Scientific Programming in Python (PHYS4038)

Instrument Detection

Divyansh Oze

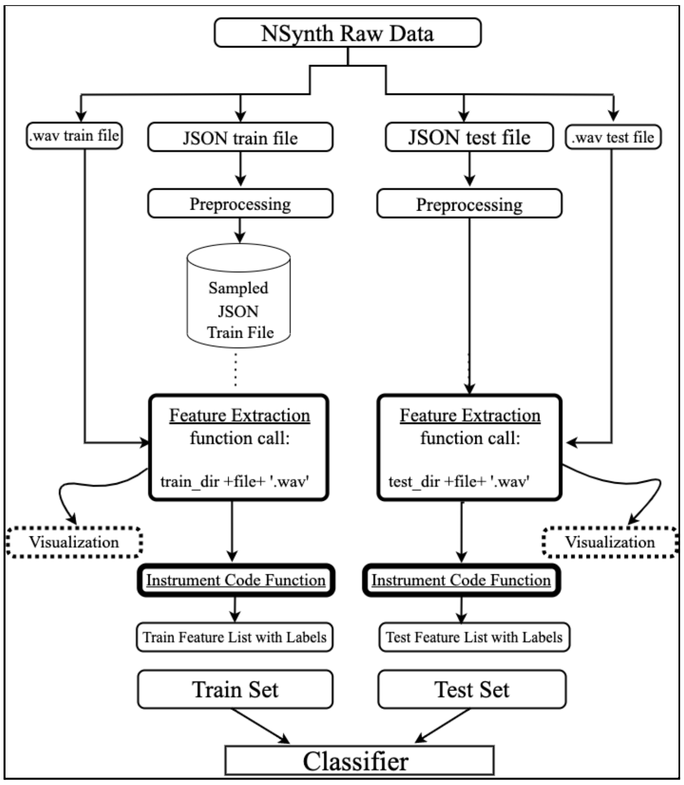
Abstract:

To predict what instrument is playing in the background of an Audio File, we start from the basic anatomy of an audio signal. We break it down into components the system can understand, then transform it into different domains for more perspective. We take the efficient features and work our predictions off them for the instrument detection.

Our optimal goal is to understand if we can detect the instrument playing in the background of an AudioClip.

1.) Introduction:

We used the NSynth Dataset to implement this task which consists of many existing features. There was unequal distribution between the files for the instrument family, so we sampled it to 400 files for each instrument family for 4000 audio files under Preprocessing section. The idea of extracting unique visual features from using the audio .wav files present in the dataset fascinated me to go beyond and use these self-extracted features to experiment visualisation tasks on them and get then further on using them to detect instruments playing in the background. A general flowchart for the code design is shown below in Figure[1].



Figure[1]: A flowchart of the code design for this project

2.) Methodology:

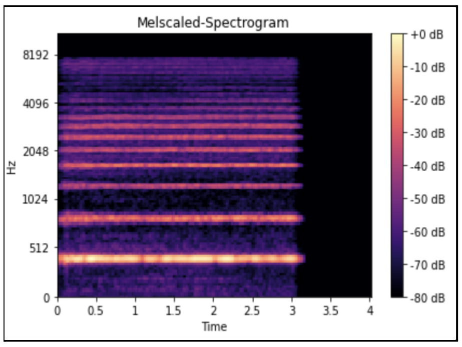
2.1) Feature Extraction:

a) **MelSpectrogram**: It helps us represent the strength of a signal; simply the “loudness”, as it varies over time at different frequencies. In addition to a spectrogram, Mel scales the frequency to match what the human ear can hear more closely, as shown in Figure[2].

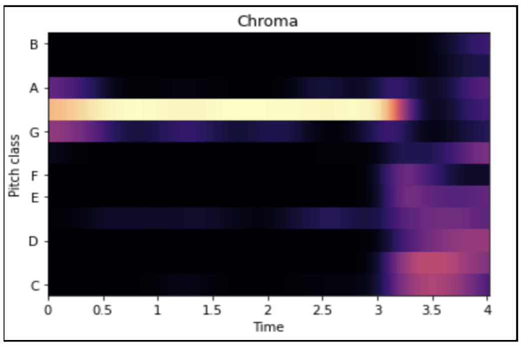
b) **MFCC:** Applying Inverse DCT (inverse discrete cosine transform) on Mel spectrograms gets us MFCC, an important cepstrum feature for our model and can also help in genre classification for dream work. Graphical Representation is shown in Figure[3].

c) **Chroma**: Chromagram helps us understand similarities between the harmonic or melodic characteristics within the signal, using the 12 pitch classes. As shown in Figure[4].

