1 Definition

(approx. 1-2 pages)

1.1 Project Overview

Music is one of the oldest ways of expression known to man. We create musical compositions by arranging notes from various tools such as our own voices, physical instruments, and synthetic means. We can identify instruments by just listening to it, often without having heard it often. Most of the general population can tell notes apart and those with music theory background can name the notes.

What is it that makes an instrument sound like it does? Why does a guitar sound differently than bass? Does a synthetic bass and a physical one sound the same? What are the features that define a sound? Is there a correlation between these features? Given sound files in wave format, how to do we answer the previous questions using a personal computer and data science techniques?

This report will delve into music classification from a computational angle, present the objectivesand explain the various techniques available to solve them. To do this we use NSynth [1], a large scale and high-quality dataset of annotated music notes and free to download for academic purposes (https://magenta.tensorflow.org/datasets/nsynth).

1.2 Problem Statement

The objective of this project is to classify the instrument used to generate a note. The note has a single instrument source and is stored as a four second wave file, Table 1 below displays the list of instruments used to generate notes, there are a total of 11 distinct labels.
Table 2 shows the frequency distribution of these instruments and further breaks them down by source. The index corresponds to the instrument label and our goal is to correctly predict these labels.

Table 1 Instrument Families

Index	ID
0	bass
1	brass
2	flute
3	guitar
4	keyboard
5	mallet
6	organ
7	reed
8	string
9	synth_lead
10	vocal

Table 2 Frequency Counts for Instrument Classes

Family	Acoustic	Electronic	Synthetic	Total
Bass	200	8,387	60,368	68,955
Brass	13,760	70	0	13,830
Flute	6,572	35	2,816	9,423
Guitar	13,343	16,805	5,275	35,423
Keyboard	8,508	42,645	3,838	54,991
Mallet	27,722	5,581	1,763	35,066
Organ	176	36,401	0	36,577
Reed	14,262	76	528	14,866
String	20,510	84	0	20,594
Synth Lead	0	0	5,501	5,501
Vocal	3,925	140	6,688	10,753
Total	108,978	110,224	86,777	305,979

The dataset provided as training, validation and testing files, each file contains two components:

- 16-bit PCM Wave files
- JSON files containing non-audio features

First we will conduct an exploration of the non-audio features to get a sense of their distribution and ascertain if they can be useful for classification. We will then extract features from the wave files using packages such as Librosa and repeat the exercise, this time extracting audio features and molding them into digestible fragments. The objective of this part is to provide a concise definition of each feature, what it means in the domain of music theory and for classification.

Given extracted and polished features, we will attempt to predict the instrument labels by means of supervised learning algorithms. Some algorithms such as support vector machines might require additional feature processing. All features will be explained and not all of them will make the cut into the feature space.

This exercise will be useful in determining which features are more useful than others and serve as a sensitivity study. If we can obtain solid results with less features, we avoid unnecessary computation and can perhaps scale more easily to larger datasets or additional classes.

Finally, we leverage a simple convolutional neural network for music classification. To do so we develop a way to represent each wave file as an image. The shape of the image representation can also serve as a useful sensitivity study. Classifying several images using CNNs has yielded impressive results, most notably in the CIFAR-10 dataset by Krizhevsky [2].

1.3 Metrics

Our goal is to correctly label several instruments, we are therefore performing multiclass classification. The metrics defined below are explained in more details and with python code examples by Müller [3]. The first and governing measure of performance is accuracy, defined as:

$$Accuracy = \frac{Number\ of\ True\ Predictions}{Total\ Number\ of\ Predictions}$$

However what if we take for example an imaginary dataset for classification with labels distributed as 90-5-5, what does 90% accuracy represent? This poses a clear problem

Given the nature of the problem and the imbalanced distribution of labels, we introduce the confusion matrix as a better method to quantify performance.

The output of the confusion matrix is an *nxn* matrix where n is the number of classes, the rows correspond to the true classes and the columns correspond to the predicted class.

Table 3 Confusion Matrix for Binary Classification

Negative Class	TN	FP
Positive Class	FN	TP
	Predicted Negative	Predicted Positive

Table 3 below shows a confusion matrix for binary classification, to maximize our accuracy score we must increase the values on the diagonal as they correspond to correct predictions. We use the table above to introduce precision and recall, defined as:

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

We use the above two equations to f-score (also referred to as f1 score) corresponding to the harmonic mean of precision and recall:

$$f_{score} = \frac{precision \times recall}{precision + recall}$$

To compute the multiclass f-score, we just compute the binary f-score per class, such that the class in question corresponds to the positive class and all other classes are the negative class. For example, the precision of 'bass' class is:

$$precission_{bass} = \frac{Correct\ Bass\ Predictions}{Correct\ Bass\ Predictions + \ Not\ Bass\ but\ Predicted\ Bass}$$

We then average these f-scores using one of three methods:

- Macro averaging: computes the unweighted per class scores, gives equal weight to all classes, regardless of size. If we care about each class equally as much, macro averaging is the preferred method.
- Weighted averaging: computes mean of the per-class f-scores, weighted by their support.
- Micro averaging: computes the total number of false positives, false negatives, and true positives over all classes, then computes precision, recall, and f-score using these counts. If we care about each sample equally as much, micro averaging is he preferred method.

For neural networks we will stick to accuracy as the only performance metric. However we will consider the difference in training loss between training and validation datasets as means of exploring how well our model trains to the data.

