Configuration Manual

Identification of Distressed Animal Vocalisation Using Deep Transfer Clustering

MSc Research Project
Data Analytics

Midhun Satheesan
Student ID: 19146035

School of Computing
National College of Ireland

Supervisor: Dr. Catherine Mulwa

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Midhun Satheesan
Student ID:	19146035
Programme:	MSCDAD **Year:** 2020-2021
Module:	Research Project
Lecturer:	Dr. Catherine Mulwa
Submission Due Date:	17th December 2020
Project Title:	Identification of Distressed Animal Vocalisation Using Deep Transfer Clustering
Word Count:	2641 **Page Count:** 51

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: …………………Midhun Satheesan…………

Date: ………………………17th December 2020……………

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is	□

lost or mislaid. It is not sufficient to keep a copy on computer.	

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Midhun Satheesan
Student ID: 19146035

1 Introduction

This document helps the reader to set up and run the ICT solution created to cluster and detect distress in animal vocalisation. You will find detailed step-by-step procedures for hardware set up, software installations, data exploration and analysis, classification modelling and k-means clustering of audios associated with the project.

2 Hardware Configuration

This was a hardware resource intensive project due to the scale of data computations involved. The recommended specification for the training tasks performed is in Figure 1.

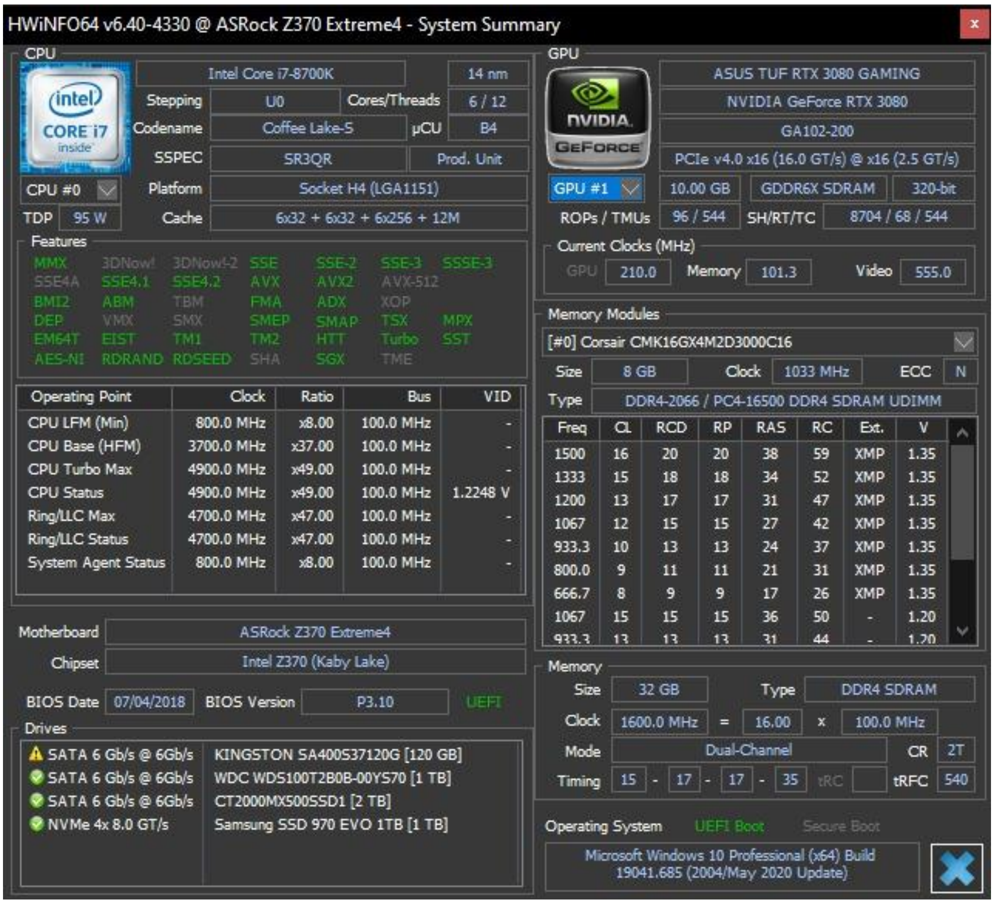


Figure 1: Primary system specs (recommended).

The data analysis, clustering and other less demanding computation can be done on the system with the following specs (Figure 2). The model training was configured to run on the CUDA supported NVIDEA GPUs of both the systems.

Figure 2: Secondary system specs.

3 Software Configuration

This project was done on a Microsoft Windows 10 Operating System. This project can also be run on any Linux or Apple Mac OS environments if the installation manuals of all the required software listed in this section are followed. However, explanation of configuration in all the environments is out of scope of this configuration manual. The succeeding subsections explains elaborately the setting up of software environment in the Windows 10 OS.

3.1 Installing Anaconda

Anaconda is a free, open-source distribution which simplifies working with Python and R for machine learning.

1. Download Anaconda installer from the official website.[1]

[1] https://www.anaconda.com/products/individual

Figure 3: Anaconda download web page

2. Begin the installer and choose the appropriate directory to install Anaconda.

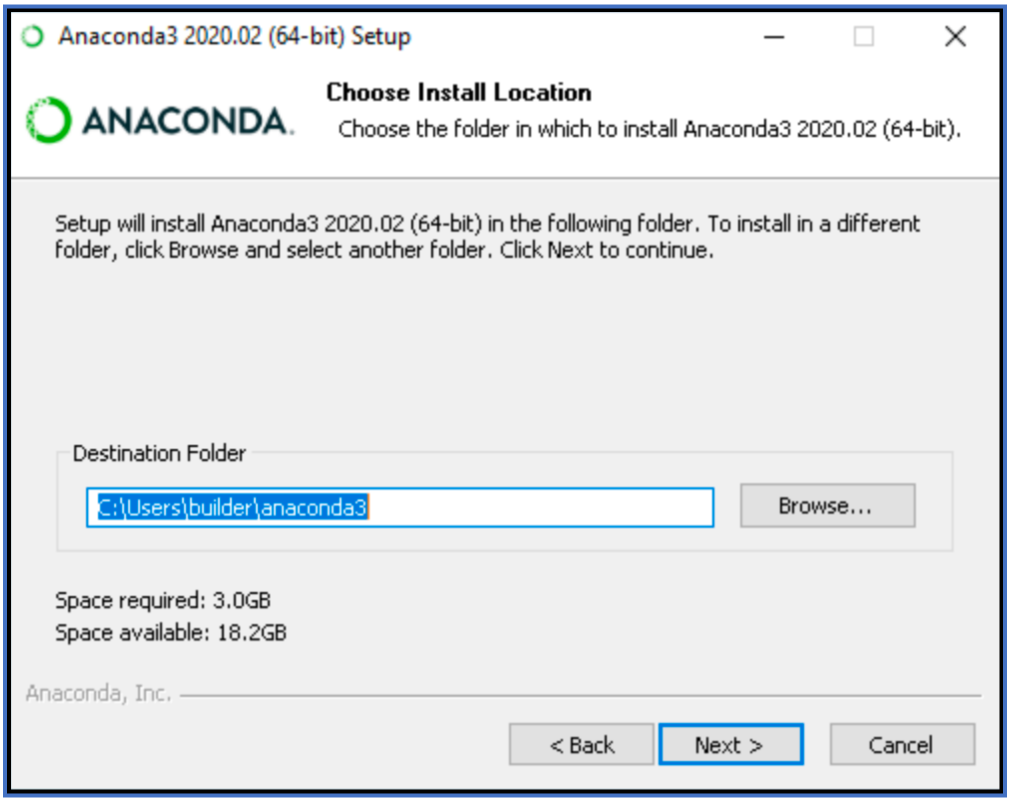


Figure 4: Select installation directory

3. Choose the default Python installation for Anaconda to register as shown in Figure 5. Avoid adding to PATH variable as this might cause unwanted interference.

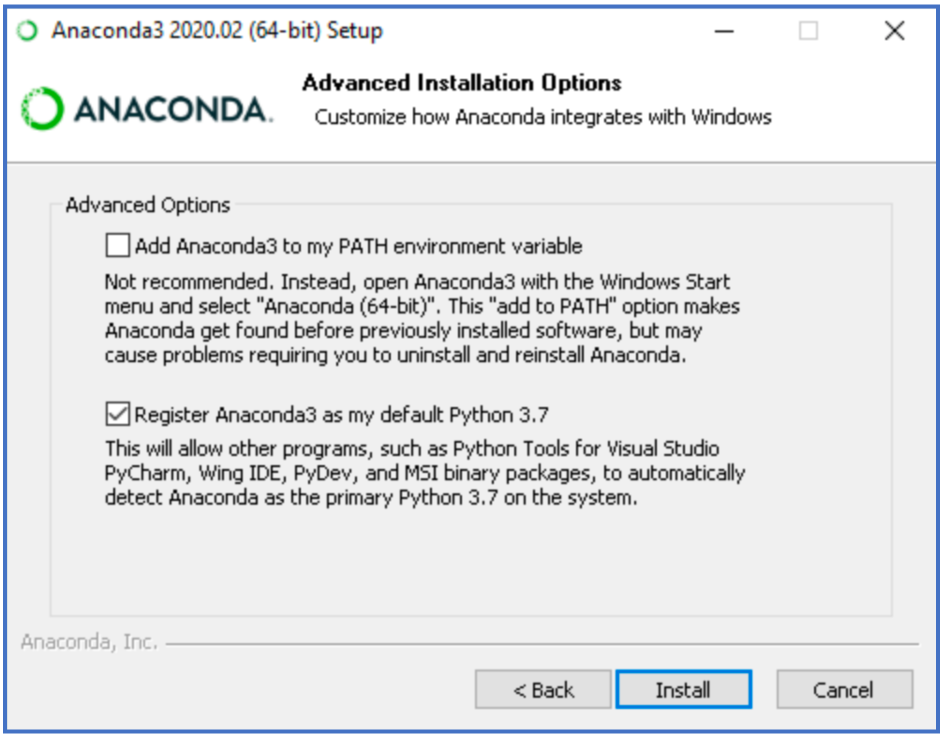


Figure 5: Choose Python version

4. Click next to complete Anaconda installation.

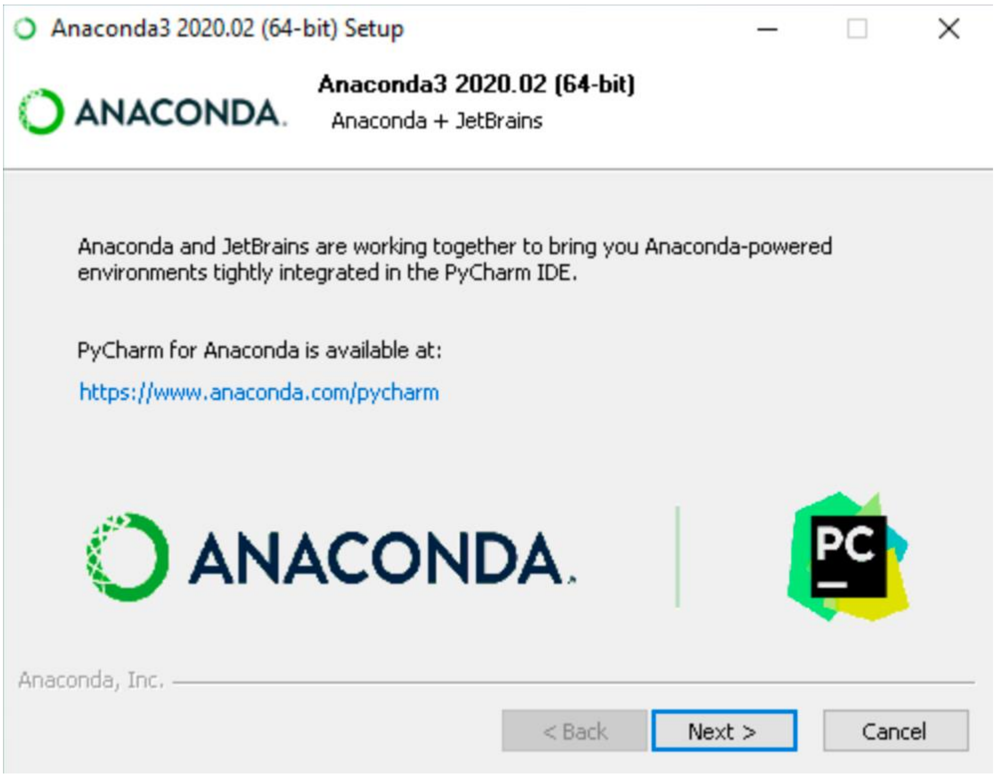


Figure 6: Option to download pycharm

Installation will be complete with this screen. Open and explore Anaconda.

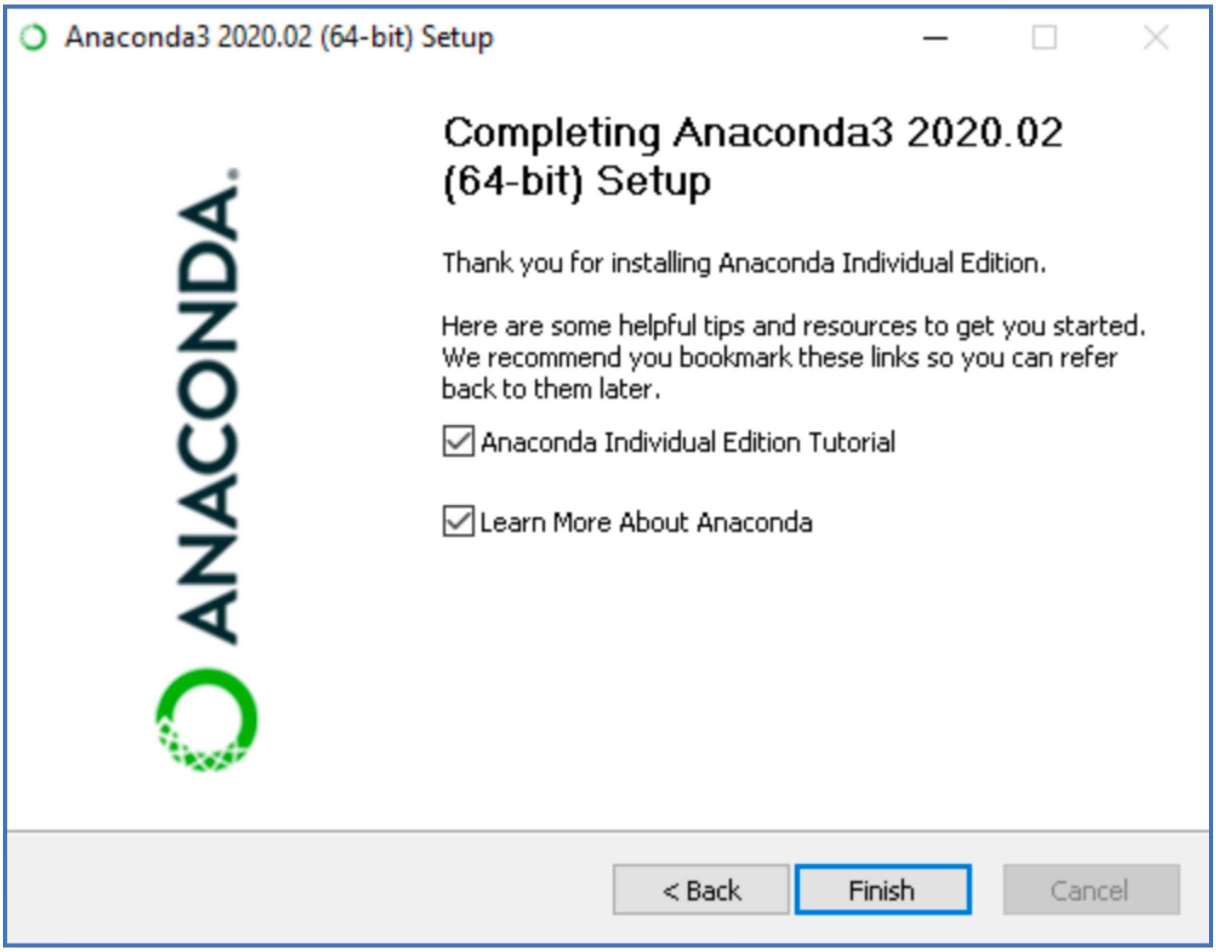


Figure 7: Installation completed

3.2 Installing TensorFlow 2.x (latest)

1. Open Jupyter notebook. It is part of the Anaconda framework. Search for Jupyter notebook in windows search bar and choose open. This will bring the screen in Figure 8