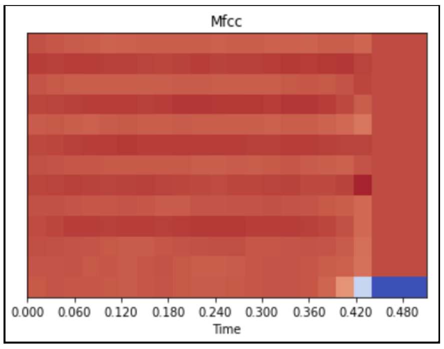
d) **Contrast**: Spectral contrast to observe peaks, valleys and to analyse their difference between each frequency sub-band. As shown in Figure[5].



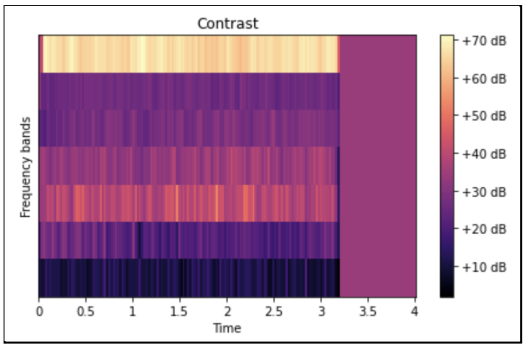
Figure[2]: Visual representation of Mel-Spectrogram



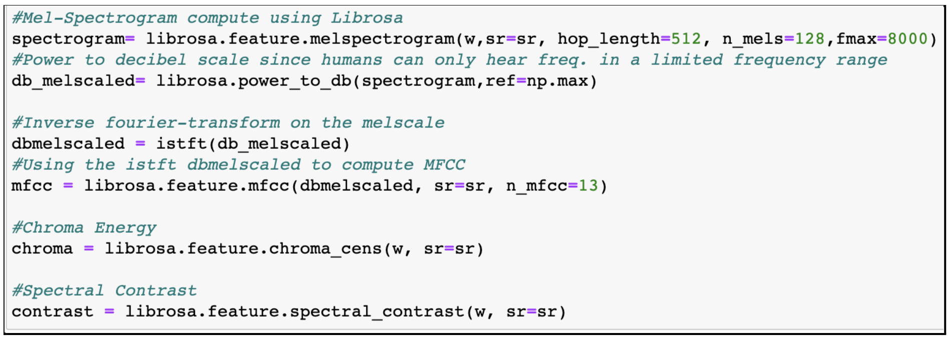
Figure[4]: Visual representation of Chroma Energy



Figure[4]: Visual representation of MFCC



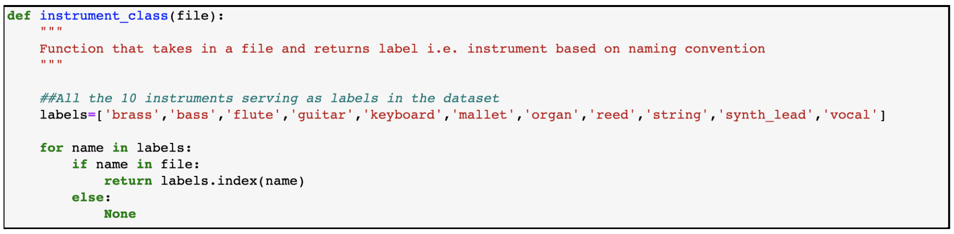
Figure[5]: Visual representation of Spectral Contrast



```
#Mel-Spectrogram compute using Librosa
spectrogram= librosa.feature.melspectrogram(w,sr=sr, hop_length=512, n_mels=128,fmax=8000)
#Power to decibel scale since humans can only hear freq. in a limited frequency range
db_melscaled= librosa.power_to_db(spectrogram,ref=np.max)

#Inverse fourier-transform on the melscale
dbmelscaled = istft(db_melscaled)
#Using the istft dbmelscaled to compute MFCC
mfcc = librosa.feature.mfcc(dbmelscaled, sr=sr, n_mfcc=13)

#Chroma Energy
chroma = librosa.feature.chroma_cens(w, sr=sr)

#Spectral Contrast
contrast = librosa.feature.spectral_contrast(w, sr=sr)
```