2 Analysis

2.1 Data Exploration

The dataset is publicly available for download. As shown in previous sections, all the statistics regarding instrument counts and distributions for certain characteristics are already provided. The documentation states that no datapoints are missing. In addition, the input space provides both categorical variables and their equivalent numeric representation.

2.2 Exploratory Visualization

We begin exploration by picking ten random wave files, one for each class. The objective is to explain each audio feature in and provide visualization using Librosa's feature extraction method. Before diving deeper we introduce the concept of frequency: the number of occurrences of a repeating event per unit of time. Frequency is measured in Hertz or in s^{-1} (inverse seconds).

The exploration of wave files can be found here:

https://github.com/NadimKawwa/NSynth/blob/master/NSynth_WaveFile_Exploration.ipynb

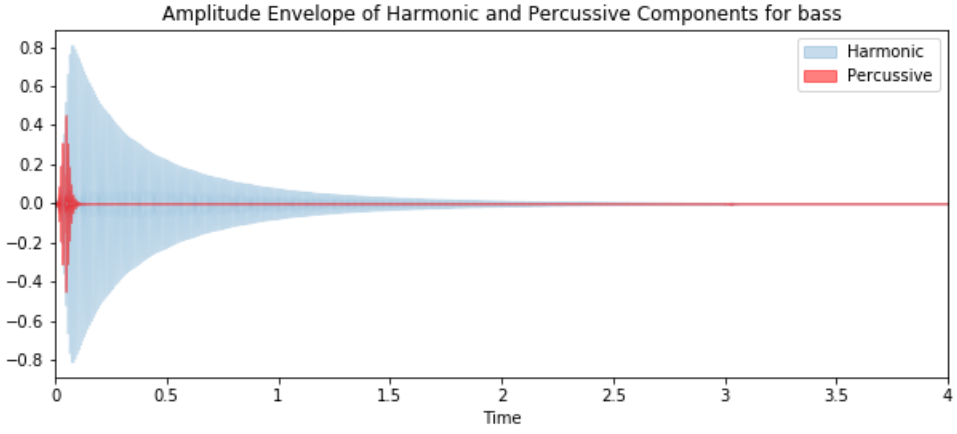
2.2.1 Waveform and Sampling Rate

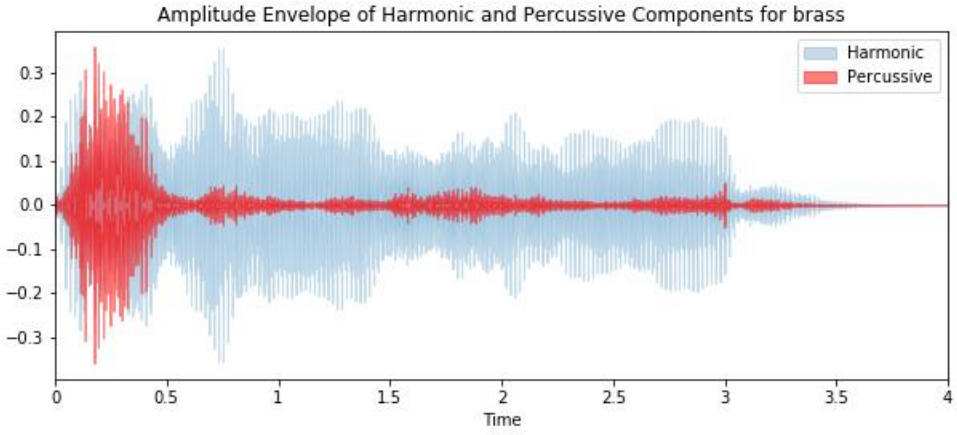
A sound is a continuous time signal that is usually represented by a **waveform**; this continuous signal is **sampled** and converted into a discrete time signal. **Sampling Rate** is the number of samples of audio recorded per second. The sampling rate determines the maximum audio frequency that can be reproduced, Indeed the maximum frequency that can be represented is half the sample rate. Most humans can hear sounds in the frequency of 20-20,000Hz [4] and so most some sounds like CDs are sampled are 44,000Hz, that is to say in one second the analog signal is sampled 44,100. When loading the files with librosa.core.load we opt to keep the original sampling rate. A sound can be either percussive or harmonic. Since the duration *t* audio file is 4 seconds and the sampling rate *sr* is 16,000 for all, the length of the output amplitude *y* can be inferred from the following equality:

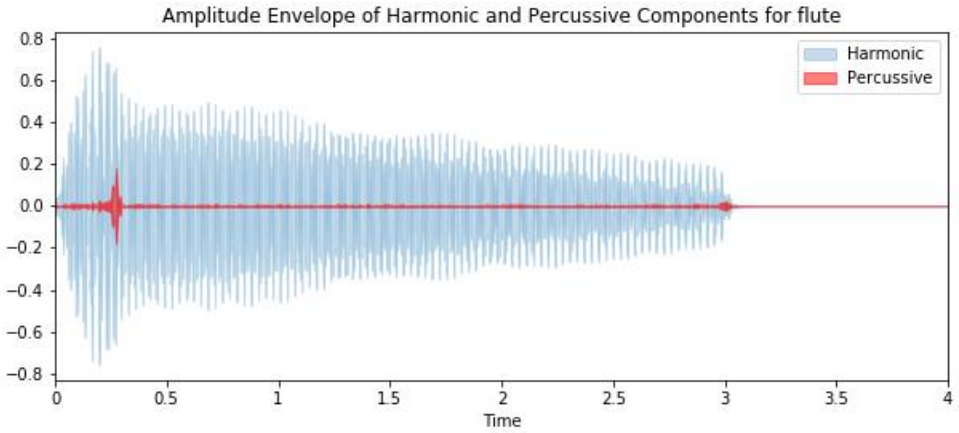
$$t\ (s) = \frac{length(y)}{sr\ (Hz)}$$

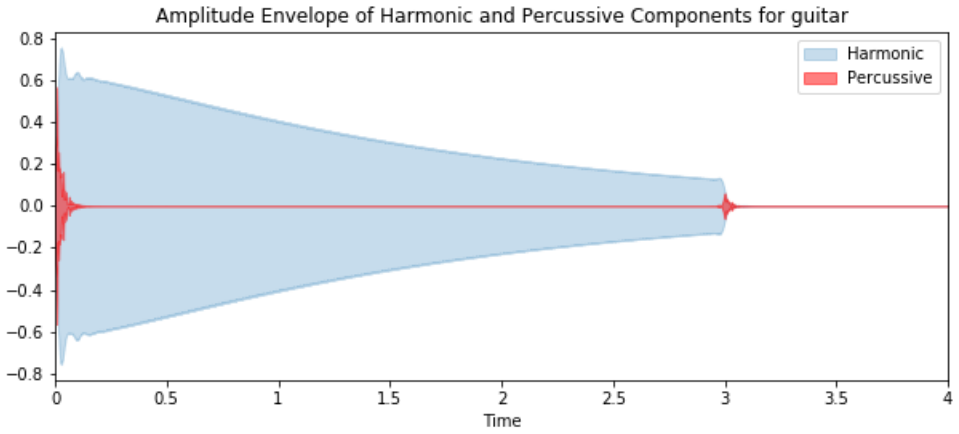
The above equation will later prove useful if we decide to manipulate sample size. The length of the file is a constant, therefore the length of the waveform is inversely proportional to the sampling rate.

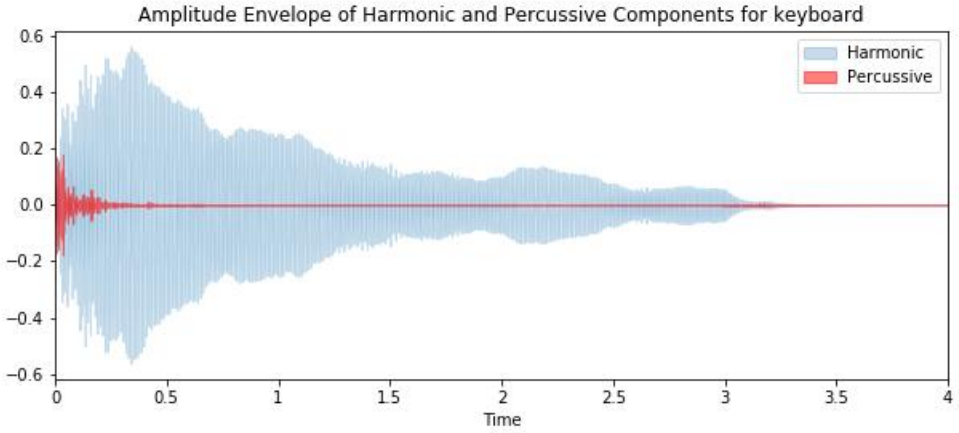
A harmonic sound is one where series of sounds within which the fundamental frequency of each of them is an integral multiple of the lowest fundamental frequency [5]. A percussion instrument is any object you can bang on [6]. Librosa contains a useful feature that allows us to decompose a sound into harmonic and percussive components based on Fitzgerald [7]. The plots below show the harmonic and percussive amplitudes of each instrument. It is interesting to notice that almost no instrument either purely harmonic or percussive. We can infer a lot of information from these plots by looking at the amplitudes and their decays. Notice how the flute is a series of high frequency decaying amplitudes while the mallet is one large “bang” at the beginning and immediately stops emitting sound.