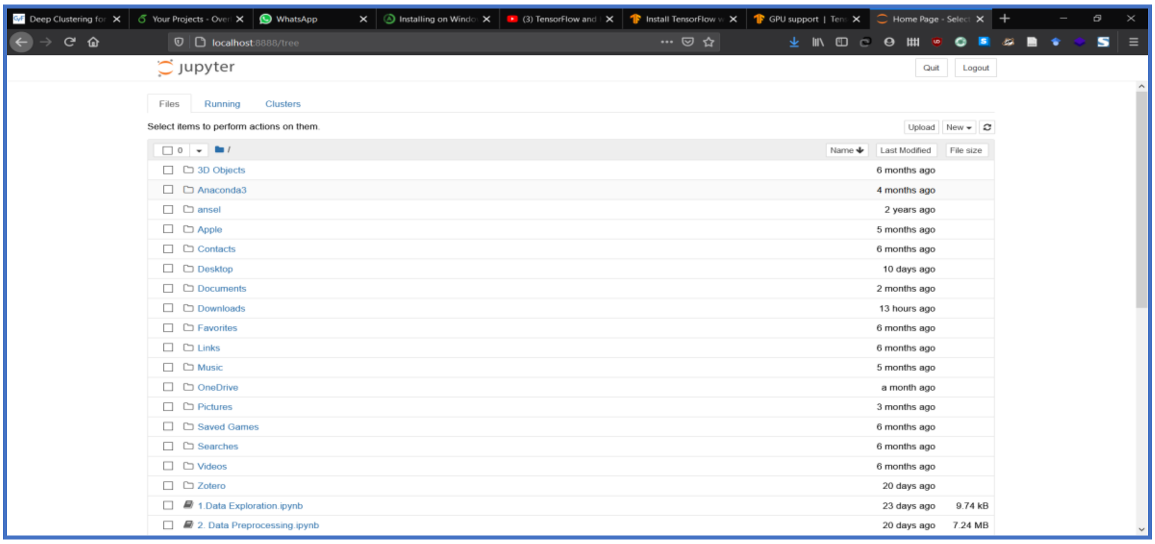


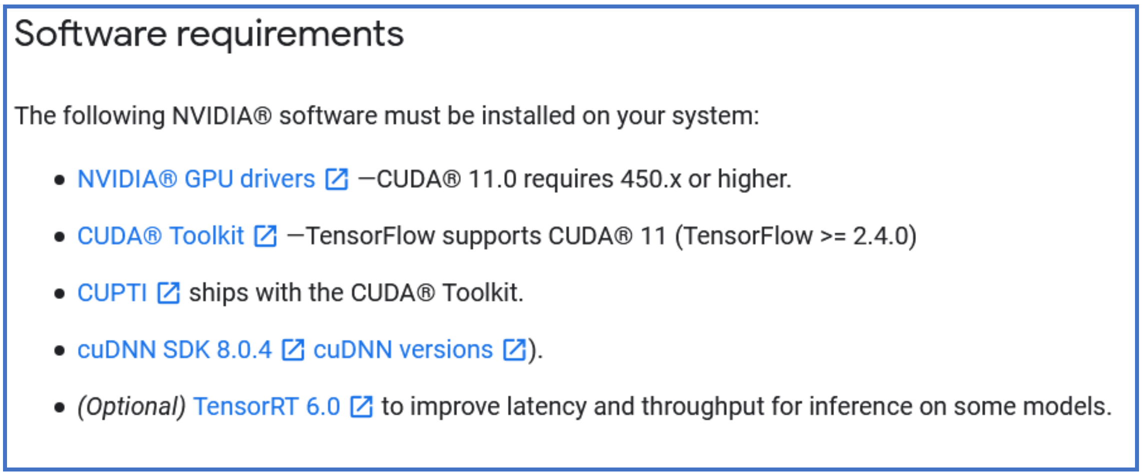
Figure 8: Jupyter notebook home directory.

2. Open a new Python 3 notebook from the top left UI dropdown (Figure 8).
3. Type the following command and run the cell. TensorFlow will be installed.

```
!pip install tensorflow
```

3.3 Configure TensorFlow and Keras for GPU Support

TensorFlow's GPU support do not come out of the box. There are few libraries including CUDA Toolkit and cuDNN to be installed and configured I order to make things work. The figure 9 shows all the required installations required. A more detailed install process is available in the official TensorFlow website – GPU Support.
For a well detailed video walkthrough of the entire installation, please check out the following link - TensorFlow and Keras GPU Support - CUDA GPU Setup (Video) - deeplizard. Please make sure that you have an NVIDIA manufactured graphics card. Make sure it is in the list in the official website - CUDA GPUs by NVIDIA



Software requirements

The following NVIDIA® software must be installed on your system:

- NVIDIA® GPU drivers —CUDA® 11.0 requires 450.x or higher.
- CUDA® Toolkit —TensorFlow supports CUDA® 11 (TensorFlow >= 2.4.0)
- CUPTI ships with the CUDA® Toolkit.
- cuDNN SDK 8.0.4 cuDNN versions).
- *(Optional)* TensorRT 6.0 to improve latency and throughput for inference on some models.

Figure 9: Software required for TensorFlow GPU support.

4 Data Acquisition and Exploration

This section explains how to obtain the dataset used in this project. After downloading the dataset, we explore the data using tools and libraries in Python.

4.1 FSD50K Dataset Download

All the data required for this project is obtained from the FSD50K dataset which was released on October 2, 2020. It is the first if its kind, large scale, open dataset of human labelled audios. The download link is given below.

FSD50K Download Link

It is highly recommended to read the research paper by the creators of the dataset to thoroughly understand its properties. This will equip the reader with insights on how to use this dataset. The research paper is available at – FSD50K – Electronic Preprint .

More information regarding the dataset can be found in the links given below –

Creator's Personal Website

Companion website with all 200 labels listing

1. The download link will take you to the open access data repository – Zenodo shown in the figure 10.

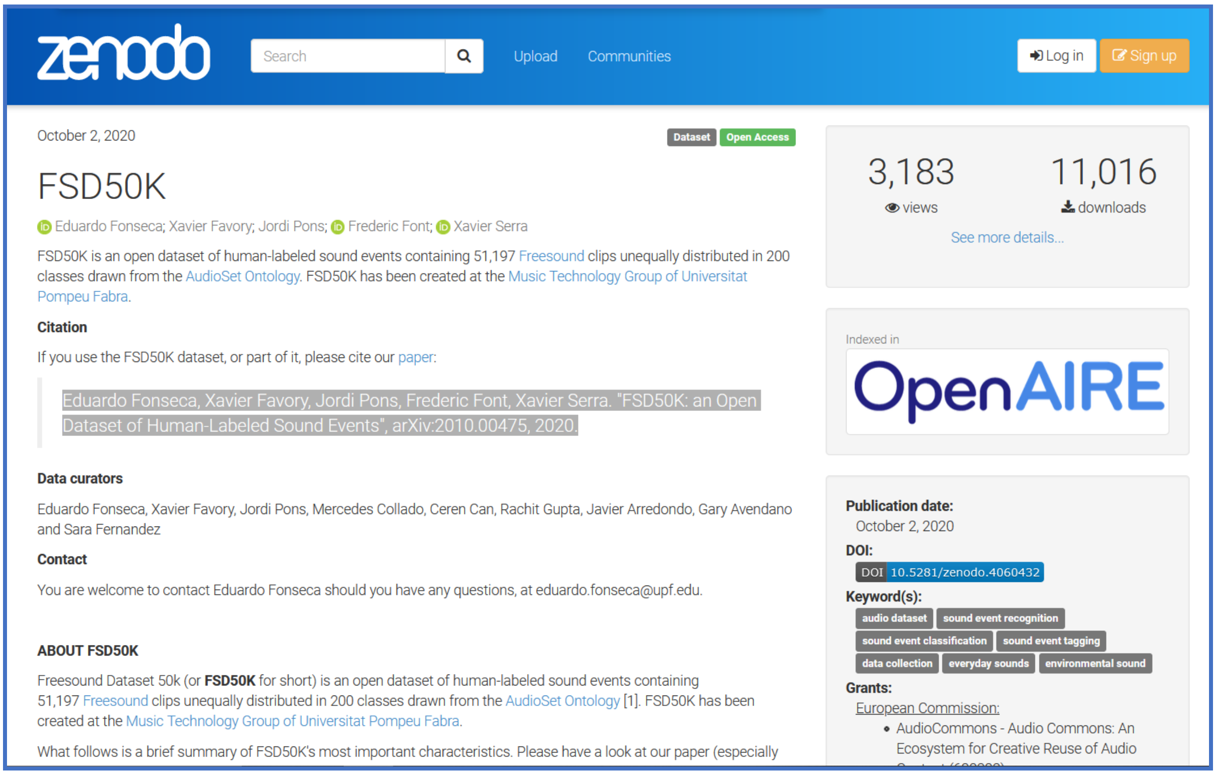
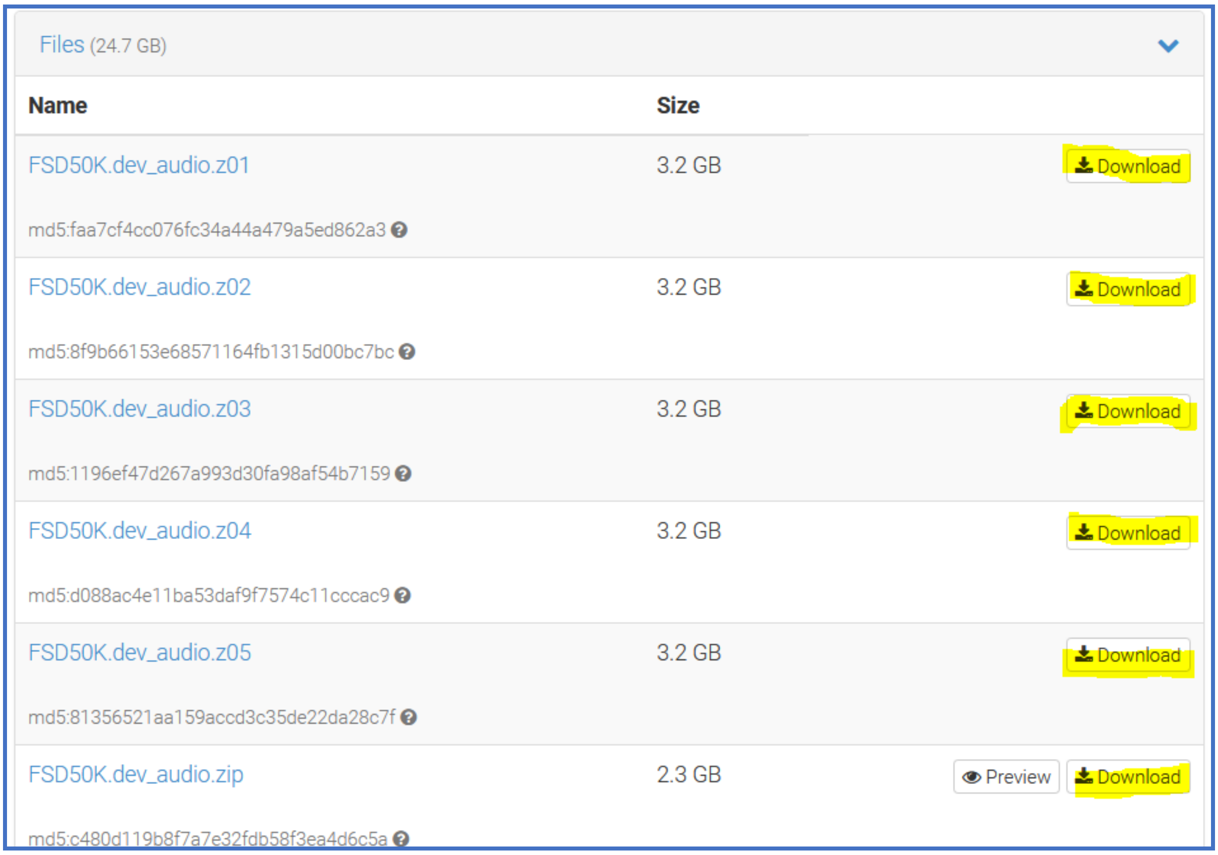


Figure 10: FSD50K download - Zenodo.

2. The dataset can be downloaded as a series of ZIP files from this page. The figure 11 displays the directory structure of the dataset. Figure 12 shows the section in the web page where the ZIP files and metadata can be downloaded. Figure 13 displays the evaluation dataset and the metadata download section of the dataset.

```
root
│
└───FSD50K.dev_audio/                   Audio clips in the dev set
│
└───FSD50K.eval_audio/                  Audio clips in the eval set
│
└───FSD50K.ground_truth/                Files for FSD50K's ground truth
│   │
│   └─── dev.csv                              Ground truth for the dev set
│   │
│   └─── eval.csv                             Ground truth for the eval set
│   │
│   └─── vocabulary.csv                       List of 200 sound classes in FSD50K
│
└───FSD50K.metadata/                    Files for additional metadata
│   │
│   └─── class_info_FSD50K.json               Metadata about the sound classes
│   │
│   └─── dev_clips_info_FSD50K.json           Metadata about the dev clips
│   │
│   └─── eval_clips_info_FSD50K.json          Metadata about the eval clips
│   │
│   └─── pp_pnp_ratings_FSD50K.json           PP/PNP ratings
│   │
│   └─── collection/                          Files for the *sound collection* format
│
└───FSD50K.doc/
    │
    └───README.md                             The dataset description file that you are reading
    │
    └───LICENSE-DATASET                       License of the FSD50K dataset as an entity
```

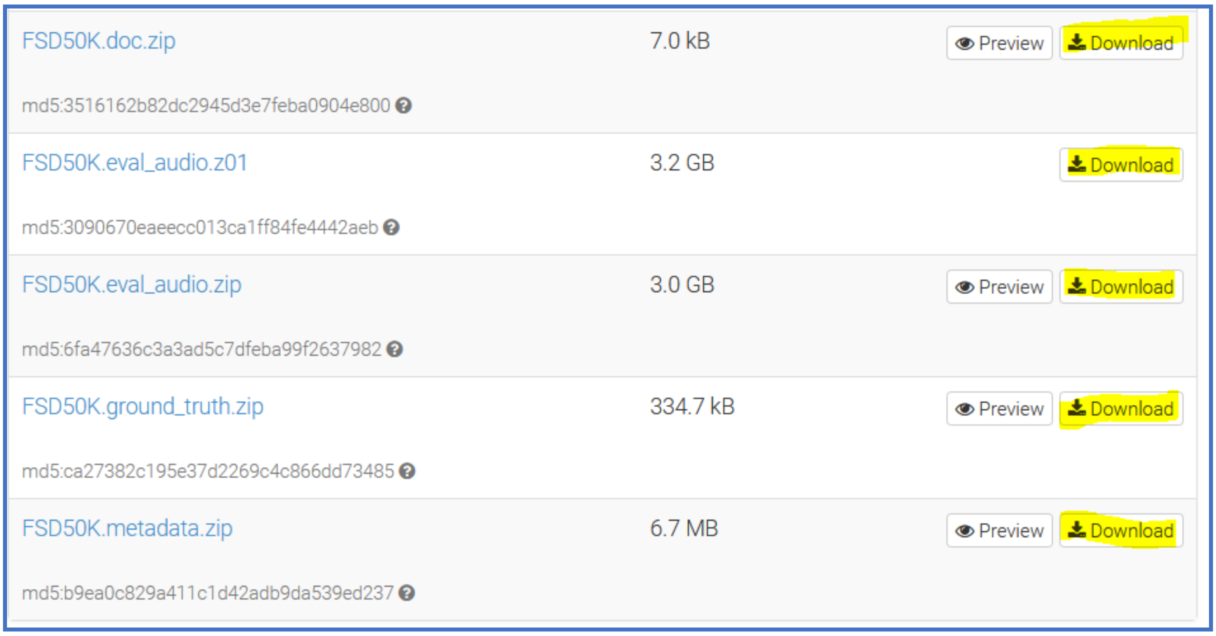
Figure 11: Directory structure - FSD50K



Files (24.7 GB)

Name	Size	
FSD50K.dev_audio.z01 md5:faa7cf4cc076fc34a44a479a5ed862a3	3.2 GB	Download
FSD50K.dev_audio.z02 md5:8f9b66153e68571164fb1315d00bc7bc	3.2 GB	Download
FSD50K.dev_audio.z03 md5:1196ef47d267a993d30fa98af54b7159	3.2 GB	Download
FSD50K.dev_audio.z04 md5:d088ac4e11ba53daf9f7574c11cccac9	3.2 GB	Download
FSD50K.dev_audio.z05 md5:81356521aa159accd3c35de22da28c7f	3.2 GB	Download
FSD50K.dev_audio.zip md5:c480d119b8f7a7e32fdb58f3ea4d6c5a	2.3 GB	Preview Download

Figure 12: Download section of web page



FSD50K.doc.zip md5:3516162b82dc2945d3e7feba0904e800	7.0 kB	Preview Download
FSD50K.eval_audio.z01 md5:3090670eaeecc013ca1ff84fe4442aeb	3.2 GB	Download
FSD50K.eval_audio.zip md5:6fa47636c3a3ad5c7dfeba99f2637982	3.0 GB	Preview Download
FSD50K.ground_truth.zip md5:ca27382c195e37d2269c4c866dd73485	334.7 kB	Preview Download
FSD50K.metadata.zip md5:b9ea0c829a411c1d42adb9da539ed237	6.7 MB	Preview Download

Figure 13: Evaluation dataset and Metadata

3. After downloading the ZIP files to the preferred folder, choose all the files and Unzip together to combine them into one using 7zip. The figure 14: shows the files in the folder after unzipping. There are total of 40,996 items of variable length.

