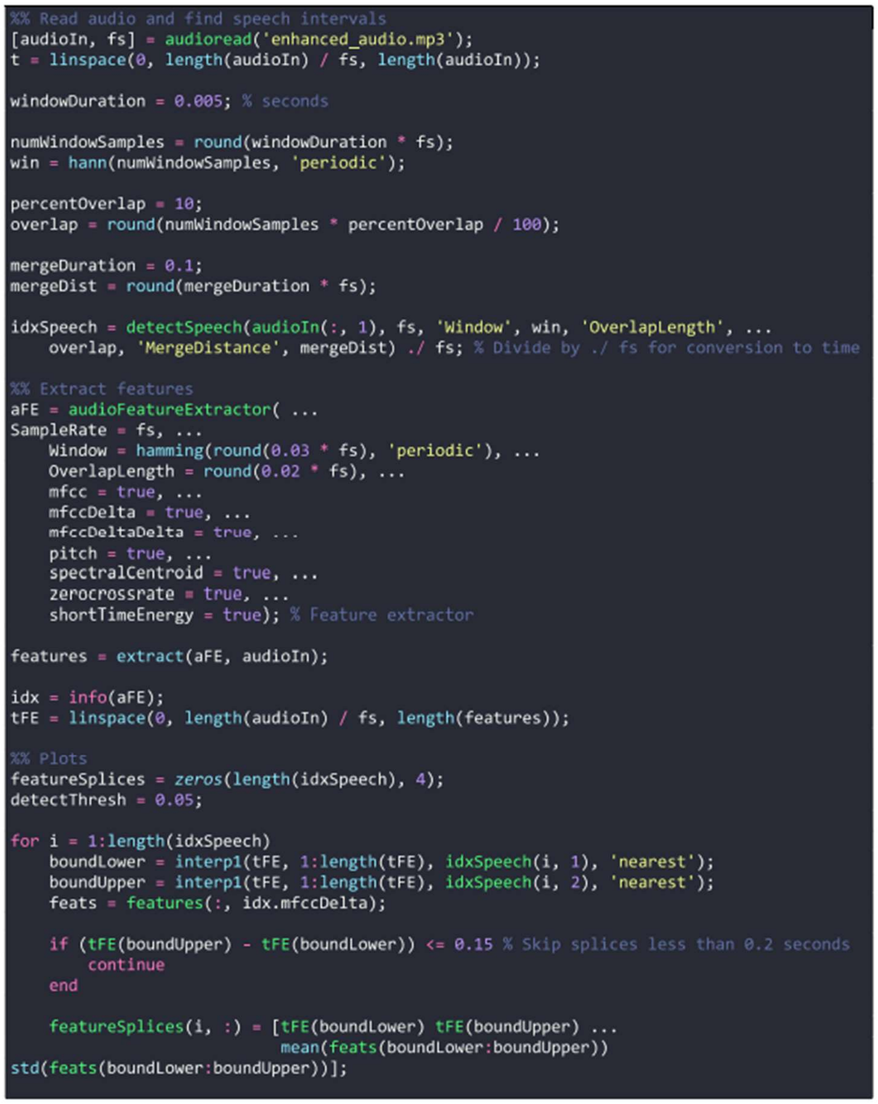
```
[]
```

			classA	classB	classC	classD	ClassE
file	start_time	end_time					
./Audio Semester Project.wav	0.0	8.0	0.070842	-0.063174	-0.249437	0.249386	0.571017
	4.0	12.0	-0.020733	-0.088916	-0.337169	0.169212	0.605175
	8.0	16.0	0.249210	-0.103583	-0.319973	0.252177	0.492014
	12.0	20.0	0.207276	-0.001815	-0.378278	0.226530	0.479767
	16.0	24.0	0.102930	-0.221146	-0.499128	0.279286	0.682019

9 USING MATLAB AUDIO TOOLBOX:

By using MATLAB audio toolkit, we performed audio splicing by using Mel-frequency cepstrum (MFC). In MFC, the frequency bands are equally spaced on the mel-scale, which approximates the human auditory system's response more closely than the linearly spaced frequency bands used in the normal spectrum. Hence, allowing for better sound representation. As stated earlier, what we specifically target are the delta features of the MFC, MFCCDelta, so as in order to understand the dynamics of power spectrum better, and directly in relation to time.

The code for our splicing detection is given as:



```
%% Read audio and find speech intervals
[audioIn, fs] = audioread('enhanced_audio.mp3');
t = linspace(0, length(audioIn) / fs, length(audioIn));

windowDuration = 0.005; % seconds

numWindowSamples = round(windowDuration * fs);
win = hann(numWindowSamples, 'periodic');

percentOverlap = 10;
overlap = round(numWindowSamples * percentOverlap / 100);

mergeDuration = 0.1;
mergeDist = round(mergeDuration * fs);

idxSpeech = detectSpeech(audioIn(:, 1), fs, 'Window', win, 'OverlapLength', ...
    overlap, 'MergeDistance', mergeDist) ./ fs; % Divide by ./ fs for conversion to time

%% Extract features
aFE = audioFeatureExtractor( ...
SampleRate = fs, ...
    Window = hamming(round(0.03 * fs), 'periodic'), ...
    OverlapLength = round(0.02 * fs), ...
    mfcc = true, ...
    mfccDelta = true, ...
    mfccDeltaDelta = true, ...
    pitch = true, ...
    spectralCentroid = true, ...
    zerocrossrate = true, ...
    shortTimeEnergy = true); % Feature extractor

features = extract(aFE, audioIn);

idx = info(aFE);
tFE = linspace(0, length(audioIn) / fs, length(features));

%% Plots
featureSplices = zeros(length(idxSpeech), 4);
detectThresh = 0.05;

for i = 1:length(idxSpeech)
    boundLower = interp1(tFE, 1:length(tFE), idxSpeech(i, 1), 'nearest');
    boundUpper = interp1(tFE, 1:length(tFE), idxSpeech(i, 2), 'nearest');
    feats = features(:, idx.mfccDelta);

    if (tFE(boundUpper) - tFE(boundLower)) <= 0.15 % Skip splices less than 0.2 seconds
        continue
    end

    featureSplices(i, :) = [tFE(boundLower) tFE(boundUpper) ...
                            mean(feats(boundLower:boundUpper))
std(feats(boundLower:boundUpper))];
```