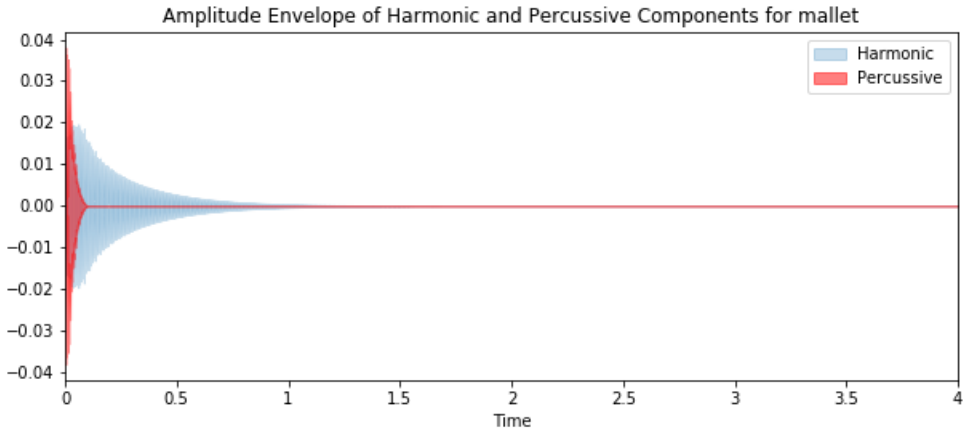


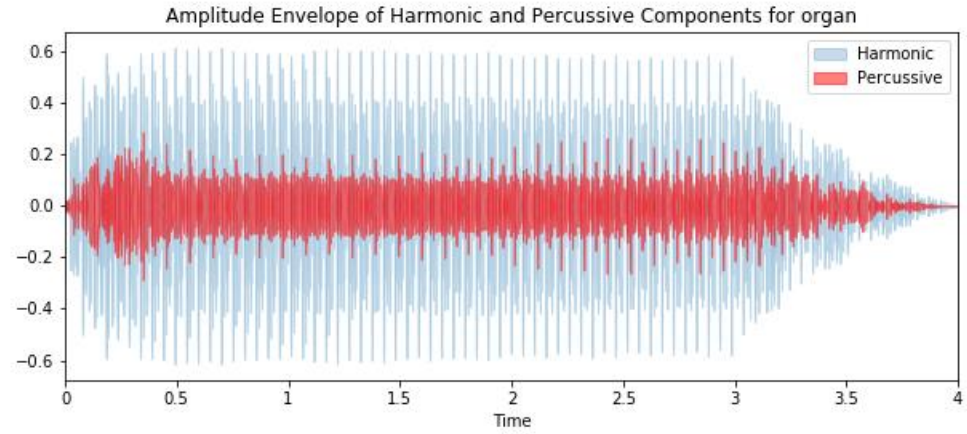


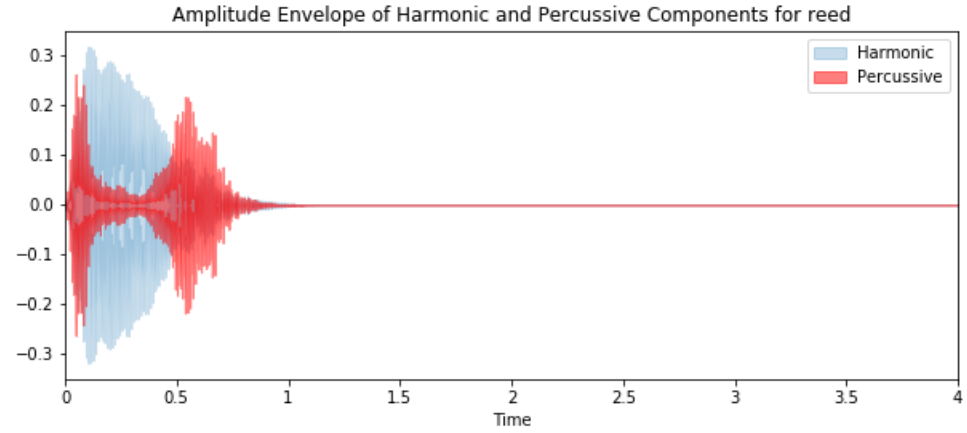

Amplitude Envelope of Harmonic and Percussive Components for flute
Harmonic
Percussive
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
0
0.5
1
1.5
2
2.5
3
3.5
4
Time

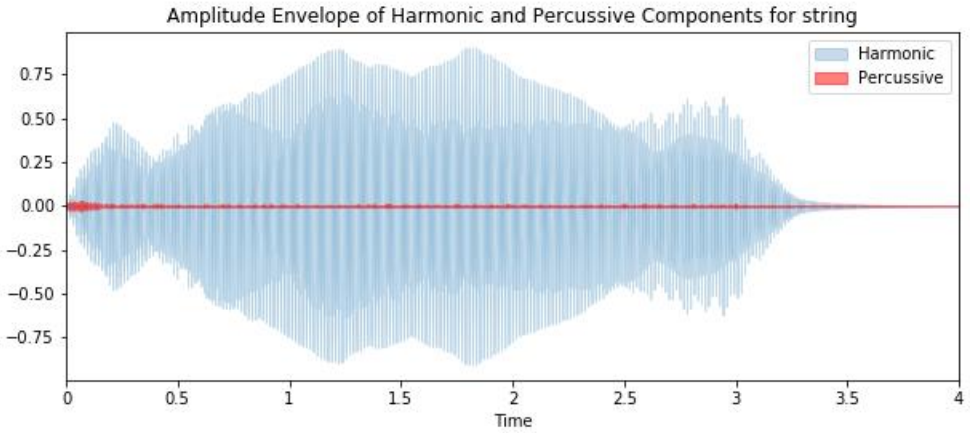

Amplitude Envelope of Harmonic and Percussive Components for guitar
Harmonic
Percussive
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
0
0.5
1
1.5
2
2.5
3
3.5
4
Time

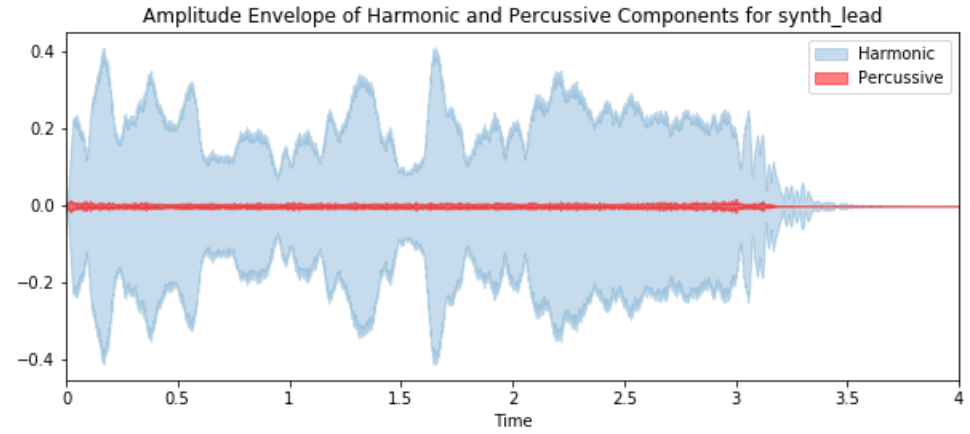

Amplitude Envelope of Harmonic and Percussive Components for keyboard
Harmonic
Percussive
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
0
0.5
1
1.5
2
2.5
3
3.5
4
Time