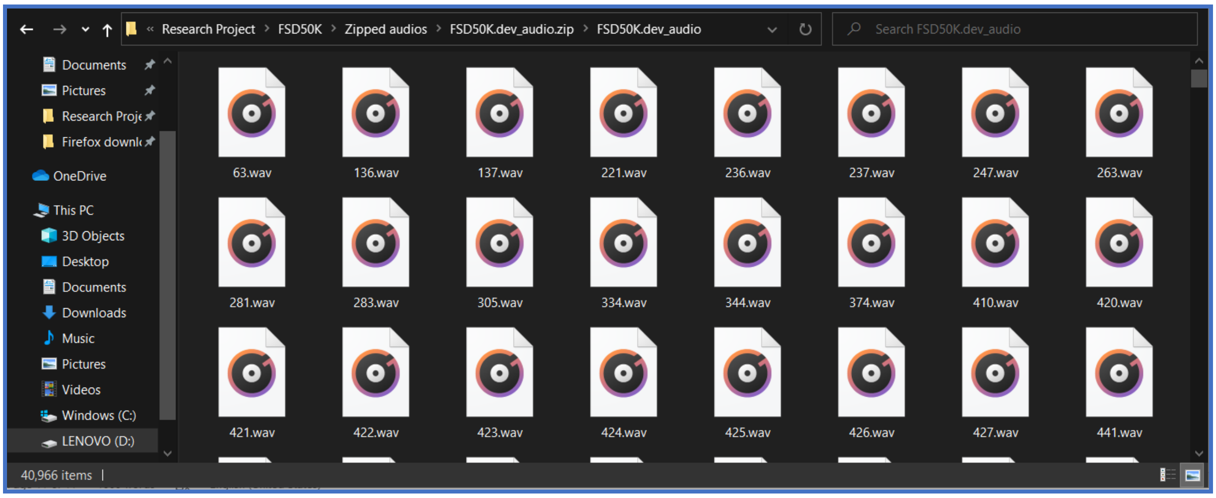


Figure 14: Audio files.

4. The figure 15 shows the ground_truth folder which has all the meta data. The dev.csv contains all the labels associated with dev folder audios (figure 16). The eval.csv is similar contains all the labels associated with evaluation folder audios. The files also contain information regarding whether the file belong to train split or test spit. There is a vocabulary.csv (figure 17) file which includes the list of all labels.

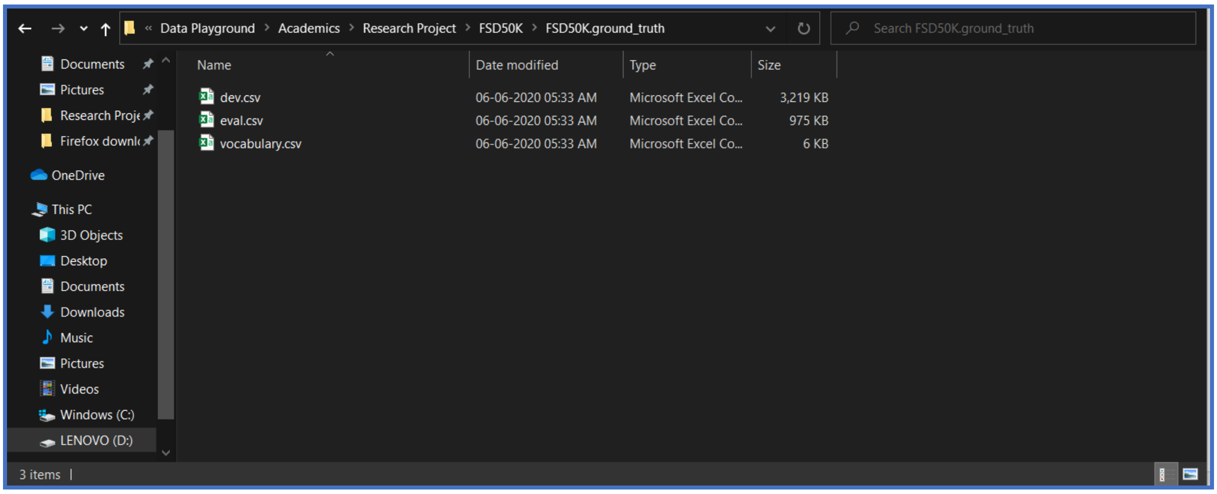


Figure 15: Metadata Folder

	A	B	C	D	E
1	fname	labels	mids	split	
2	64760	Electric_guitar,Guitar,Plucked_string_instrument	/m/02sgy,/	train	
3	16399	Electric_guitar,Guitar,Plucked_string_instrument	/m/02sgy,/	train	
4	16401	Electric_guitar,Guitar,Plucked_string_instrument	/m/02sgy,/	train	
5	16402	Electric_guitar,Guitar,Plucked_string_instrument	/m/02sgy,/	train	
6	16404	Electric_guitar,Guitar,Plucked_string_instrument	/m/02sgy,/	train	
7	345111	Electric_guitar,Guitar,Plucked_string_instrument	/m/02sgy,/	val	
8	64761	Electric_guitar,Guitar,Plucked_string_instrument	/m/02sgy,/	train	
9	268259	Electric_guitar,Guitar,Plucked_string_instrument	/m/02sgy,/	train	
10	64762	Electric_guitar,Guitar,Plucked_string_instrument	/m/02sgy,/	train	
11	160826	Electric_guitar,Guitar,Plucked_string_instrument	/m/02sgy,/	val	

Figure 16: ground_truth of audios CSV

	A	B	C	D
1	0	Accelerating_and_revving_and_	/m/07q2z82	
2	1	Accordion	/m/0mkg	
3	2	Acoustic_guitar	/m/042v_gx	
4	3	Aircraft	/m/0k5j	
5	4	Alarm	/m/07pp_mv	
6	5	Animal	/m/0jbk	
7	6	Applause	/m/028ght	
8	7	Bark	/m/05tny_	
9	8	Bass_drum	/m/0bm02	
10	9	Bass_guitar	/m/018vs	
11	10	Bathtub_(filling_or_washing)	/m/03dnzn	

Figure 17: vocabulary.csv

4.2 Data Exploration (Extra – Not included in the Research Report)

Data exploration is done to get to know the features of the samples in the dataset. This is done using Python and an audio library called librosa. Librosa can be installed by following their documentation – [Librosa Documentation](). Librosa is used to convert the audio signal to their feature representations. We will see how to convert and display the audios into pictorial feature representations such as: Waveforms, power spectrums, log-mel spectrograms and MFCCs.

1. Import all necessary libraries (Figure 18).

```
In [1]: import os
        import numpy as np
        import pandas as pd
        import tensorflow as tf
        from tensorflow import keras
        import matplotlib.pyplot as plt
        import seaborn as sns
```

Figure 18: Importing libraries.

2. Figure 19 displays the setting of path and reading of the CSV file. It is noted that there are 200 rows. A sample of the data is displayed.

```
ABS_PATH = os.path.abspath(os.path.join('D:', '/Data Playground/Academics/Research Project'))
```

```
vocabulary = pd.read_csv(ABS_PATH+'/FSD50K/FSD50K.ground_truth/vocabulary.csv', header=None)
print("Number of rows in data =",vocabulary.shape[0])
print("Number of columns in data =",vocabulary.shape[1])
print("\n")
print("**Sample data:**")
vocabulary.head()
```

```
Number of rows in data = 200
Number of columns in data = 3


**Sample data:**
```

	0	1	2
0	0	Accelerating_and_revving_and_vroom	/m/07q2z82
1	1	Accordion	/m/0mkg
2	2	Acoustic_guitar	/m/042v_gx
3	3	Aircraft	/m/0k5j
4	4	Alarm	/m/07pp_mv

Figure 19: The labels.

3. Figure 20 displays the metadata of the audios. This includes the labels.

```
In [4]: ground_truth = pd.read_csv(ABS_PATH+'/FSD50K/FSD50K.ground_truth/dev.csv')
        print("Number of audio files =",ground_truth.shape[0])
        print("Number of columns in data =",ground_truth.shape[1])
        print("\n")
        print("**Sample data:**")
        ground_truth.head()

        Number of audio files = 40966
        Number of columns in data = 4


        **Sample data:**
Out[4]:
```

	fname	labels	mids	split
0	64760	Electric_guitar,Guitar,Plucked_string_instrume...	/m/02sgy,/m/0342h,/m/0fx80y,/m/04szw,/m/04rlf	train
1	16399	Electric_guitar,Guitar,Plucked_string_instrume...	/m/02sgy,/m/0342h,/m/0fx80y,/m/04szw,/m/04rlf	train
2	16401	Electric_guitar,Guitar,Plucked_string_instrume...	/m/02sgy,/m/0342h,/m/0fx80y,/m/04szw,/m/04rlf	train
3	16402	Electric_guitar,Guitar,Plucked_string_instrume...	/m/02sgy,/m/0342h,/m/0fx80y,/m/04szw,/m/04rlf	train
4	16404	Electric_guitar,Guitar,Plucked_string_instrume...	/m/02sgy,/m/0342h,/m/0fx80y,/m/04szw,/m/04rlf	train

Figure 20: Audio metadata and labels.

4. Figure 21 shows the number of samples in train and test split according to the CSV file.

```
ground_truth.groupby(['split']).size()

split
train    36796
val       4170
dtype: int64
```

Figure 21: Train and test splits.

5. Import librosa and matplotlib.pyplot for visualising the audio (Figure 22).

```
In [8]: import librosa, librosa.display
        import matplotlib.pyplot as plt
        import IPython.display as ipd
```

Figure 22: Import libraries.

6. Figure 23 contains the function to play the audio in jupyter notebook.

```
In [ ]: def play(audio):
            x , sr = librosa.load(audio, sr=44100)
            print(type(x), type(sr))
            print(x.shape, sr)
            ipd.Audio(audio)
```

Figure 23: Code for playing audio.

7. Figure 24 contains the code which will create the plots for waveforms, power spectrum, MFCCs and spectrograms of the audio signal.

```
def pictures(audio):
    signal , sr = librosa.load(audio, sr=44100)
    print(x.shape, sr)
    # ipd.Audio(scream_audio1)
    # play(scream_audio2)
    # play(scream_audio3)
    # play(scream_audio4)
    #waveform
    librosa.display.waveplot(signal, sr=sr)
    plt.xlabel("Time")
    plt.ylabel("Amplitude")
    plt.show()
    #fft -> spectrum
    fft = np.fft.fft(signal)
    magnitude = np.abs(fft) #contribution of each frequency
    frequency = np.linspace(0, sr, len(magnitude))
    left_frequency = frequency[:int(len(frequency)/2)]
    left_magnitude = magnitude[:int(len(magnitude)/2)]
    power spectrum
    plt.plot(left_frequency, left_magnitude)
    plt.xlabel("Frequency")
    plt.ylabel("Magnitude")
    plt.show()
    #stft -> spectrogram
    n_fft = 2048
    hop_length = 512
    stft = librosa.core.stft(signal, hop_length=hop_length ,n_fft=n_fft)
    spectrogram = np.abs(stft)
    log_spectrogram = librosa.amplitude_to_db(spectrogram)
    librosa.display.specshow(log_spectrogram, sr=sr, hop_length=hop_length)
    plt.xlabel("Time")
    plt.ylabel("Frequency")
    plt.colorbar()
    plt.show()
    #MFCCs
    MFCCs = librosa.feature.mfcc(signal, n_fft=n_fft, hop_length=hop_length, n_mfcc=13)
    log_spectrogram = librosa.amplitude_to_db(MFCCs)
    librosa.display.specshow(log_spectrogram, sr=sr, hop_length=hop_length)
    plt.xlabel("Time")
    plt.ylabel("MFCC")
    plt.colorbar()
    plt.show()
```

Figure 24: Code to create waveforms, power spectrum, log mel spectrograms and MFCCs

8. Of all the labels, we focus on the labels signifying danger or distress and of that of animals. In the next few steps, we explore how these sounds will look like when represented in pictorial representations. Figure 25 shows setting paths right after manually finding samples in the dev audio folder. Figure 26 - 33 represents 4 samples each from the distress labels chosen in real life and animal vocalisations.

```
In [9]: scream_audio1 = 'D:/Data Playground/Academics/Research Project/FSD50K/FSD50K.dev_audio/40964.wav' #Yell
        scream_audio2 = 'D:/Data Playground/Academics/Research Project/FSD50K/FSD50K.dev_audio/333412.wav' #gasp
        scream_audio3 = 'D:/Data Playground/Academics/Research Project/FSD50K/FSD50K.dev_audio/381957.wav' #Siren
        scream_audio4 = 'D:/Data Playground/Academics/Research Project/FSD50K/FSD50K.dev_audio/254476.wav' #screaming

        animal_audio1 = 'D:/Data Playground/Academics/Research Project/FSD50K/FSD50K.dev_audio/236064.wav' #Dog
        animal_audio2 = 'D:/Data Playground/Academics/Research Project/FSD50K/FSD50K.dev_audio/402930.wav' #wild animals
        animal_audio3 = 'D:/Data Playground/Academics/Research Project/FSD50K/FSD50K.dev_audio/392243.wav' #Cat
        animal_audio4 = 'D:/Data Playground/Academics/Research Project/FSD50K/FSD50K.dev_audio/18073.wav' #Live stock
```