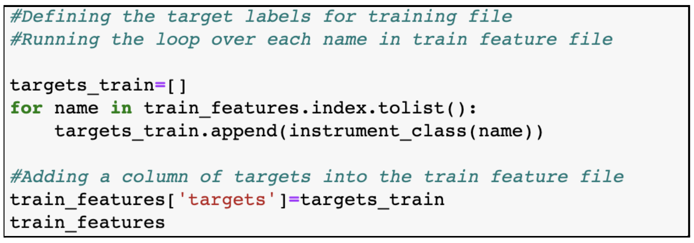
Figure.[6]: A code snippet showing how we extracted our features

<u>2.2) Instrument Code</u>: It gives out the instrument family code for the passed file after matching the contents in the file with defined labels. Used to store labels in train and test set as targets. The function, i.e. Figure.[7] with how it is working on keeping targets in the feature list for the train/test set, i.e. Figure.[8].



```python
def instrument_class(file):
    """
    Function that takes in a file and returns label i.e. instrument based on naming convention
    """

    ##All the 10 instruments serving as labels in the dataset
    labels=['brass','bass','flute','guitar','keyboard','mallet','organ','reed','string','synth_lead','vocal']

    for name in labels:
        if name in file:
            return labels.index(name)
        else:
            None
```

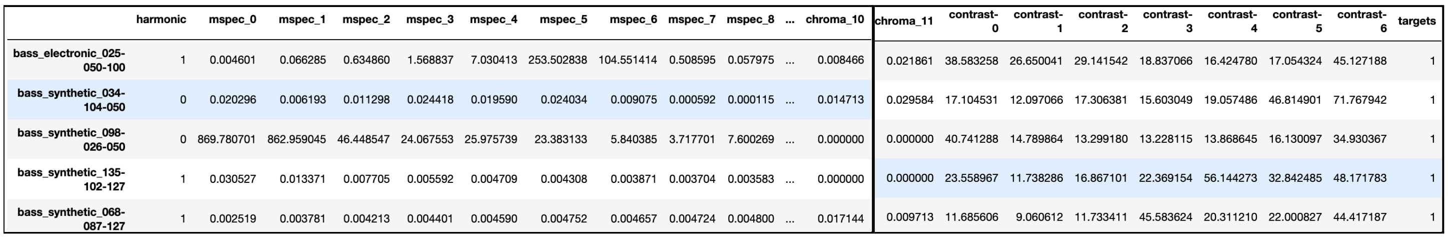
Figure[7]: A snippet showing the working for Instrument_Class Function



```python
#Defining the target labels for training file
#Running the loop over each name in train feature file

targets_train=[]
for name in train_features.index.tolist():
    targets_train.append(instrument_class(name))

#Adding a column of targets into the train feature file
train_features['targets']=targets_train
train_features
```

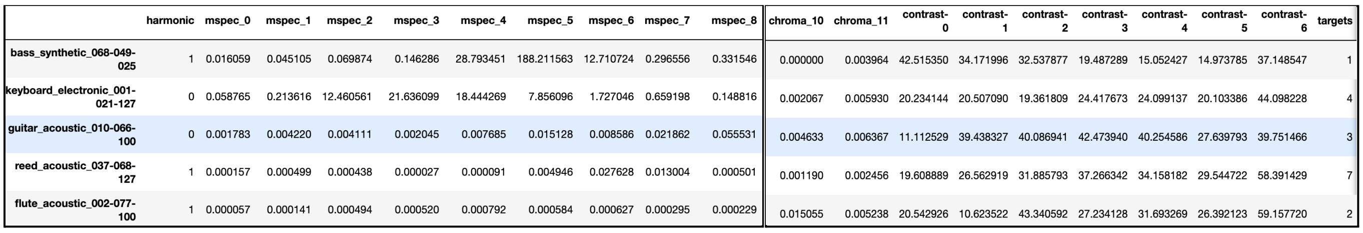
Figure[8]: A part of code where we put targets in the feature list