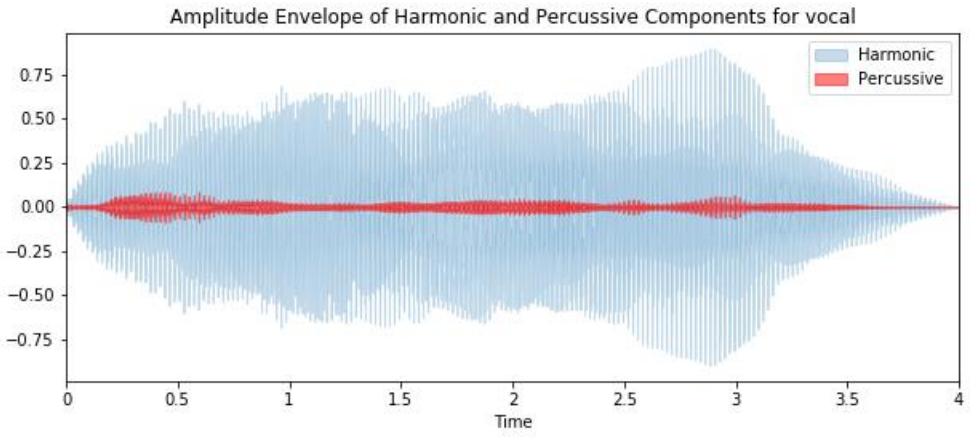

Amplitude Envelope of Harmonic and Percussive Components for mallet
Harmonic
Percussive
0.04
0.03
0.02
0.01
0.00
−0.01
−0.02
−0.03
−0.04
0
0.5
1
1.5
2
2.5
3
3.5
4
Time


Amplitude Envelope of Harmonic and Percussive Components for organ
Harmonic
Percussive
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
0
0.5
1
1.5
2
2.5
3
3.5
4
Time


Amplitude Envelope of Harmonic and Percussive Components for reed
Harmonic
Percussive
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
0
0.5
1
1.5
2
2.5
3
3.5
4
Time







2.2.2 Tempo

We refer to the Merriam-Webster entry for Tempo:

Tempo is the rate of speed of a music or passage indicated by one of a series of directions such as allegro and often by an exact metronome marking.

From the NSynth documentation we are informed that each instrument is played only once. This means that we should not be able to record any tempo using librosa.beat.beat_track, which is the case. This simplifies our task and probably limits our application to monotonic sounds.

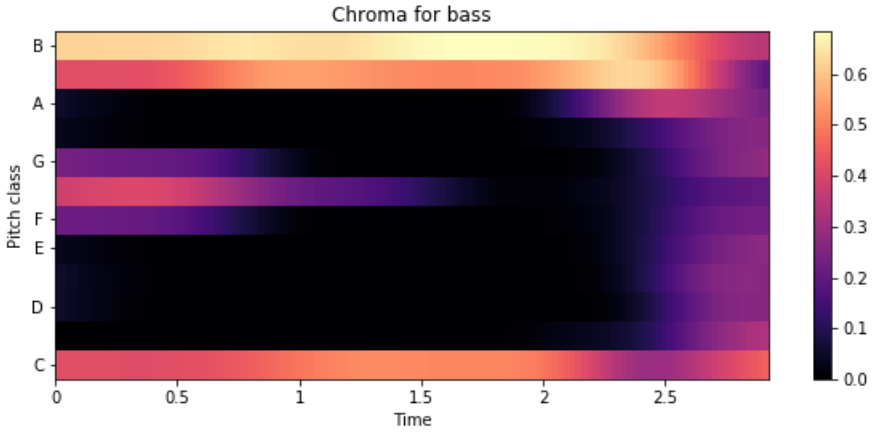
2.2.3 Chroma Energy

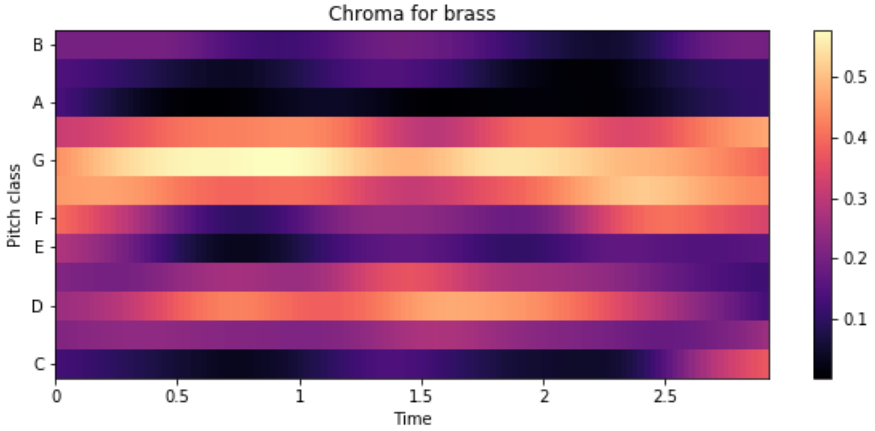
A pitch is the quality of a sound governed by the rates of vibrations producing it. Two pitches are perceived as similar in "color" if they differ by an octave. Based on this observation, a pitch can be separated into two components, which are referred to as tone height and chroma. Assuming an equal-tempered scale, the chromas correspond to the set {C, C♯, D, . . . , B} that consists of the twelve pitch spelling attributes as used in Western music notation. Thus, a chroma feature is represented by a 12 dimensional vector:

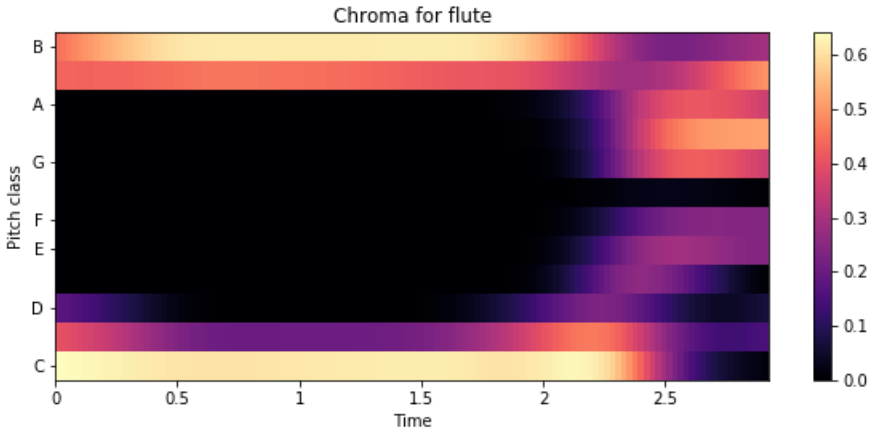
$$x = (x(1), x(2), \dots, (x12))^T$$

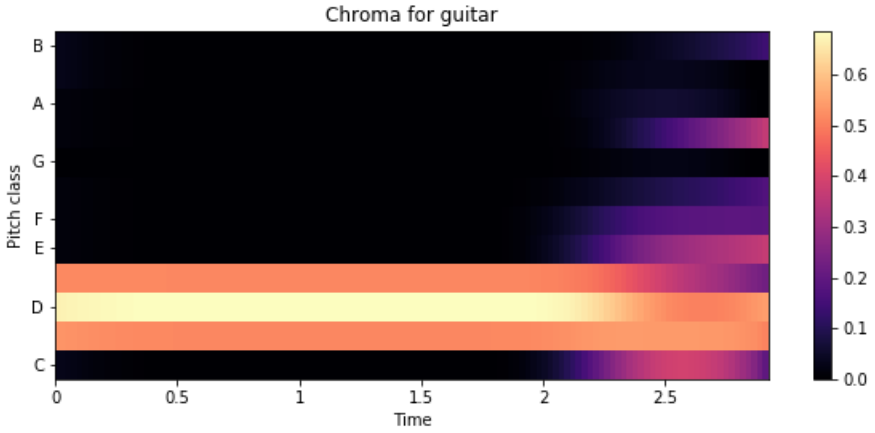
Where x(1) corresponds to chroma C, x(2) to chroma C# and so on. In the feature extraction step, a given audio signal is converted into a sequence of chroma features each expressing how the short-time energy of the signal is spread over the twelve chroma bands. The extraction of chromas is done using librosa.feature.chroma_cens based on Müller [8] and displays the variant chorma energy normalized (CENS).

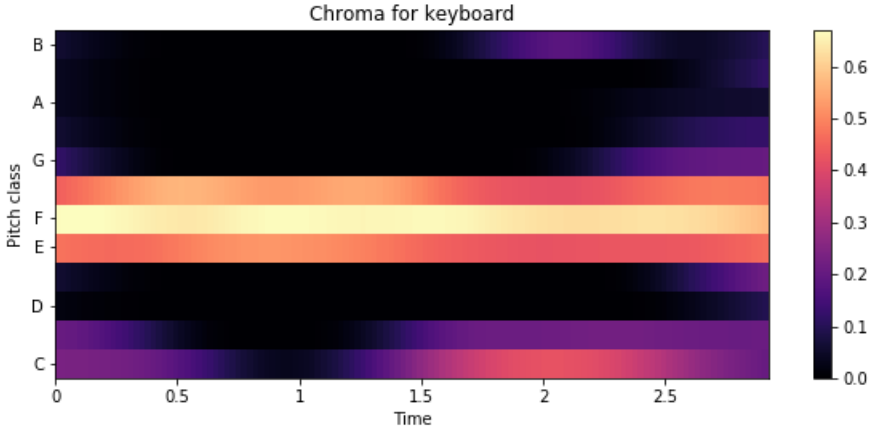
The plots below display the present chromas by color bands. Each color bands corresponds to the notes that the instrument hits. We also notice that some notes are more activated then others and some decay suddenly. For the purpose of our exercise, this feature is probably of little use. This feature plays a much more significant role if we add pitch prediction to our objective.