Figure 25: Paths

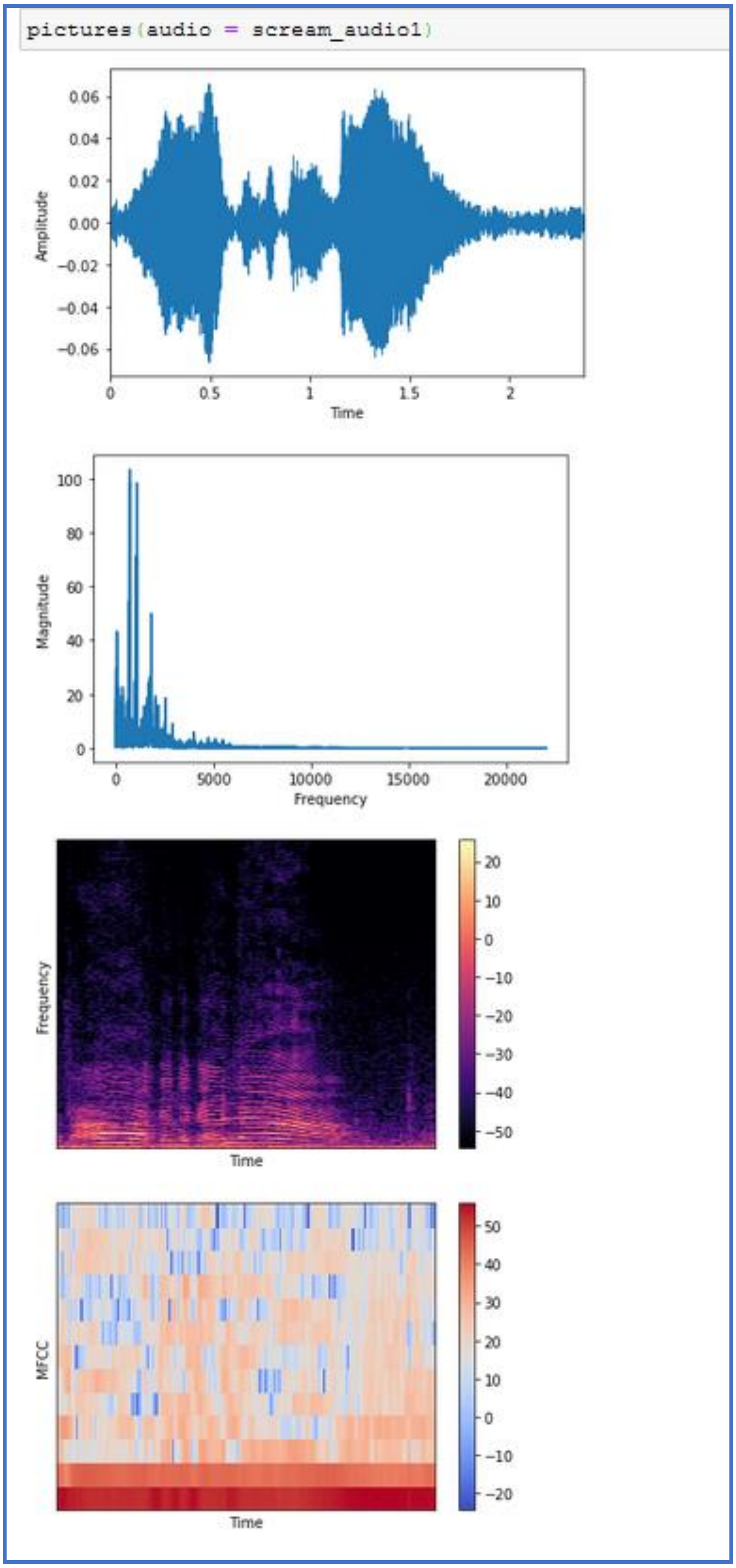


Figure 26: Yell

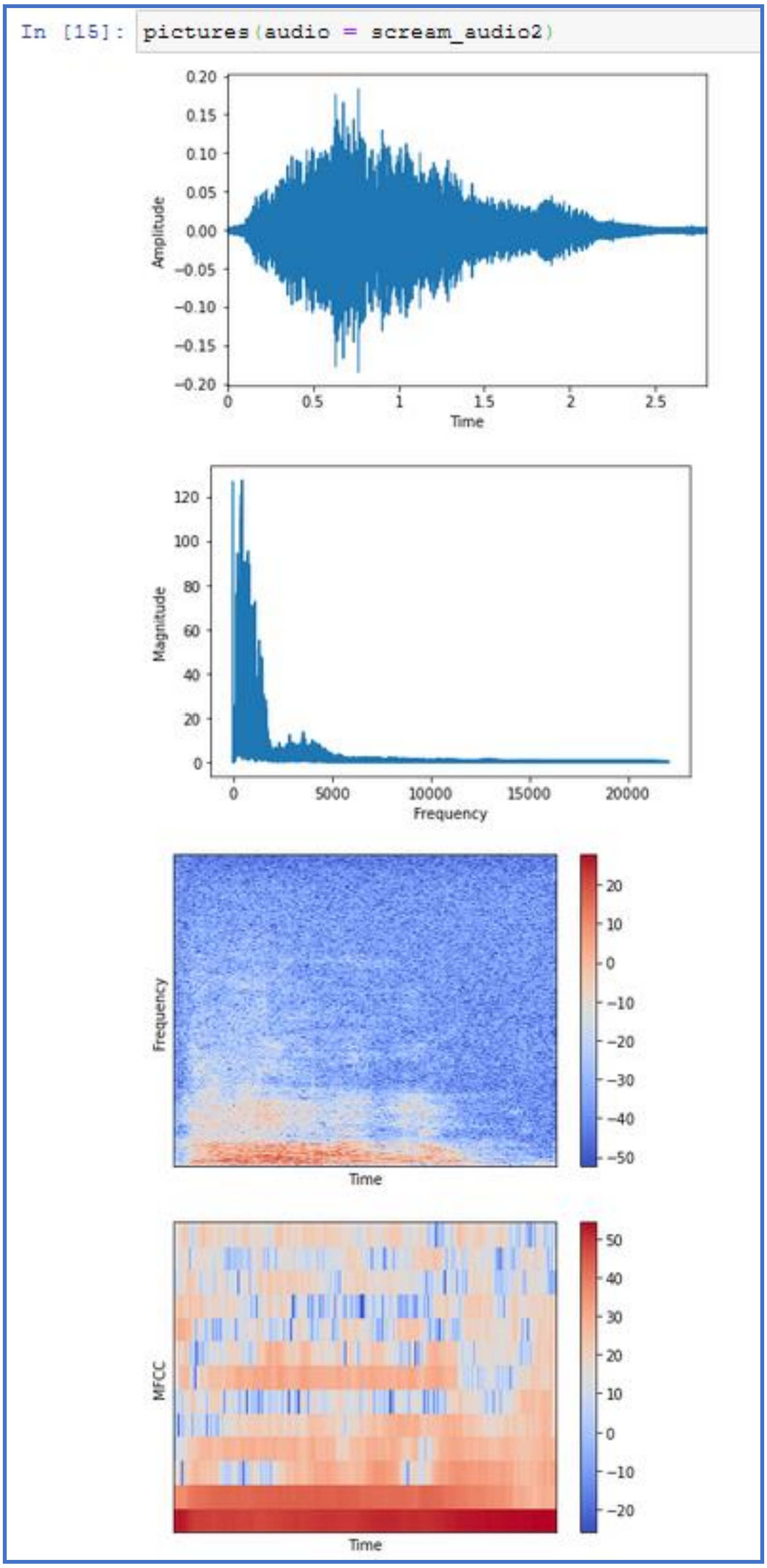


Figure 27: Gasp

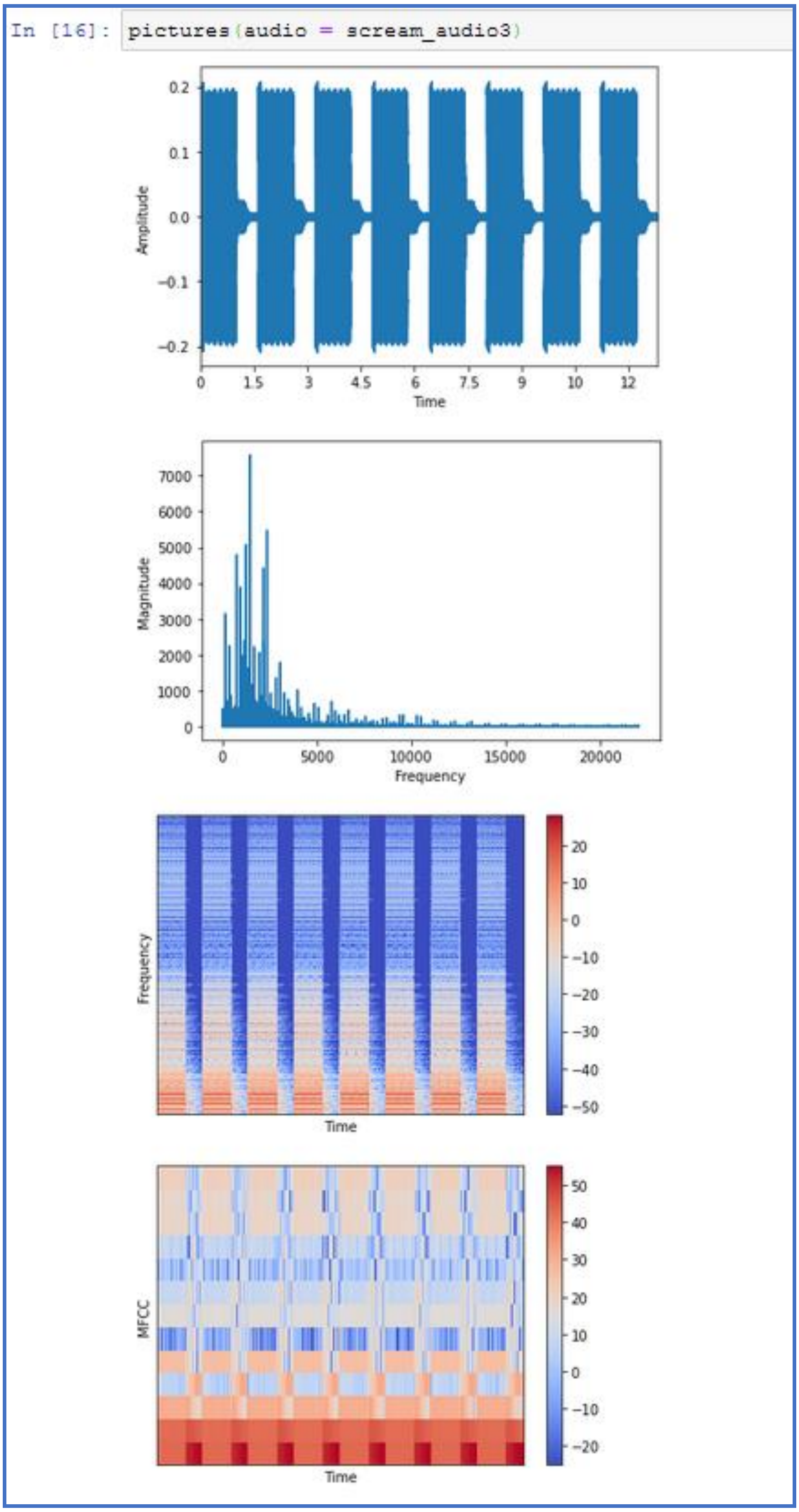


Figure 28: Siren

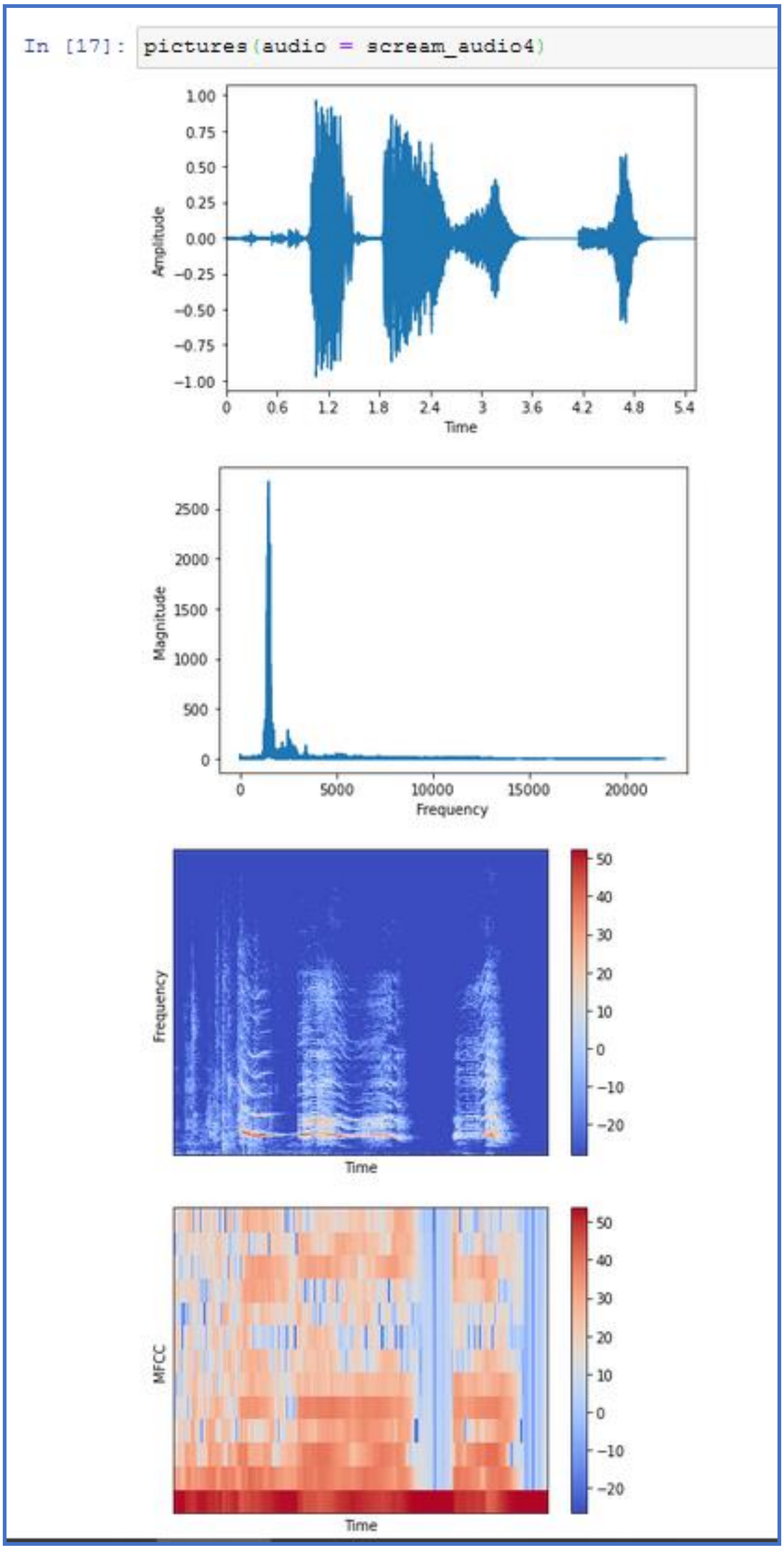


Figure 29: Screaming

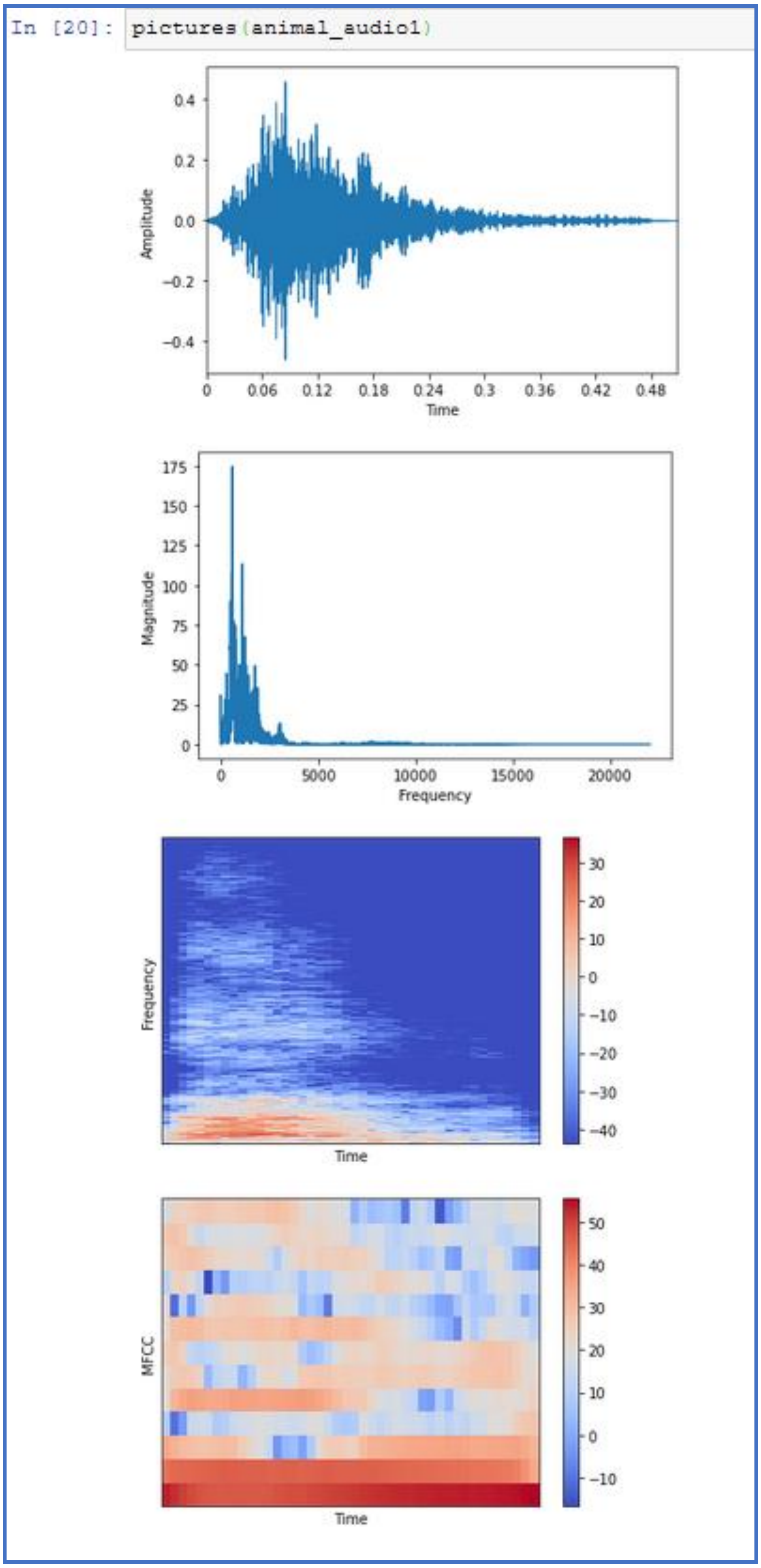


Figure 30: Dog

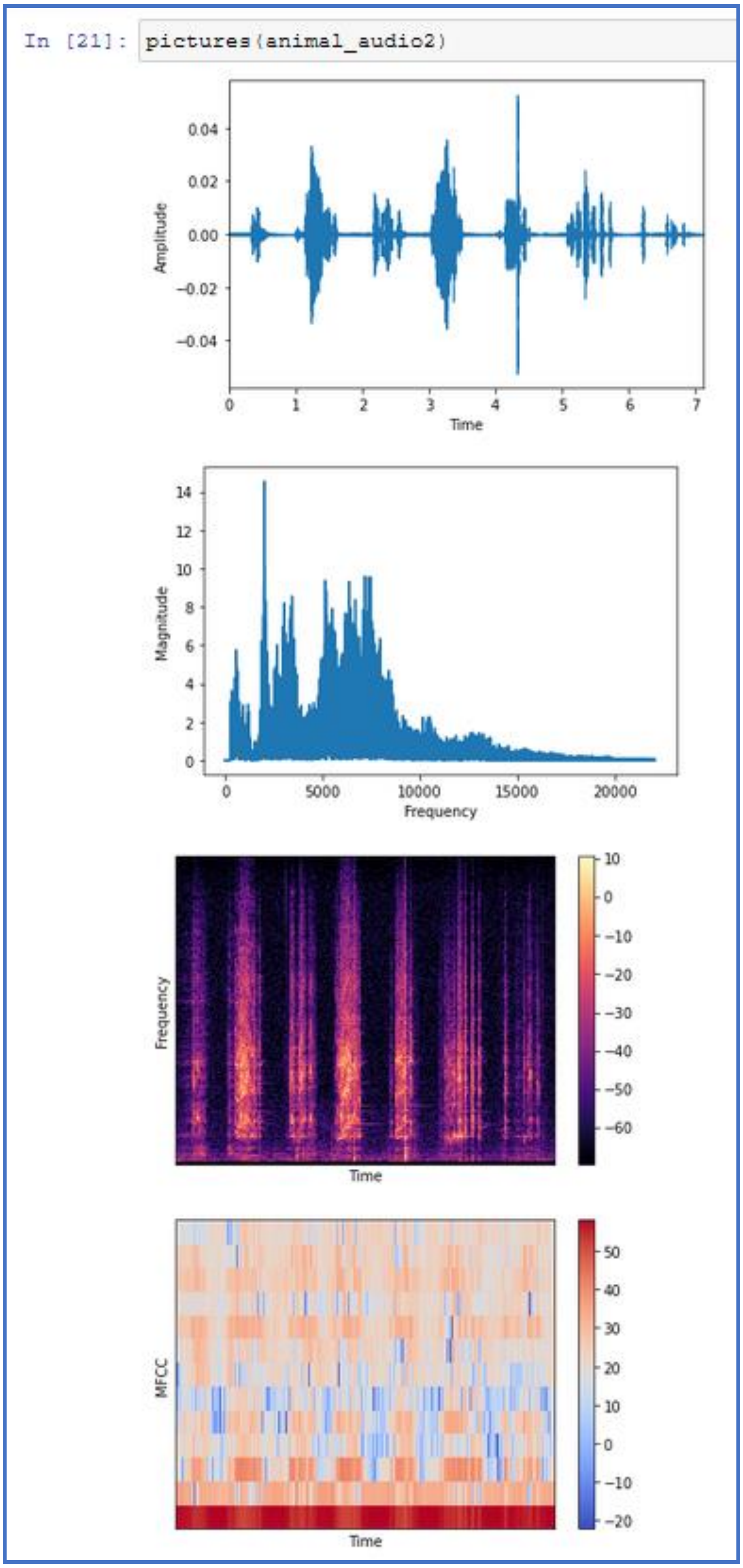


Figure 31: Wild animals

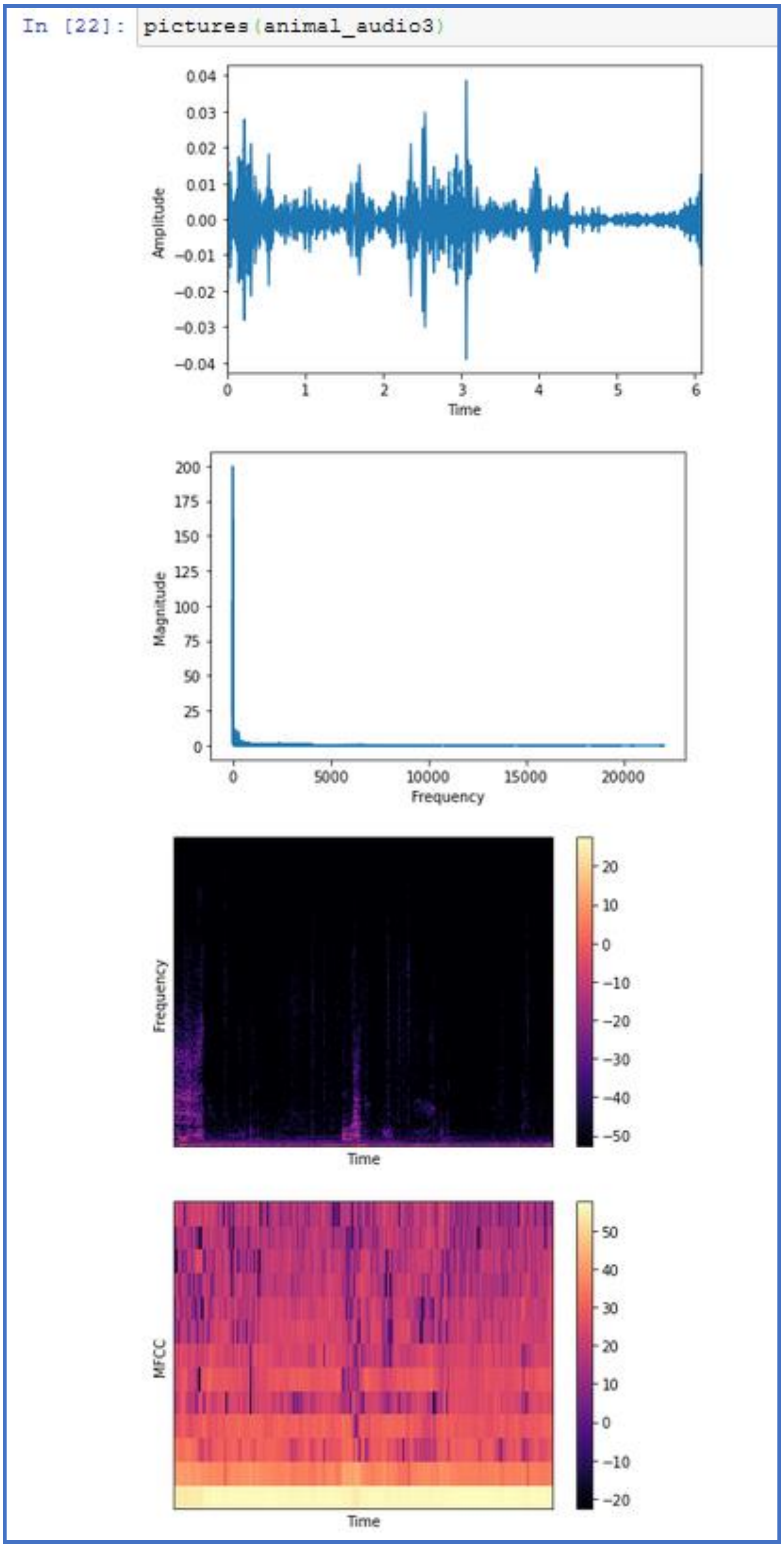


Figure 32: Cat

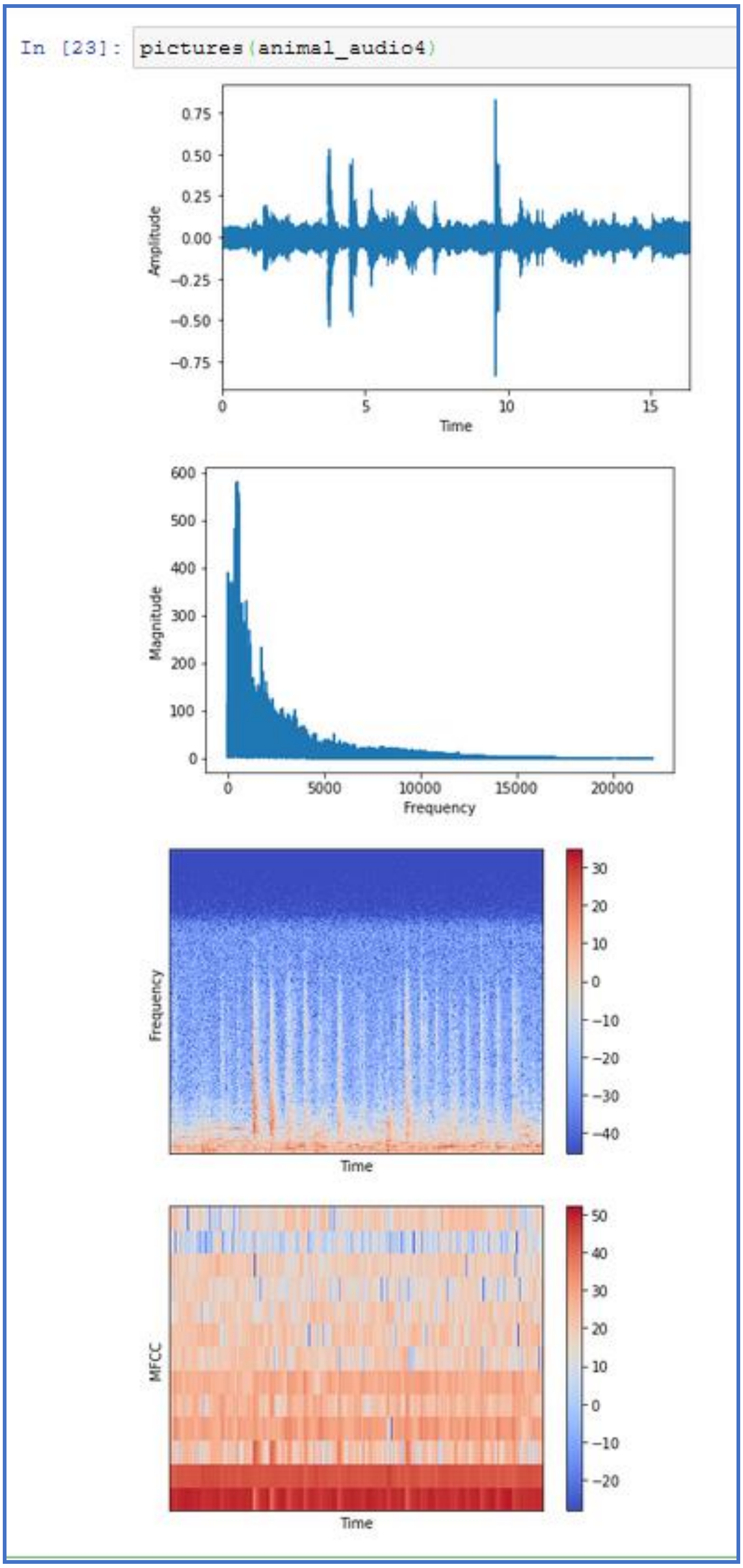


Figure 33: Livestock

5 Data Pre-Processing

This section is the beginning of processes leading to the classification and clustering. Audio cannot be directly fed into the model due to complexities. The activities done as part of pre processing prepares data to be fed into the CNNs.

5.1 Data Selection and pre-processing.

The data had to be grouped to meet the project's needs.

1. Labels from [FSD Annotator]() (Figure 34) were shortlisted manually to belong to distress and animal categories.

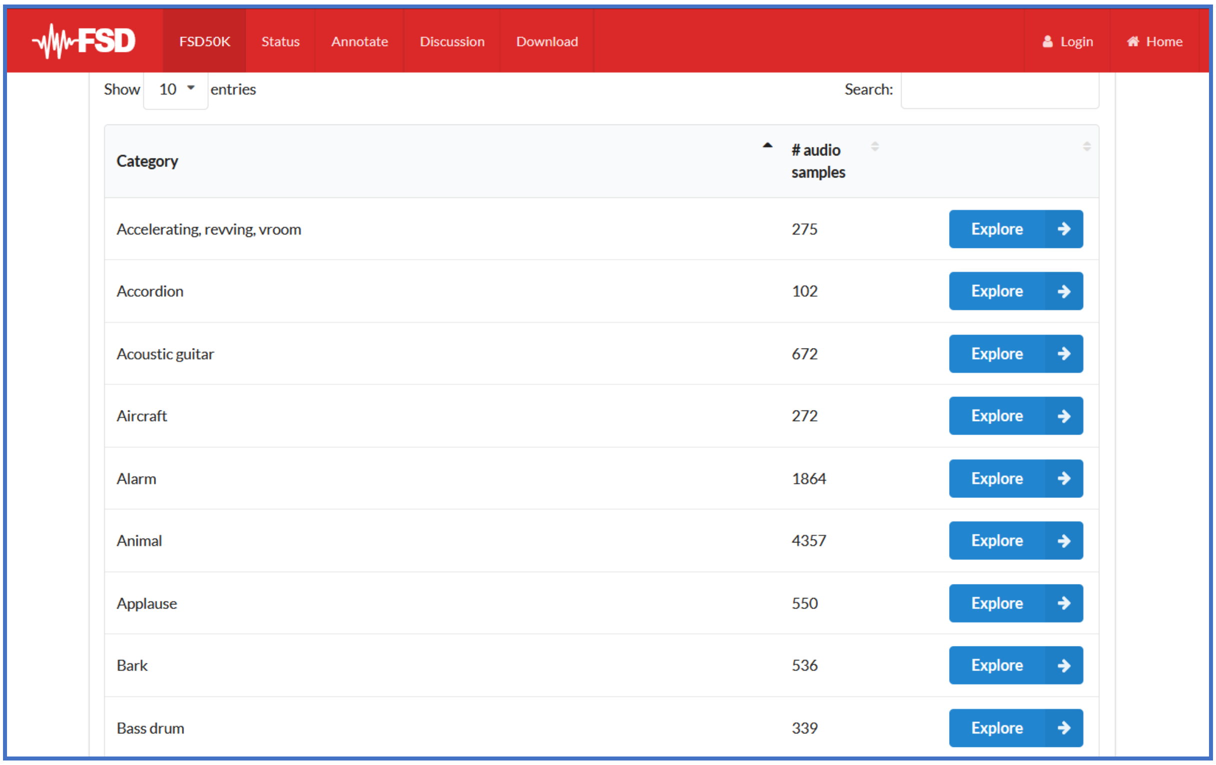


Figure 34: FSD Annotator

2. The required libraries are imported (Figure 35). *Librosa* is used for converting audio into graphical representations. *Soundfile* is used for saving the segmented audios into the hard disk location.

```
In [1]: import os
        import numpy as np
        import librosa
        import librosa.display
        import tensorflow as tf
        from tensorflow import keras
        import matplotlib.pyplot as plt
        import pandas as pd
        import soundfile as sf
```

Figure 35: Import libraries

3. In figure 36 the chosen labels are used to identify the audio files corresponding to distress labels. All audios under the chosen set of labels were marked as True and the rest False under the new key "*labels*". Figure 37 represents the output of the labelling.

```
In [15]: ABS_PATH = os.path.abspath(os.path.join('D:', '/Data Playground/Academics/Research Project'))
         ground_truth = pd.read_csv(ABS_PATH+'/FSD50K/FSD50K.ground_truth/dev.csv')

In [3]: # label distress labels to True and the rest to False

        red_labels = {'Alarm', 'Gasp', 'Crying_and_sobbing', 'Hiss',
                      'Screech', 'Shatter', 'Yell', 'Siren', 'Sigh',
                      'Screaming', 'Growling', 'Shout', 'Boom', 'Crackle', 'Explosion'}