<u>2.3) Train and Test set</u>: After putting targets in our feature list, we are ready with these sets to pass through the Classifier for the Classification Task. We changed the MxN array values of all self-generated features to have their separate indexed columns for them to be in a tabular format and not cause us problems in the prediction task.



	harmonic	mspec_0	mspec_1	mspec_2	mspec_3	mspec_4	mspec_5	mspec_6	mspec_7	mspec_8	...	chroma_10	chroma_11	contrast-0	contrast-1	contrast-2	contrast-3	contrast-4	contrast-5	contrast-6	targets
bass_electronic_025-050-100	1	0.004601	0.066285	0.634860	1.568837	7.030413	253.502838	104.551414	0.508595	0.057975	...	0.008466	0.021861	38.583258	26.650041	29.141542	18.837066	16.424780	17.054324	45.127188	1
bass_synthetic_034-104-050	0	0.020296	0.006193	0.011298	0.024418	0.019590	0.024034	0.009075	0.000592	0.000115	...	0.014713	0.029584	17.104531	12.097066	17.306381	15.603049	19.057486	46.814901	71.767942	1
bass_synthetic_098-026-050	0	869.780701	862.959045	46.448547	24.067553	25.975739	23.383133	5.840385	3.717701	7.600269	...	0.000000	0.000000	40.741288	14.789864	13.299180	13.228115	13.868645	16.130097	34.930367	1
bass_synthetic_135-102-127	1	0.030527	0.013371	0.007705	0.005592	0.004709	0.004308	0.003871	0.003704	0.003583	...	0.000000	0.000000	23.558967	11.738286	16.867101	22.369154	56.144273	32.842485	48.171783	1
bass_synthetic_068-087-127	1	0.002519	0.003781	0.004213	0.004401	0.004590	0.004752	0.004657	0.004724	0.004800	...	0.017144	0.009713	11.685606	9.060612	11.733411	45.583624	20.311210	22.000827	44.417187	1

Figure[9]: Snippet showing the sampled Train DataFrame with features and respective targets



	harmonic	mspec_0	mspec_1	mspec_2	mspec_3	mspec_4	mspec_5	mspec_6	mspec_7	mspec_8	chroma_10	chroma_11	contrast-0	contrast-1	contrast-2	contrast-3	contrast-4	contrast-5	contrast-6	targets
bass_synthetic_068-049-025	1	0.016059	0.045105	0.069874	0.146286	28.793451	188.211563	12.710724	0.296556	0.331546	0.000000	0.003964	42.515350	34.171996	32.537877	19.487289	15.052427	14.973785	37.148547	1
keyboard_electronic_001-021-127	0	0.058765	0.213616	12.460561	21.636099	18.444269	7.856096	1.727046	0.659198	0.148816	0.002067	0.005930	20.234144	20.507090	19.361809	24.417673	24.099137	20.103386	44.098228	4
guitar_acoustic_010-066-100	0	0.001783	0.004220	0.004111	0.002045	0.007685	0.015128	0.008586	0.021862	0.055531	0.004633	0.006367	11.112529	39.438327	40.086941	42.473940	40.254586	27.639793	39.751466	3
reed_acoustic_037-068-127	1	0.000157	0.000499	0.000438	0.000027	0.000091	0.004946	0.027628	0.013004	0.000501	0.001190	0.002456	19.608889	26.562919	31.885793	37.266342	34.158182	29.544722	58.391429	7
flute_acoustic_002-077-100	1	0.000057	0.000141	0.000494	0.000520	0.000792	0.000584	0.000627	0.000295	0.000229	0.015055	0.005238	20.542926	10.623522	43.340592	27.234128	31.693269	26.392123	59.157720	2

Figure[10]: Snippet showing the Test DataFrame with features and respective targets

2.4) Random Forest Classifier:

We used Random Forest Classifier for our Classification Task. It consists of multiple uncorrelated decision trees to emit out a Class Prediction; then, we pick the class seen the maximum number of times in the system. More uncorrelation between the decision trees, the higher the accuracy for our predictions.