The detected splices in MATLAB graph are given as:

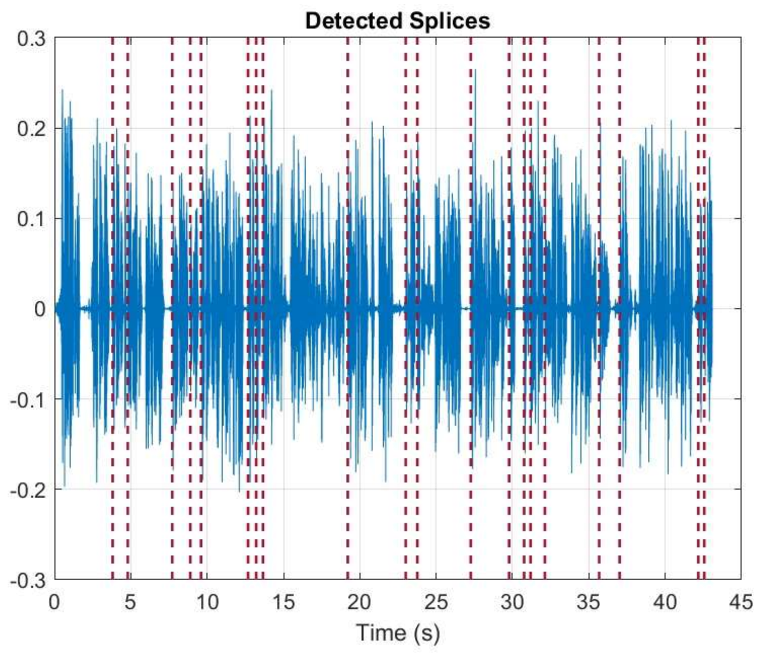


Figure 8

The following graph obtained after performing the mel- frequency cepstrum on the Fourier coefficients obtained from the python code. We observe that the mel frequency spectrum performed on different parts of the audio gives the different graphs which can be used to detect the different splicing.

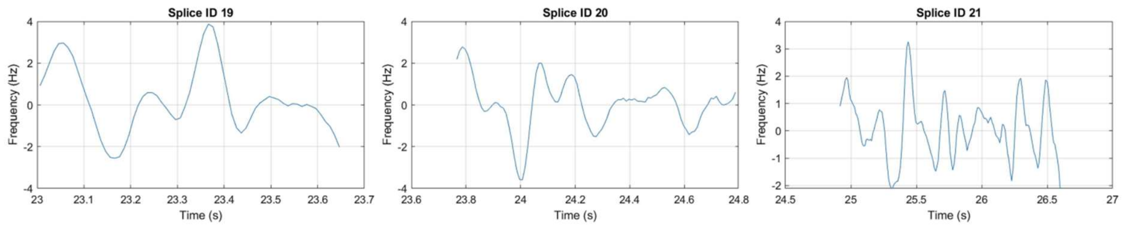


Figure 9

10 Detecting Splices:

The splicing in the given audio signal is given as:

Splice	Time
1	3.8
2	4.2
3	7.7
4	12.5
5	13.7
6	17.9
7	18.5
8	19.7
9	20.8
10	21.6
11	22.6
12	23.0
13	23.8
14	29.8
15	31.2
16	31.9
17	32.1
18	35.7
19	36.8
20	37.0
21	37.9
22	39.0
23	39.8
24	42.2
25	42.6

11 LIMITATIONS OF THIS TECHNIQUE:

The problems faced during the project are as under:

The proposed technique is good for audio that does not have a significant amount of noise in it. As the noise in the audio is increased the abrupt change in frequency due to noise will be detected as splicing. This issue was solved to some extent during pre-processing and enhancing of the audio, but it is not an appropriate solution.

The intervals taken while comparing the spectrogram of a different part of the audio were eight seconds. If the number of intervals is increased while reducing the duration, better result could be obtained but the Resnet18 is limited to only a few intervals.

12 CONCLUSION:

In this project we learnt how to preprocess audio using both MATLAB and python to preprocess audio and how to filter out different effects to focus on the required part of the audio. We performed different functions on our audio signal to detect the splicing using ML models such as **RESNET 18** and used MATLAB audio tool kit. This gave us an insight on different aspects of how MATLAB can be sued in our daily life effectively and its applications.

13 REFERENCES:

The references are as shown:

1. https://scholar.google.com.pk/scholar?q=audio+splicing+detection&hl=en&as_sdt=0&as_vis=1&oi=scholart
2. https://arxiv.org/abs/1411.7084
3. https://arxiv.org/abs/2210.03581
4. https://link.springer.com/article/10.1007/s11042-016-3758-7
5. https://www.researchgate.net/publication/268819903_Audio_Splicing_Detection_and_Localization_Using_Environmental_Signature
6. https://ieeexplore.ieee.org/document/8944345