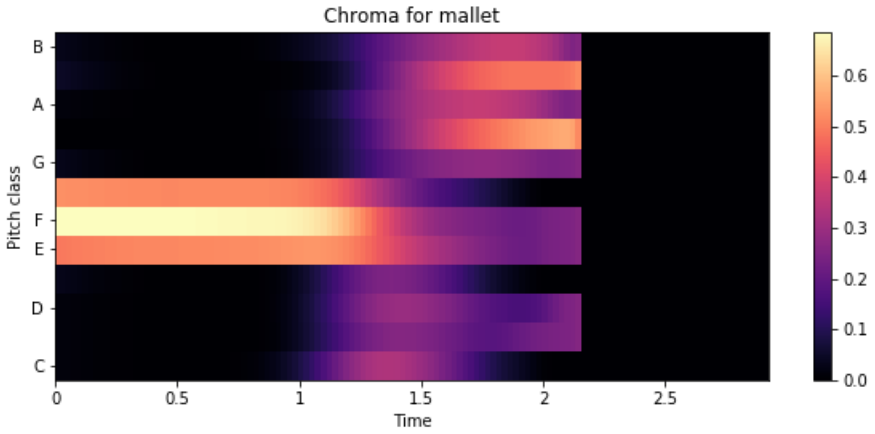


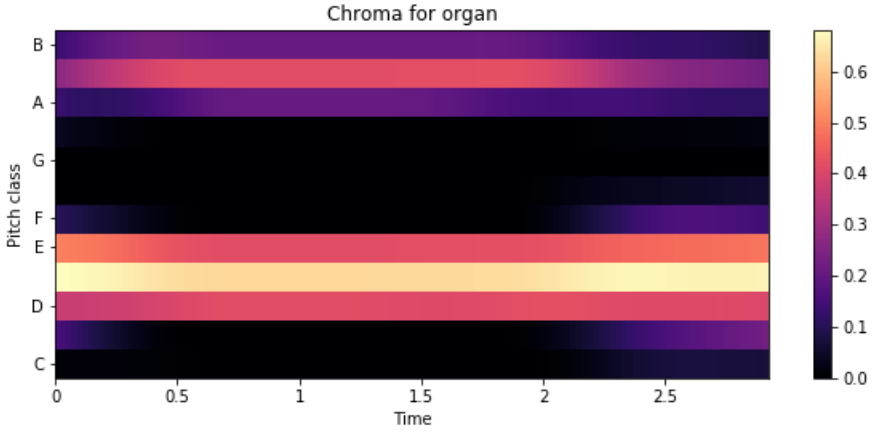

Chroma for brass
Pitch class
B
A
G
F
E
D
C
0
0.5
1
1.5
2
2.5
Time
0.5
0.4
0.3
0.2
0.1

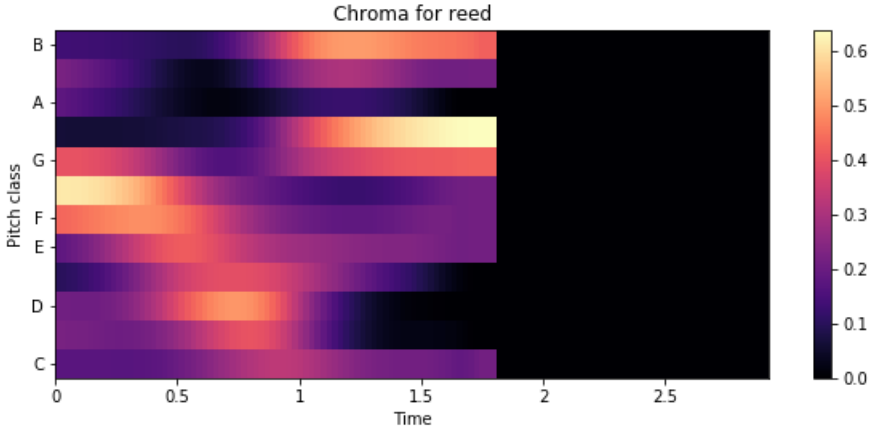

Chroma for flute
Pitch class
B
A
G
F
E
D
C
0
0.5
1
1.5
2
2.5
Time
0.6
0.5
0.4
0.3
0.2
0.1
0.0

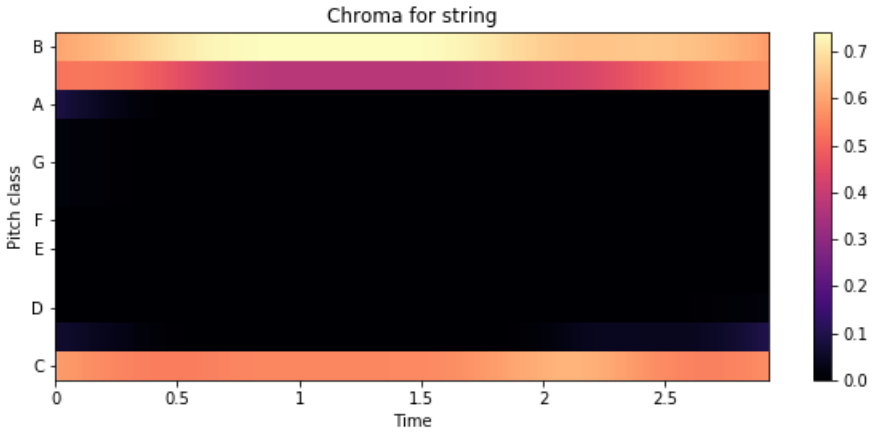

Chroma for guitar
Pitch class
B
A
G
F
E
D
C
0
0.5
1
1.5
2
2.5
Time
0.6
0.5
0.4
0.3
0.2
0.1
0.0