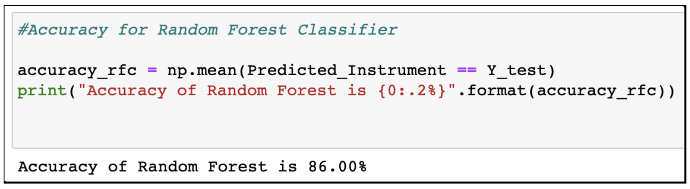
2.4.1) Why we used Random Forest specifically for our model:

A. It moves down in-depth to compare all of the features in terms of their efficiency for the prediction. It helps declutter the feature overlap within our self-generated components extracted from the audio .wav file.

B. It is computationally cheaper and more time effective than any deep learning framework.

2.4.2) Model Accuracy and Precision:

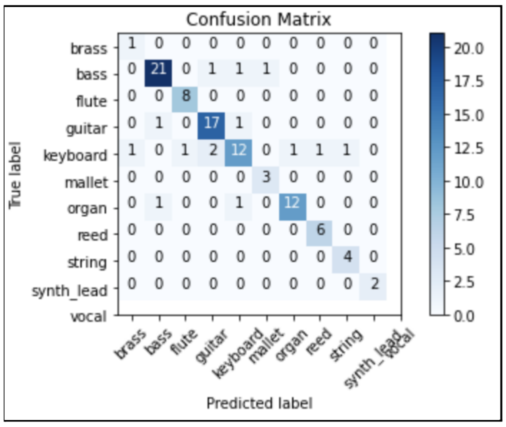
We trained the model using X and Y train sets, then tested it on X test to get predictions for instruments, i.e. Predicted_Instrument. After comparing the predictions to labels in the Y test, We get accuracy for the Classifier, as shown below in Figure[11], and a confusion matrix to evaluate performance shown in Figure[12].



```
#Accuracy for Random Forest Classifier

accuracy_rfc = np.mean(Predicted_Instrument == Y_test)
print("Accuracy of Random Forest is {0:.2%}".format(accuracy_rfc))
```

```
Accuracy of Random Forest is 86.00%
```

Figure[11]: A code snippet calculating the Accuracy for Random Forest Classifier



	brass	bass	flute	guitar	keyboard	mallet	organ	reed	string	synth_lead	vocal
brass	1	0	0	0	0	0	0	0	0	0	
bass	0	21	0	1	1	1	0	0	0	0	
flute	0	0	8	0	0	0	0	0	0	0	
guitar	0	1	0	17	1	0	0	0	0	0	
keyboard	1	0	1	2	12	0	1	1	1	0	
mallet	0	0	0	0	0	3	0	0	0	0	
organ	0	1	0	0	1	0	12	0	0	0	
reed	0	0	0	0	0	0	0	6	0	0	
string	0	0	0	0	0	0	0	0	4	0	
synth_lead	0	0	0	0	0	0	0	0	0	2	
vocal											

Figure[12]: Confusion Matrix describing the performance for each instrument

We got good enough accuracy on the classification task for our model since we have trained it on all of the data and music files from the NSynth Dataset, we could try training our model on the data it has not seen to know if The features we have extracted are efficient in getting us the instrument or failure to do so.

3.) Learning Curve:

There has been immense learning from working on the .wav audio components taking mathematical expressions like Fourier-Transforms using **Librosa**. To get to visualise all sorts of self-generated features using **Matplotlib**, refining hands-on **NumPy** and **pandas** for handling data throughout the project. A little bit of File Handling I/O operations to store and dump progress in the pickle files. Python has all sorts of libraries to make the work successful in no time if put in the right effort. We studied the anatomy of a .wav file to extract some features off from the NSynth audio files. Further, we used the features to detect the instrument playing in the background. It is fair to conclude that we can detect instruments playing in the background using python.

4.) Improvements:

- Add more features to understand peculiarities within audio wave file components.
- Try to run this over an entire music clip to detect the powerful instrument playing in the background.
- Live visualisation effects on the audio playing in sync.

5.) References:

- Nsynth Dataset, https://magenta.tensorflow.org/datasets/nsynth
- Librosa Library, https://librosa.org/doc/latest/index.html
- Random Forests, Leo Breiman and Adele Cutler
- Music and Instrument Classification using Deep Learning Technics, Lara Haidar-Ahmad, CS230.
- Speech Recognition Using Articulatory and Excitation Source Features, SpringerBriefs in Speech Technology, [2017]. K.S. Rao and Manjunath K.E.