        pattern = '|'.join(red_labels)


        value = ground_truth.labels.str.contains(pattern)
        value
        new_GT = pd.merge(ground_truth, value, right_index=True, left_index=True)
        new_GT.columns = ['fname', 'grouped_labels', 'mids', 'split', 'labels']
        new_GT
```

Figure 36: Selected distress labels and Creation of a pandas dataframe with the Information

Out[3]:

	fname	grouped_labels	mids	split	labels
0	64760	Electric_guitar,Guitar,Plucked_string_instrume...	/m/02sgy,/m/0342h,/m/0fx80y,/m/04szw,/m/04rlf	train	False
1	16399	Electric_guitar,Guitar,Plucked_string_instrume...	/m/02sgy,/m/0342h,/m/0fx80y,/m/04szw,/m/04rlf	train	False
2	16401	Electric_guitar,Guitar,Plucked_string_instrume...	/m/02sgy,/m/0342h,/m/0fx80y,/m/04szw,/m/04rlf	train	False
3	16402	Electric_guitar,Guitar,Plucked_string_instrume...	/m/02sgy,/m/0342h,/m/0fx80y,/m/04szw,/m/04rlf	train	False
4	16404	Electric_guitar,Guitar,Plucked_string_instrume...	/m/02sgy,/m/0342h,/m/0fx80y,/m/04szw,/m/04rlf	train	False
...	...	...	...	...	...
40961	102863	Fowl,Livestock_and_farm_animals_and_working_an...	/m/025rv6n,/m/0ch8v,/m/0jbk	train	False
40962	389607	Fowl,Livestock_and_farm_animals_and_working_an...	/m/025rv6n,/m/0ch8v,/m/0jbk	train	False
40963	90091	Fowl,Livestock_and_farm_animals_and_working_an...	/m/025rv6n,/m/0ch8v,/m/0jbk	train	False
40964	244718	Fowl,Livestock_and_farm_animals_and_working_an...	/m/025rv6n,/m/0ch8v,/m/0jbk	train	False
40965	24061	Fowl,Livestock_and_farm_animals_and_working_an...	/m/025rv6n,/m/0ch8v,/m/0jbk	train	False

40966 rows × 5 columns

```
In [4]: new_GT.groupby(['labels']).size()

Out[4]: labels
        False    36962
        True      4004
        dtype: int64
```

Figure 37: New dataframe with "labels" key

4. Create folders to store audios that are split into 3s segments (Figure 38).

```
In [16]: DATA_PATH = ABS_PATH + '/FSD50K/FSD50K.dev_audio/'
         SAMPLE_RATE = 44100

In [11]: os.makedirs(ABS_PATH+'/SplitAudio')
         os.makedirs(ABS_PATH+'/SplitAudio/train')
         os.makedirs(ABS_PATH+'/SplitAudio/test')
         os.makedirs(ABS_PATH+'/SplitAudio/train/Danger')
         os.makedirs(ABS_PATH+'/SplitAudio/train/Other')
         os.makedirs(ABS_PATH+'/SplitAudio/test/Danger')
         os.makedirs(ABS_PATH+'/SplitAudio/test/Other')
```

Figure 38: Creating new folders

5. Figure 39 displays the functions written to slice and store clippings to the appropriate folders conditionally.

```
In [6]: def store_wav(data, is_train, is_distressed, file_name, file_identifier):
            if is_train:
                if is_distressed:
                    save_location = TRAIN_PATH + 'Danger/' + file_name + str(file_identifier) + '.wav'
                elif not is_distressed:
                    save_location = TRAIN_PATH + 'Other/' + file_name + str(file_identifier) + '.wav'
            elif not is_train:
                if is_distressed:
                    save_location = TEST_PATH + 'Danger/' + file_name + str(file_identifier) +'.wav'
                elif not is_distressed:
                    save_location = TEST_PATH + 'Other/' + file_name + str(file_identifier) +'.wav'

            sf.write(file=save_location, data=data, samplerate=SAMPLE_RATE)

In [7]:
        def slice_data(start, end, raw_data,  sample_rate):
            max_ind = len(raw_data)
            start_ind = min(int(start * sample_rate), max_ind)
            end_ind = min(int(end * sample_rate), max_ind)
            return raw_data[start_ind: end_ind]
```

Figure 39. Functions for storing audio files and slicing files

6. Figure 40 shows the constants set to clip audios. CLIP_LEN decides that 3s segments are to be created.

```
In [17]: TRAIN_PATH = ABS_PATH + '/SplitAudio/train/'
         TEST_PATH = ABS_PATH + '/SplitAudio/test/'
         CLIP_LEN = 3
         a_len = 44100 * CLIP_LEN
```

Figure 40: Constants

7. Code in figure 41 splits the audio into 3s segments. Silence is added to the remaining portion of an audio to make it a 3s segment. Names are appended with the sequence numbers to differentiate the segments.

```
In [10]: for i, (dirpath, dirnames,  filenames) in enumerate(os.walk(DATA_PATH)):
             #For each file in directory
             for f in filenames:
                 file_path = os.path.join(dirpath, f)
                 signal, sr = librosa.load(file_path, sr=SAMPLE_RATE)
                 duration = librosa.get_duration(y=signal, sr=SAMPLE_RATE)
                 start = 0
                 end = CLIP_LEN
                 file_identifier = 0
                 #Removing too small & too large files
                 if(duration<=30 and duration>=3):
                     file_name = f.replace(".wav", "")
                     #Locating meta data
                     row_focus = new_GT.loc[new_GT['fname'] == int(file_name)]
                     is_distressed = row_focus.iloc[0]['labels']
                     is_train = row_focus.iloc[0]['split'] == 'train'
                     #Creating and storing 3s clippings
                     while end<duration:
                         sliced_data = slice_data(start=start, end=end, raw_data=signal, sample_rate=44100)
                         start = end
                         end = end + CLIP_LEN
                         store_wav(sliced_data, is_train, is_distressed, file_name, file_identifier)
                         file_identifier = file_identifier + 1
                     #Adding silence padding to the shorter final clipping
                     if (duration-end) >= (CLIP_LEN/2):
                         end = duration
                         sliced_data = slice_data(start=start, end=end, raw_data=signal, sample_rate=44100)
                         padded_data = librosa.util.pad_center(sliced_data, a_len)
                         store_wav(padded_data, is_train, is_distressed, file_name, file_identifier)
```

Figure 41: Splitting audios to 3s segments

8. Create directories for storing the spectrograms from the audio snippets (Figure 42).

```
In [42]: os.makedirs(ABS_PATH+'/MelSpectrograms')
         os.makedirs(ABS_PATH+'/MelSpectrograms/train')
         os.makedirs(ABS_PATH+'/MelSpectrograms/test')
         os.makedirs(ABS_PATH+'/MelSpectrograms/train/Danger')
         os.makedirs(ABS_PATH+'/MelSpectrograms/train/Other')
         os.makedirs(ABS_PATH+'/MelSpectrograms/test/Danger')
         os.makedirs(ABS_PATH+'/MelSpectrograms/test/Other')
```

Figure 42: Creating folders

9. This function converts audio into spectrograms images and stores as PNG files.

```
In [18]: def save_mel_spectrogram(cmap_type, path, class_path, save_loc):
             for i, (dirpath, dirnames,  filenames) in enumerate(os.walk(path + '/' + class_path)):
                 for f in filenames:
                     file_path = os.path.join(dirpath, f)
                     signal, sr = librosa.load(file_path, sr=SAMPLE_RATE)
                     duration = librosa.get_duration(y=signal, sr=SAMPLE_RATE)
                     if duration == CLIP_LEN:
                         plt.figure(figsize=(10,3))
                         stft = librosa.stft(signal)
                         log_spectrogram = librosa.amplitude_to_db(abs(stft))
                         librosa.display.specshow(log_spectrogram, sr=SAMPLE_RATE, cmap=cmap_type, x_axis='time', y_axis='hz')
                         plt.ylim(0, 10000)
                         file_name = f.replace(".wav", "")
                         plt.savefig(save_loc + file_name + '.png')
                         plt.clf()
                         plt.close(plt.gcf())
```

Figure 43: Function to create spectrograms.

10. The function in figure 43 is called to create all required inputs (figure 44).

```
In [6]: save_mel_spectrogram(cmap_type = 'magma', path = TRAIN_PATH, class_path='Danger',
                             save_loc=ABS_PATH+'/MelSpectrograms/train/Danger/')

In [19]: save_mel_spectrogram(cmap_type = 'magma', path = TRAIN_PATH, class_path='Other',
                             save_loc=ABS_PATH+'/MelSpectrograms/train/Other/')

In [ ]: save_mel_spectrogram(cmap_type = 'magma', path = TEST_PATH, class_path='Other',
                             save_loc=ABS_PATH+'/MelSpectrograms/test/Other/')

In [ ]: save_mel_spectrogram(cmap_type = 'magma', path = TEST_PATH, class_path='Danger',
                             save_loc=ABS_PATH+'/MelSpectrograms/test/Danger/')
```

Figure 44: Creating spectrograms

11. Figure 45 represents a sample folder which stores the created images.

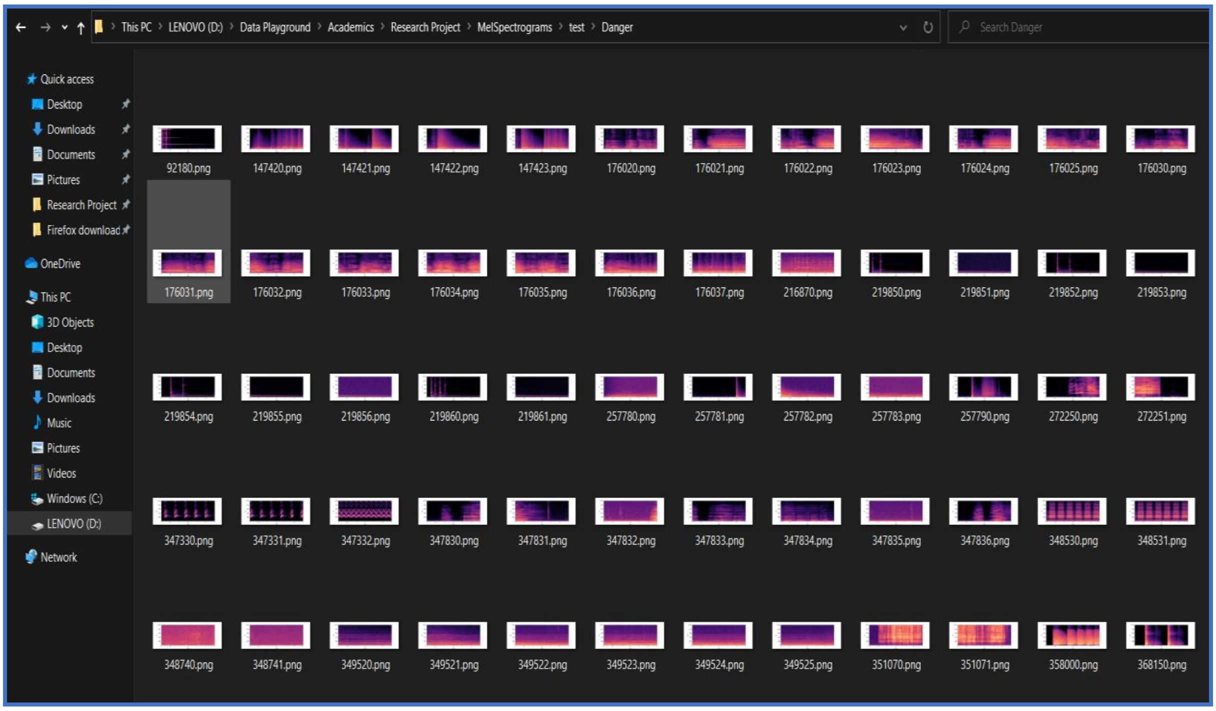


Figure 45: Spectrograms folder

12. Similarly animal labels are chosen and spectrograms are made (figure 46).

In [3]:
```
# label animal labels to True and the rest to False

animal_labels = {'Bark', 'Animal', 'Cat', 'Dog', 'pets', 'Purr',
                 'wild', 'domestic_animals', 'farm_animals', 'Meow', 'working_animals', 'livestock' }
pattern = '|'.join(animal_labels)


value = ground_truth.labels.str.contains(pattern)
value
new_GT = pd.merge(ground_truth, value, right_index=True, left_index=True)
new_GT.columns = ['fname', 'grouped_labels', 'mids', 'split', 'labels']
new_GT
```

Out[3]:

	fname	grouped_labels	mids	split	labels
0	64760	Electric_guitar,Guitar,Plucked_string_instrume...	/m/02sgy,/m/0342h,/m/0fx80y,/m/04szw,/m/04rlf	train	False
1	16399	Electric_guitar,Guitar,Plucked_string_instrume...	/m/02sgy,/m/0342h,/m/0fx80y,/m/04szw,/m/04rlf	train	False
2	16401	Electric_guitar,Guitar,Plucked_string_instrume...	/m/02sgy,/m/0342h,/m/0fx80y,/m/04szw,/m/04rlf	train	False
3	16402	Electric_guitar,Guitar,Plucked_string_instrume...	/m/02sgy,/m/0342h,/m/0fx80y,/m/04szw,/m/04rlf	train	False
4	16404	Electric_guitar,Guitar,Plucked_string_instrume...	/m/02sgy,/m/0342h,/m/0fx80y,/m/04szw,/m/04rlf	train	False
...	...	...	...	...	...
40961	102863	Fowl,Livestock_and_farm_animals_and_working_an...	/m/025rv6n,/m/0ch8v,/m/0jbk	train	True
40962	389607	Fowl,Livestock_and_farm_animals_and_working_an...	/m/025rv6n,/m/0ch8v,/m/0jbk	train	True
40963	90091	Fowl,Livestock_and_farm_animals_and_working_an...	/m/025rv6n,/m/0ch8v,/m/0jbk	train	True
40964	244718	Fowl,Livestock_and_farm_animals_and_working_an...	/m/025rv6n,/m/0ch8v,/m/0jbk	train	True
40965	24061	Fowl,Livestock_and_farm_animals_and_working_an...	/m/025rv6n,/m/0ch8v,/m/0jbk	train	True

40966 rows × 5 columns

In [4]:
```
new_GT.groupby(['labels']).size()
```

Out[4]:
```
labels
False    37691
True      3275
dtype: int64
```

Figure 46: Animal data selection and labelling

6 Classification of Distress and Non-distress Sounds Using VGG-16 and MobileNetV2 Architectures.

VGG-16 and MobileNetV2 are the 2 CNN networks used in this project. The following links provide more information about these architectures.

VGG-16 - Keras

MobileNetV2 - Keras

MobileNetV2 – Research Preprint

The following steps provide information on how the training processes are done.

6.1 Training Using VGG-16

1. Import libraries such as TensorFlow and Keras (figure 47).

```
In [1]:  from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.applications.vgg16 import preprocess_input
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.layers import Dense, Conv2D, MaxPool2D , Flatten
from tensorflow.keras.losses import categorical_crossentropy
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
from tensorflow.keras import metrics

from sklearn.utils import class_weight
from collections import Counter

import matplotlib.pyplot as plt

from os import listdir
from os.path import isfile, join

import pandas as pd
```

Figure 47: Libraries import

2. In figure 48, the code uses Keras's ImageDataGenerator to resize the image to set it to the CNN's input dimensional requirements.

```
In [ ]: train_loc = 'D:/Data Playground/Academics/Research Project/MelSpectrograms/train'
        test_loc = 'D:/Data Playground/Academics/Research Project/MelSpectrograms/test'

In [ ]: trdata = ImageDataGenerator()
        traindata = trdata.flow_from_directory(directory=train_loc, target_size=(224,224))
        tsdata = ImageDataGenerator()
        testdata = tsdata.flow_from_directory(directory=test_loc, target_size=(224,224))
```

Figure 48: Image transformation

3. The VGG-16 is loaded using code used in figure 49. Extra dense layers are added. The resultant model is displayed using model.summary in figure 50.

```
In [6]: vgg16 = VGG16(weights='imagenet')
        vgg16.summary()

        x  = vgg16.get_layer('fc2').output
        prediction = Dense(2, activation='softmax', name='predictions')(x)

        model = Model(inputs=vgg16.input, outputs=prediction)
```

Figure 49: VGG 16 configuration

```
Model: "vgg16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 224, 224, 3)]     0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
fc1 (Dense)                  (None, 4096)              102764544
_________________________________________________________________
fc2 (Dense)                  (None, 4096)              16781312
_________________________________________________________________
predictions (Dense)          (None, 1000)              4097000
=================================================================
Total params: 138,357,544
Trainable params: 138,357,544
Non-trainable params: 0
```

Figure 50: Modified VGG-16 model for classification

4. All the layers except the dense layers are set to be untrainable (figure 51).

```
In [ ]: for layer in model.layers:
            layer.trainable = False

        for layer in model.layers[-20:]:
            layer.trainable = True
            print("Layer '%s' is trainable" % layer.name)
```

Figure 51: Setting trainable attribute of layers

5. In figure 52, the code used for setting optimizer, loss, metrics, checkpoints, and early stopping criteria are displayed. The class weights are calculated, and the model is run (figure 53).

```
In [ ]: opt = Adam(lr=0.000001)
        model.compile(optimizer=opt, loss='binary_crossentropy',
                      metrics=['accuracy', 'mae'])
        model.summary()

In [ ]: checkpoint = ModelCheckpoint("vgg16_base_res.h5", monitor='val_accuracy', verbose=1,
                                     save_best_only=True, save_weights_only=False, mode='auto')
        early = EarlyStopping(monitor='val_accuracy', min_delta=0, patience=20, verbose=1, mode='auto')

In [ ]: counter = Counter(traindata.classes)
        max_val = float(max(counter.values()))
        class_weights = {class_id : max_val/num_images for class_id, num_images in counter.items()}
        class_weights

In [ ]: hist = model.fit(traindata, steps_per_epoch=traindata.samples//traindata.batch_size, validation_data=testdata,
                         class_weight=class_weights, validation_steps=testdata.samples//testdata.batch_size,
                         epochs=110,callbacks=[checkpoint,early])
```

Figure 52: Adding hyperparameters and training options.

```
In [*]: hist = model.fit(traindata, steps_per_epoch=traindata.samples//traindata.batch_size, validation_data=testdata,
                         class_weight=class_weights, validation_steps=testdata.samples//testdata.batch_size,
                         epochs=110,callbacks=[checkpoint,early])

        Epoch 1/110
        2079/2079 [==============================] - 624s 297ms/step - loss: 0.5179 - accuracy: 0.8218 - mae: 0.1747 - val_loss: 1.5647
        - val_accuracy: 0.2106 - val_mae: 0.7395

        Epoch 00001: val_accuracy improved from -inf to 0.21065, saving model to vgg16_base_res.h5
        Epoch 2/110
        2079/2079 [==============================] - 328s 158ms/step - loss: 0.3351 - accuracy: 0.8830 - mae: 0.1152 - val_loss: 1.3541
        - val_accuracy: 0.3192 - val_mae: 0.6506

        Epoch 00002: val_accuracy improved from 0.21065 to 0.31916, saving model to vgg16_base_res.h5
        Epoch 3/110
        2079/2079 [==============================] - 372s 179ms/step - loss: 0.2752 - accuracy: 0.9072 - mae: 0.0932 - val_loss: 0.9916
        - val_accuracy: 0.5131 - val_mae: 0.4871

        Epoch 00003: val_accuracy improved from 0.31916 to 0.51312, saving model to vgg16_base_res.h5
        Epoch 4/110
        2079/2079 [==============================] - 438s 211ms/step - loss: 0.2185 - accuracy: 0.9319 - mae: 0.0709 - val_loss: 1.1067
        - val_accuracy: 0.5182 - val_mae: 0.4764

        Epoch 00004: val_accuracy improved from 0.51312 to 0.51823, saving model to vgg16_base_res.h5
        Epoch 5/110
         673/2079 [========>.....................] - ETA: 4:58 - loss: 0.1546 - accuracy: 0.9519 - mae: 0.0504
```

Figure 53: Training process

6. The training was run for 25 epochs and the training plots are generated from the attributs of “*hist*” variable (figure 54).

```
In [12]: acc = hist.history['accuracy']
         val_acc = hist.history['val_accuracy']

         loss = hist.history['loss']
         val_loss = hist.history['val_loss']

         plt.figure(figsize=(8, 8))
         plt.subplot(2, 1, 1)
         plt.plot(acc, label='Training Accuracy')
         plt.plot(val_acc, label='Validation Accuracy')
         plt.legend(loc='lower right')
         plt.ylabel('Accuracy')
         plt.ylim([min(plt.ylim()),1])
         plt.title('Training and Validation Accuracy')

         plt.subplot(2, 1, 2)
         plt.plot(loss, label='Training Loss')
         plt.plot(val_loss, label='Validation Loss')
         plt.legend(loc='upper right')
         plt.ylabel('Cross Entropy')
         plt.ylim([0,max(plt.ylim())])
         plt.title('Training and Validation Loss')
         plt.show()
```

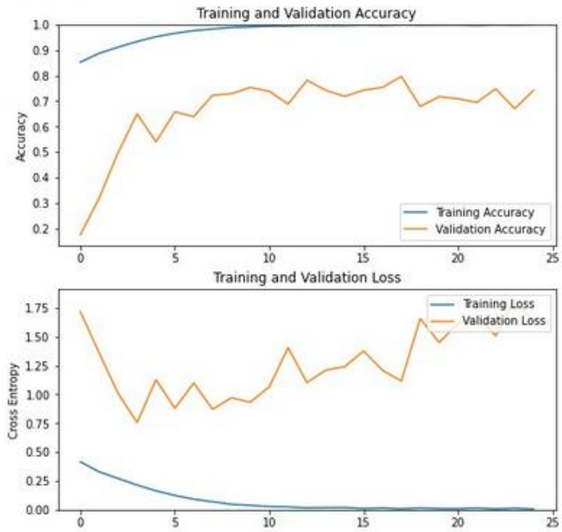


Figure 54: Training plots

6.2 Training Using MobileNetV2

This section explains how the training is done using the MobileNetV2 architecture.

1. The model is loaded from Keras architecture and custom top layers are added to meet our needs (figure 55).

```
In [4]: from keras.applications import MobileNetV2
        mobilenet = MobileNetV2(include_top=False, input_shape=(224, 224, 3),weights="imagenet")

In [6]: mobilenet.trainable = False

In [26]: model = Sequential([mobilenet,
                             GlobalAveragePooling2D(),
                             Dense(2, activation='softmax')])

In [27]: model.summary()

         Model: "sequential_1"
         _________________________________________________________________
         Layer (type)                 Output Shape              Param #
         =================================================================
         mobilenetv2_1.00_224 (Functi (None, 7, 7, 1280)        2257984
         _________________________________________________________________
         global_average_pooling2d_1 ( (None, 1280)              0
         _________________________________________________________________
         dense_1 (Dense)              (None, 2)                 2562
         =================================================================
         Total params: 2,260,546
         Trainable params: 2,562
         Non-trainable params: 2,257,984
         _________________________________________________________________
```

Figure 55: Loading MobileNetV2 and Modification

2. The model hyperparameters are added and it is run (figure 56 and 57).

```
In [28]: checkpoint = ModelCheckpoint("mobilenet_base_res.h5", monitor='val_accuracy', verbose=1,
                                      save_best_only=True, save_weights_only=False, mode='auto')
         early = EarlyStopping(monitor='val_accuracy', min_delta=0, patience=20, verbose=1, mode='auto')

In [29]: counter = Counter(traindata.classes)
         max_val = float(max(counter.values()))
         class_weights = {class_id : max_val/num_images for class_id, num_images in counter.items()}
         class_weights

Out[29]: {0: 1.5258813413585555, 1: 1.0}

In [30]: model.compile(optimizer=Adam(lr=0.000001),
                       loss='binary_crossentropy',
                       metrics=['accuracy', 'mae'])
```

Figure 56: Hyper parameter addition

```
In [*]:  hist = model.fit(traindata, steps_per_epoch=traindata.samples//traindata.batch_size, validation_data=testdata,
                    class_weight=class_weights, validation_steps=testdata.samples//testdata.batch_size,
                    epochs=25,callbacks=[checkpoint,early])

Epoch 1/25
2079/2079 [==============================] - 300s 143ms/step - loss: 1.5047 - accuracy: 0.1146 - mae: 0.7716 - val_loss: 0.9
168 - val_accuracy: 0.1975 - val_mae: 0.6530

Epoch 00001: val_accuracy improved from -inf to 0.19753, saving model to mobilenet_base_res.h5
Epoch 2/25
2079/2079 [==============================] - 296s 142ms/step - loss: 1.3537 - accuracy: 0.2064 - mae: 0.6375 - val_loss: 0.7
918 - val_accuracy: 0.2877 - val_mae: 0.5680

Epoch 00002: val_accuracy improved from 0.19753 to 0.28771, saving model to mobilenet_base_res.h5
Epoch 3/25
2079/2079 [==============================] - 295s 142ms/step - loss: 1.2952 - accuracy: 0.3179 - mae: 0.5501 - val_loss: 0.7
499 - val_accuracy: 0.3641 - val_mae: 0.5339

Epoch 00003: val_accuracy improved from 0.28771 to 0.36410, saving model to mobilenet_base_res.h5
Epoch 4/25
2079/2079 [==============================] - 295s 142ms/step - loss: 1.2600 - accuracy: 0.4601 - mae: 0.5034 - val_loss: 0.7
395 - val_accuracy: 0.4062 - val_mae: 0.5247

Epoch 00004: val_accuracy improved from 0.36410 to 0.40615, saving model to mobilenet_base_res.h5
Epoch 5/25
1191/2079 [================>.............] - ETA: 1:54 - loss: 1.2415 - accuracy: 0.5791 - mae: 0.4808
```

Figure 57: Training process

3. The training plots are generated from the history variable (figure 58).

```
In [16]:  acc = hist.history['accuracy']
          val_acc = hist.history['val_accuracy']

          loss = hist.history['loss']
          val_loss = hist.history['val_loss']

          plt.figure(figsize=(8, 8))
          plt.subplot(2, 1, 1)
          plt.plot(acc, label='Training Accuracy')
          plt.plot(val_acc, label='Validation Accuracy')
          plt.legend(loc='lower right')
          plt.ylabel('Accuracy')
          plt.ylim([min(plt.ylim()),1])
          plt.title('Training and Validation Accuracy')

          plt.subplot(2, 1, 2)
          plt.plot(loss, label='Training Loss')
          plt.plot(val_loss, label='Validation Loss')
          plt.legend(loc='upper right')
          plt.ylabel('Cross Entropy')
          plt.ylim([0,max(plt.ylim())])
          plt.title('Training and Validation Loss')
          plt.show()
```

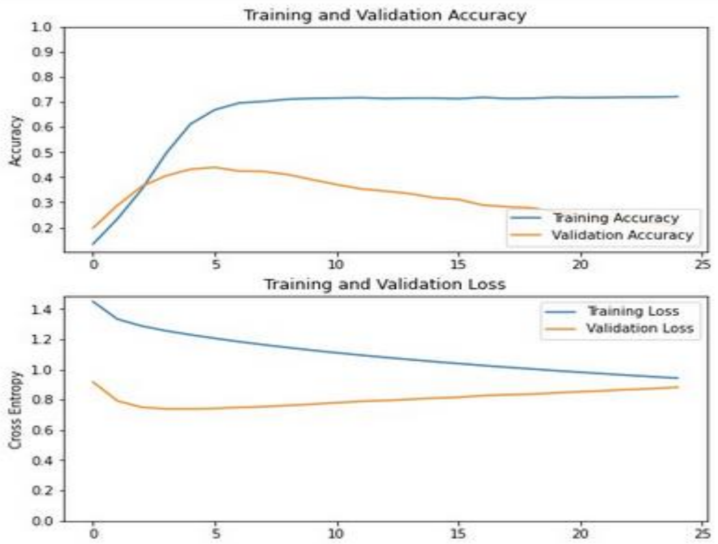


Figure 58: Training plots

7 K-Means Clustering of Animal Sounds Using VGG-16 and MobileNetV2 Feature extractors.

The standard VGG-16 and MobileNetV2 architectures are pitted against the target trained modified ones from section 6 to know if the special training has increased the ability to clustering.

1. Import libraries and set the path constants (figure 59).

```
In [1]: import numpy as np
        import tensorflow as tf
        import matplotlib.pyplot as plt
        from sklearn.cluster import KMeans
        from sklearn.metrics import silhouette_score
        import cv2
        import os, glob, shutil
        from tensorflow.keras.models import Model, Sequential

In [18]: ABS_PATH = os.path.abspath(os.path.join('D:', '/Data Playground/Academics/Research Project/Animaldir/Animals1'))
         CLUSTER_PATH = os.path.abspath(os.path.join('D:', '/Data Playground/Academics/Research Project/Clusters'))
```

Figure 59: Library imports and constants

2. Get images from the directory path and pre-process to be fed into the network (figure 60).

```
In [3]: glob_dir = ABS_PATH + '/*.png'
        images = [cv2.resize(cv2.imread(file), (224, 224)) for file in
                  glob.glob(glob_dir)]

        paths = [file for file in glob.glob(glob_dir)]
        images = np.array(np.float32(images).reshape(len(images), -1)/255)
```

Figure 60: Pre processing

3. Load standard MobileNetV2 and extract features to be clustered (figure 61).

```
In [10]: model = tf.keras.applications.MobileNetV2(include_top=False,
         weights="imagenet", input_shape=(224, 224, 3))

         predictions = model.predict(images.reshape(-1, 224, 224, 3))
         pred_images = predictions.reshape(images.shape[0], -1)
```

Figure 61: Feature extraction using standard MobileNetV2

4. Silhouette scores are calculated using both Euclidean and Cosine distance metric (figure 62 and 63).

```
In [13]: sil = []
kl = []
kmax = 10
for k in range(2, kmax+1):
    kmeans2 = KMeans(n_clusters = k).fit(pred_images)
    labels = kmeans2.labels_
    sil.append(silhouette_score(pred_images, labels, metric =   "euclidean"))
    kl.append(k)
plt.plot(kl, sil)
plt.ylabel("Silhouette Score")
plt.xlabel("k")
plt.show()
```

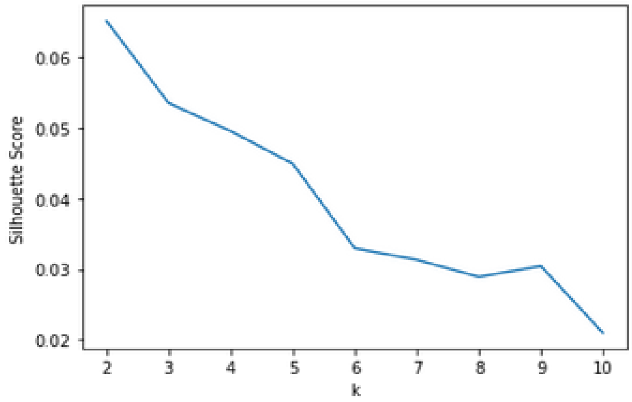


Figure 62: Silhouette score - Euclidean - Standard MobilenetV2

```
In [14]: sil = []
kl = []
kmax = 10
for k in range(2, kmax+1):
    kmeans2 = KMeans(n_clusters = k).fit(pred_images)
    labels = kmeans2.labels_
    sil.append(silhouette_score(pred_images, labels, metric =   "cosine"))
    kl.append(k)
plt.plot(kl, sil)
plt.ylabel("Silhouette Score")
plt.xlabel("K")
plt.show()
```

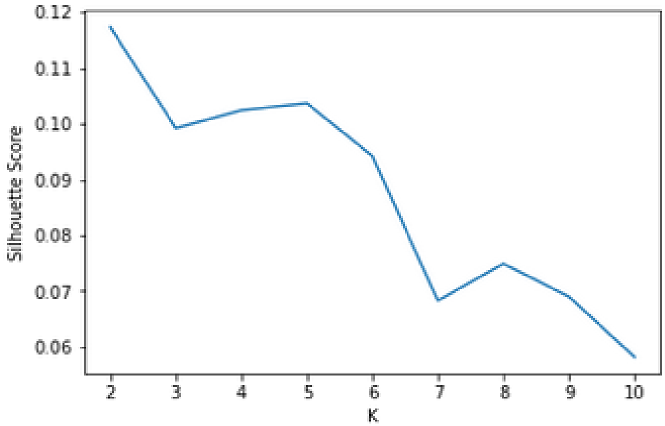


Figure 63: Silhouette score - Cosine - Standard MobilenetV2

5. The same process is done using the modified version of MobilenetV2 (figure 64 – 66).

```
In [4]: distress_model = tf.keras.models.load_model('mobilenet_base_res.h5')

In [5]: distress_model.summary()

        Model: "sequential"
        _________________________________________________________________
        Layer (type)                 Output Shape              Param #
        =================================================================
        mobilenetv2_1.00_224 (Functi (None, 7, 7, 1280)        2257984
        _________________________________________________________________
        global_average_pooling2d (Gl (None, 1280)              0
        _________________________________________________________________
        dense (Dense)                (None, 2)                 2562
        =================================================================
        Total params: 2,260,546
        Trainable params: 2,562
        Non-trainable params: 2,257,984
        _________________________________________________________________

In [6]: targetted_model = Sequential()
        for layer in distress_model.layers[:-1]: # go through until last layer
            targetted_model.add(layer)
        targetted_model.summary()

        Model: "sequential"
        _________________________________________________________________
        Layer (type)                 Output Shape              Param #
        =================================================================
        mobilenetv2_1.00_224 (Functi (None, 7, 7, 1280)        2257984
        _________________________________________________________________
        global_average_pooling2d (Gl (None, 1280)              0
        =================================================================
        Total params: 2,257,984
        Trainable params: 0
        Non-trainable params: 2,257,984
        _________________________________________________________________
```

Figure 64: Loading target trained model created in section 6.

```
In [7]: predictions = targetted_model.predict(images.reshape(-1, 224, 224, 3))
        pred_images = predictions.reshape(images.shape[0], -1)
```

```
In [8]: sil = []
        kl = []
        kmax = 10
        for k in range(2, kmax+1):
            kmeans2 = KMeans(n_clusters = k).fit(pred_images)
            labels = kmeans2.labels_
            sil.append(silhouette_score(pred_images, labels, metric =   "euclidean"))
            kl.append(k)
        plt.plot(kl, sil)
        plt.ylabel("Silhouette Score")
        plt.xlabel("K")
        plt.show()
```

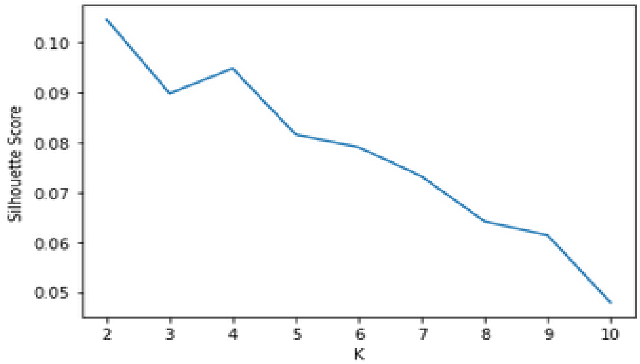


Figure 65: Silhouette score - Euclidean - Modified MobilenetV2

```
In [9]: sil = []
        kl = []
        kmax = 10
        for k in range(2, kmax+1):
            kmeans2 = KMeans(n_clusters = k).fit(pred_images)
            labels = kmeans2.labels_
            sil.append(silhouette_score(pred_images, labels, metric =   "cos
            kl.append(k)
        plt.plot(kl, sil)
        plt.ylabel("Silhouette Score")
        plt.xlabel("K")
        plt.show()
```

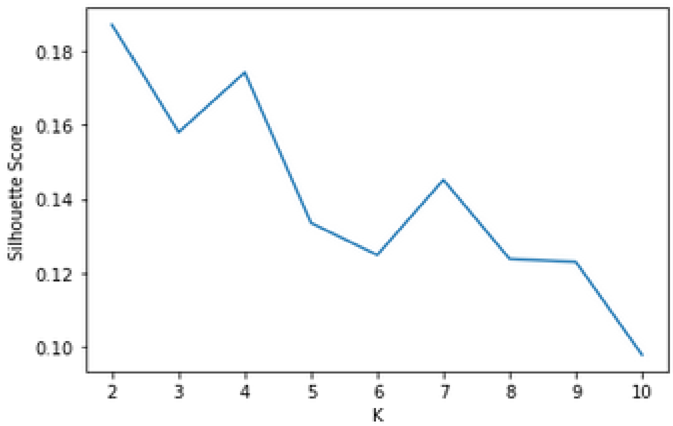


Figure 66: Silhouette score - Cosine - Modified MobilenetV2

6. The standard and modified versions of VGG-16 are loaded, and features are extracted. The Silhouette scores are calculated (figure 67 – 70) .

```
vgg = tf.keras.applications.VGG16(include_top=False,
weights="imagenet", input_shape=(224, 224, 3))
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_
kernels_notop.h5
58892288/58889256 [==============================] - 40s 1us/step
```

```
predictions = vgg.predict(images.reshape(-1, 224, 224, 3))
pred_images = predictions.reshape(images.shape[0], -1)
```

```
sil = []
kl = []
kmax = 10
for k in range(2, kmax+1):
    kmeans2 = KMeans(n_clusters = k).fit(pred_images)
    labels = kmeans2.labels_
    sil.append(silhouette_score(pred_images, labels, metric =   "euclidean"))
    kl.append(k)
plt.plot(kl, sil)
plt.ylabel("Silhouette Score")
plt.xlabel("K")
plt.show()
```

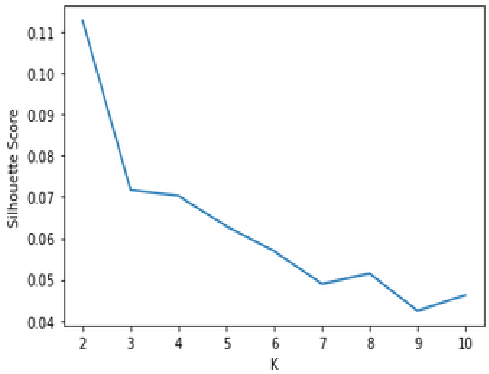


Figure 67: Silhouette score - Euclidean - Standard VGG-16

```
In [15]: sil = []
         kl = []
         kmax = 10
         for k in range(2, kmax+1):
             kmeans2 = KMeans(n_clusters = k).fit(pred_images)
             labels = kmeans2.labels_
             sil.append(silhouette_score(pred_images, labels, metric =   "cosine"))
             kl.append(k)
         plt.plot(kl, sil)
         plt.ylabel("Silhouette Score")
         plt.xlabel("K")
         plt.show()
```

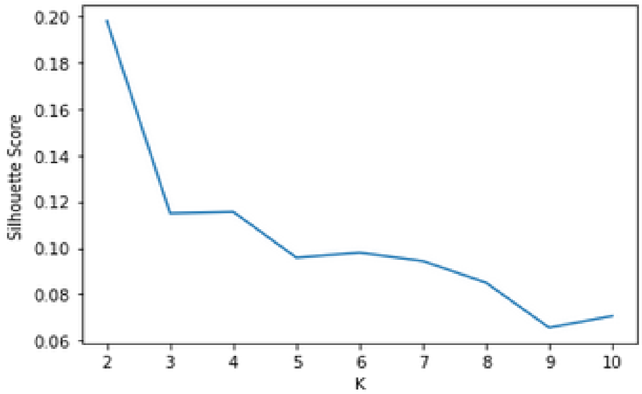


Figure 68: Silhouette score - Cosine - Standard VGG-16

```
In [15]: distress_model = tf.keras.models.load_model('vgg16_base_res_20201125_2053.h5')

In [16]: predictions = distress_model.predict(images.reshape(-1, 224, 224, 3))
         pred_images = predictions.reshape(images.shape[0], -1)

In [6]: sil = []
        kl = []
        kmax = 10
        for k in range(2, kmax+1):
            kmeans2 = KMeans(n_clusters = k).fit(pred_images)
            labels = kmeans2.labels_
            sil.append(silhouette_score(pred_images, labels, metric =   "euclidean"))
            kl.append(k)

In [10]: plt.plot(kl, sil)
         plt.ylabel("Silhouette Score")
         plt.xlabel("K")
         plt.show()
```

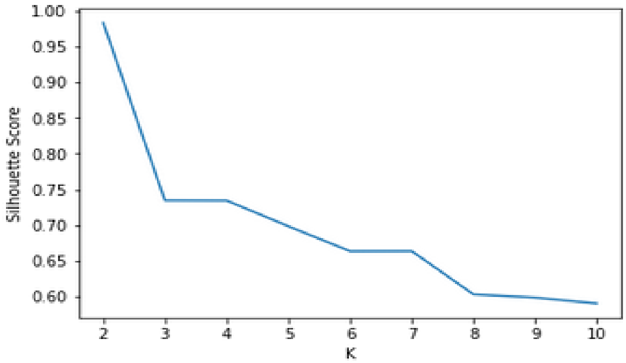


Figure 69: Silhouette score - Euclidean - Modified VGG-16

```
In [16]: sil = []
         kl = []
         kmax = 10
         for k in range(2, kmax+1):
             kmeans2 = KMeans(n_clusters = k).fit(pred_images)
             labels = kmeans2.labels_
             sil.append(silhouette_score(pred_images, labels, metric =   "cosine"))
             kl.append(k)
         plt.plot(kl, sil)
         plt.ylabel("Silhouette Score")
         plt.xlabel("K")
         plt.show()
```

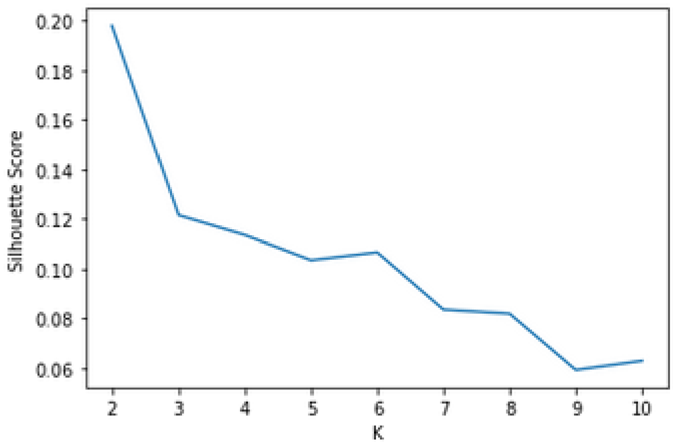


Figure 70: Silhouette score - Cosine - Modified VGG-16

7. After verifying the silhouette scores, it is concluded that the best clustering ability is shown by the target trained model of VGG-16. Next few steps will cluster the images using the best extractor. Figure 71 displays the clustering code in *Python* using *KMeans* function of *scikit learn* library.

```
In [11]: k = 2
         kmodel = KMeans(n_clusters = k, n_jobs=-1, random_state=728)
         kmodel.fit(pred_images)
         kpredictions = kmodel.predict(pred_images)
         os.makedirs(CLUSTER_PATH + "\output")

         shutil.rmtree(CLUSTER_PATH + "\output")
         for i in range(k):
             os.makedirs(CLUSTER_PATH + "\output\cluster" + str(i))
         for i in range(len(paths)):
             shutil.copy2(paths[i], CLUSTER_PATH + "\output\cluster"+str(kpredictions[i]))
```

Figure 71: K-Means Clustering

8. Figure 72 displays the cluster folders created.

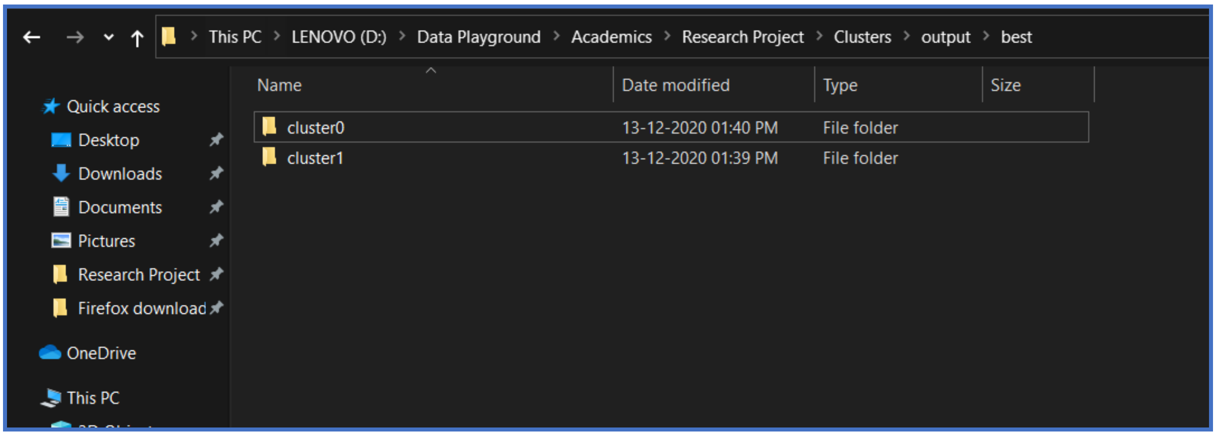


Figure 72: Folders of 2 clusters.

9. The clusters created are displayed in figures 73 and 74.

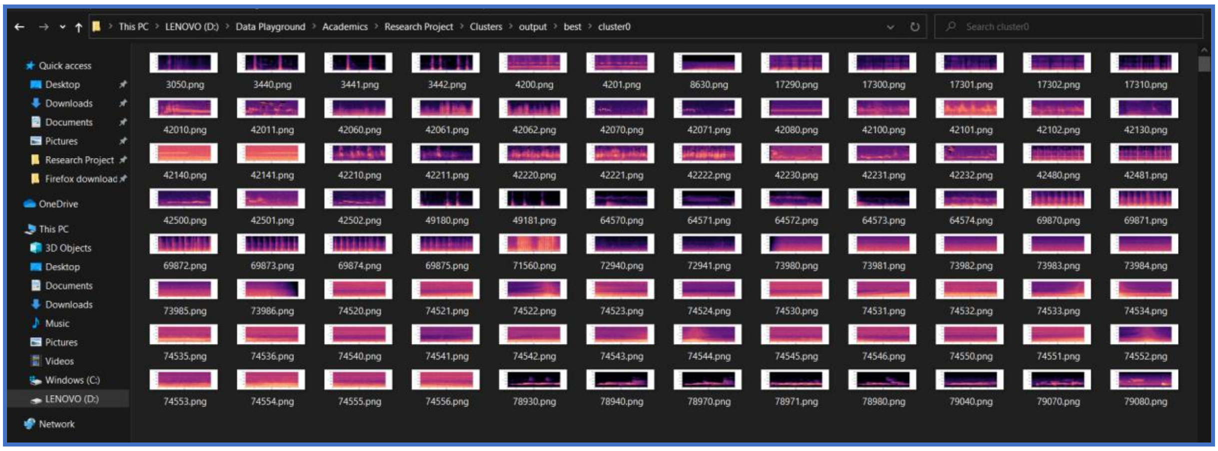


Figure 73: Cluster 0

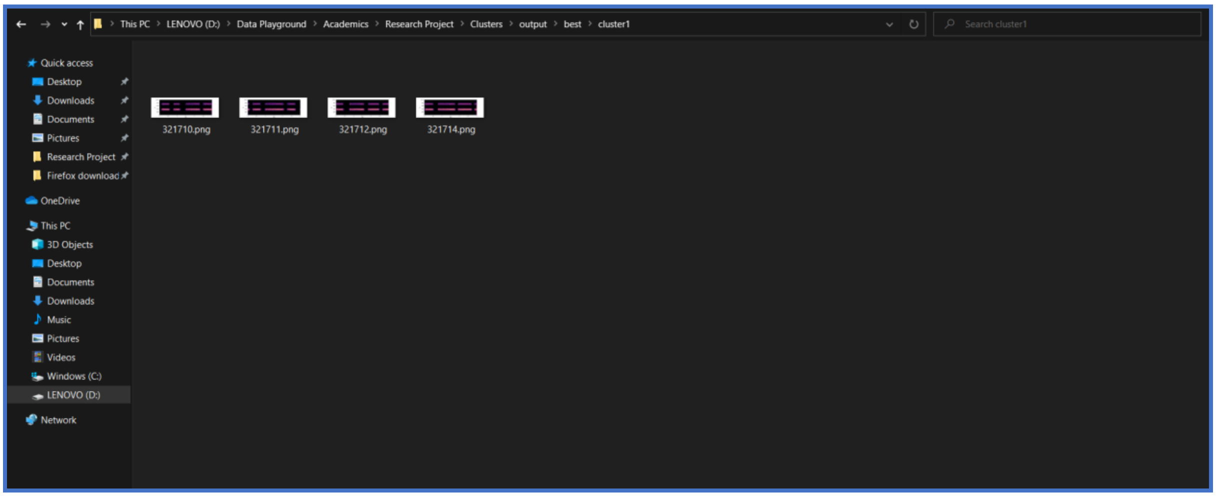


Figure 74: Cluster 1

8 Extra Work Done.

Mel-frequency cepstral coefficients (MFCCs) are coefficients which can be utilised for deep learning on audio. An attempt was made to extract these features. This method was discarded for much more favourable feature representation of log-mel-spectrograms.

```
In [1]: import os
        import numpy as np
        import librosa
        import librosa.display
        import tensorflow as tf
        from tensorflow import keras
        import matplotlib.pyplot as plt
        import IPython.display as ipd
        import pandas as pd
        import math
        import json

In [ ]: audio_data = 'D:/Data Playground/Academics/Research Project/FSD50K/FSD50K.dev_audio/136.wav'
        x , sr = librosa.load(audio_data, sr=44100)
        print(type(x), type(sr))
        print(x.shape, sr)
        ipd.Audio(audio_data)

In [2]: ABS_PATH = os.path.abspath(os.path.join('D:', '/Data Playground/Academics/Research Project'))
        DATA_PATH = ABS_PATH + '/SplitAudio/train'
        JSON_PATH_TRAIN =  ABS_PATH + '/train_mfcc.json'
        SAMPLE_RATE = 44100
```

Figure 75: Importing libraries and defining constants.

```
In [3]: def save_mfcc(dataset_path, json_path, n_mfcc=13, n_fft=2048, hop_length=512, num_segments=5):

            #dictionary to store data
            data = {
                "mapping": [],
                "mfcc": [],
                "labels": []
            }


            SAMPLES_PER_TRACK = SAMPLE_RATE*3
            num_samples_per_segment = int(SAMPLES_PER_TRACK/num_segments)
            expected_num_mfcc_vectors_per_segment = math.ceil(num_samples_per_segment / hop_length)
            #loop through the audio in folder
            for i, (dirpath, dirnames,  filenames) in enumerate(os.walk(dataset_path)):
                if dirpath is not dataset_path:
                    #save the semantic label
                    dirpath_components = dirpath.split("\\")
                    semantic_label = dirpath_components[-1]
                    data["mapping"].append(semantic_label)
                    print('\nProcessing {}'.format(semantic_label))
                    for f in filenames:

                        #load audio file
                        file_path = os.path.join(dirpath, f)
                        signal, sr = librosa.load(file_path, sr=SAMPLE_RATE)

                        #process segments extracting mfcc and storing data
                        for s in range(num_segments):
                            start_sample = num_samples_per_segment * s
                            finish_sample = start_sample + num_samples_per_segment


                            mfcc = librosa.feature.mfcc(signal[start_sample:finish_sample],
                                                        sr=SAMPLE_RATE,
                                                        n_fft=n_fft,
                                                        n_mfcc=n_mfcc,
                                                        hop_length=hop_length)
                            mfcc = mfcc.T
                            if len(mfcc) == expected_num_mfcc_vectors_per_segment:
                                data["mfcc"].append(mfcc.tolist())
                                data["labels"].append(i-1)
                                print("{}, segment:{}".format(file_path, s+1))
            with open(JSON_PATH_TRAIN, "w") as fp:
                json.dump(data,fp, indent=4)
```

Figure 76: Function to extract and store MFCCs

```
In [4]: data = save_mfcc(dataset_path=DATA_PATH, json_path=JSON_PATH_TRAIN, num_segments=1)

        Processing Danger
        D:\Data Playground\Academics\Research Project/SplitAudio/train\Danger\1001430.wav, segment:1
        D:\Data Playground\Academics\Research Project/SplitAudio/train\Danger\1001431.wav, segment:1
        D:\Data Playground\Academics\Research Project/SplitAudio/train\Danger\1001440.wav, segment:1
        D:\Data Playground\Academics\Research Project/SplitAudio/train\Danger\1001441.wav, segment:1
        D:\Data Playground\Academics\Research Project/SplitAudio/train\Danger\1003570.wav, segment:1
        D:\Data Playground\Academics\Research Project/SplitAudio/train\Danger\1003571.wav, segment:1
        D:\Data Playground\Academics\Research Project/SplitAudio/train\Danger\1003572.wav, segment:1
        D:\Data Playground\Academics\Research Project/SplitAudio/train\Danger\1003573.wav, segment:1
        D:\Data Playground\Academics\Research Project/SplitAudio/train\Danger\1004590.wav, segment:1
        D:\Data Playground\Academics\Research Project/SplitAudio/train\Danger\1004591.wav, segment:1
        D:\Data Playground\Academics\Research Project/SplitAudio/train\Danger\1004592.wav, segment:1
        D:\Data Playground\Academics\Research Project/SplitAudio/train\Danger\1004593.wav, segment:1
        D:\Data Playground\Academics\Research Project/SplitAudio/train\Danger\1004594.wav, segment:1
        D:\Data Playground\Academics\Research Project/SplitAudio/train\Danger\1004595.wav, segment:1
        D:\Data Playground\Academics\Research Project/SplitAudio/train\Danger\1004596.wav, segment:1
        D:\Data Playground\Academics\Research Project/SplitAudio/train\Danger\1004597.wav, segment:1
        D:\Data Playground\Academics\Research Project/SplitAudio/train\Danger\1004598.wav, segment:1

In [ ]: # data
        with open(JSON_PATH_TRAIN, "w") as fp:
            json.dump(data,fp, indent=4)
```

Figure 77: Creating the JSON file to store MFCCs.

9 Conclusion

All the steps required to replicate the thesis is explained in this manual. Thank you for showing interest in my work. If any queries are unanswered, please feel free to contact me @ magmidhh@gmail.com.

References

Eduardo Fonseca, Xavier Favory, Jordi Pons, Frederic Font, Xavier Serra. "FSD50K: an Open Dataset of Human-Labeled Sound Events", arXiv:2010.00475, 2020.