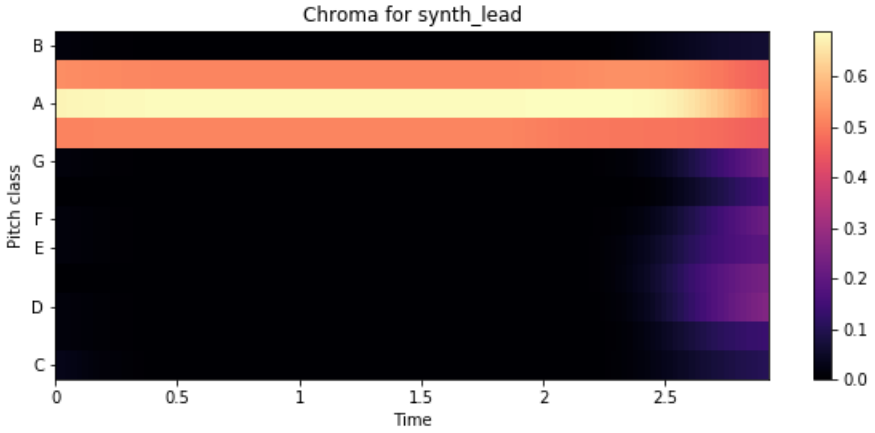

Chroma for keyboard
Pitch class
B
A
G
F
E
D
C
0
0.5
1
1.5
2
2.5
Time
0.6
0.5
0.4
0.3
0.2
0.1
0.0

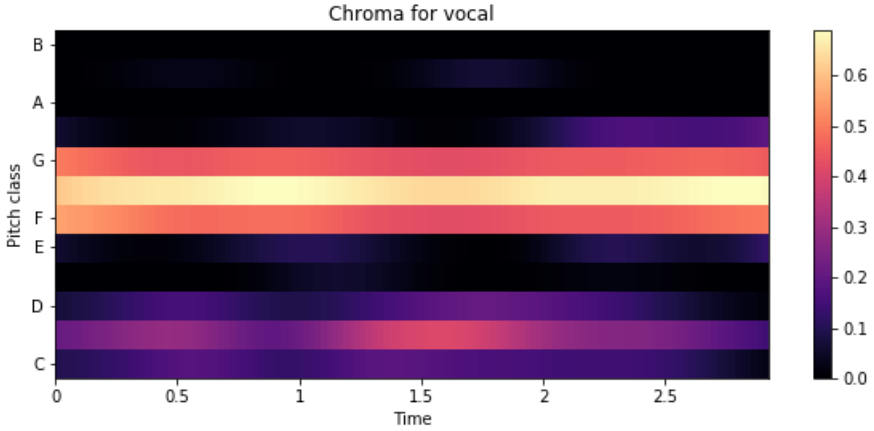

Chroma for mallet
Pitch class
B
A
G
F
E
D
C
0
0.5
1
1.5
2
2.5
Time
0.6
0.5
0.4
0.3
0.2
0.1
0.0


Chroma for organ
Pitch class
B
A
G
F
E
D
C
0
0.5
1
1.5
2
2.5
Time
0.6
0.5
0.4
0.3
0.2
0.1
0.0


Chroma for reed
Pitch class
B
A
G
F
E
D
C
0
0.5
1
1.5
2
2.5
Time
0.6
0.5
0.4
0.3
0.2
0.1
0.0


Chroma for string
Pitch class
B
A
G
F
E
D
C
0
0.5
1
1.5
2
2.5
Time
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0


Chroma for synth_lead
Pitch class
B
A
G
F
E
D
C
0
0.5
1
1.5
2
2.5
Time
0.6
0.5
0.4
0.3
0.2
0.1
0.0



2.2.4 Mel Spectrogram

The spectrogram is a basic tool in audio spectral analysis and other fields. It has been applied extensively in speech analysis. The spectrogram can be defined as an intensity plot (usually on a log scale, such as dB) of the Short-Time Fourier Transform (STFT) magnitude [9]. The STFT is simply a sequence of Fast Fourier Transforms (FFT) of windowed data segments, where the windows are usually allowed to overlap in time, typically by 25-50%. It is an important representation of audio data because human hearing is based on a kind of real-time spectrogram encoded by the cochlea of the inner ear.

The spectrogram has been used extensively in the field of computer music as a guide during the development of sound synthesis algorithms. When working with an appropriate synthesis model, matching the spectrogram often corresponds to matching the sound extremely well. In fact, spectral modeling synthesis (SMS) is based on synthesizing the short-time spectrum directly by some means.

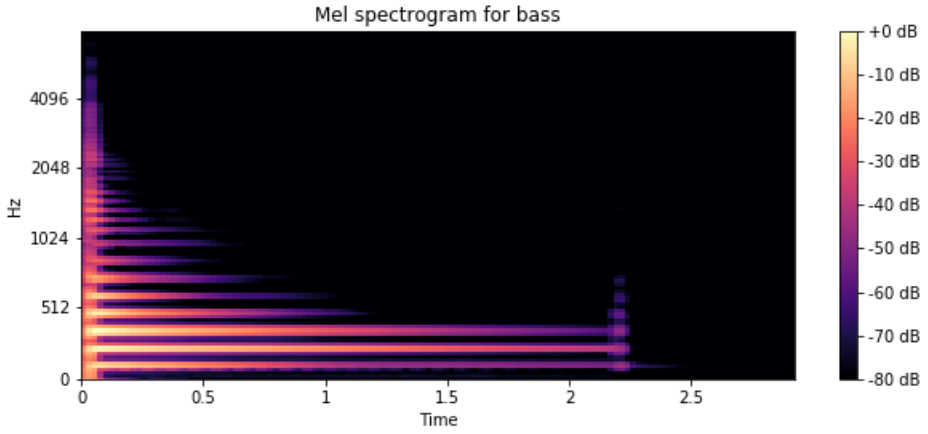
The mel is a unit of pitch. The mel scale is a scale of pitches judged by listeners to be equal in distance one from another. We note that the mel scale is based on empirical evidence and has no theoretical basis. For example, a popular formula to convert a frequency *f* into mels is:

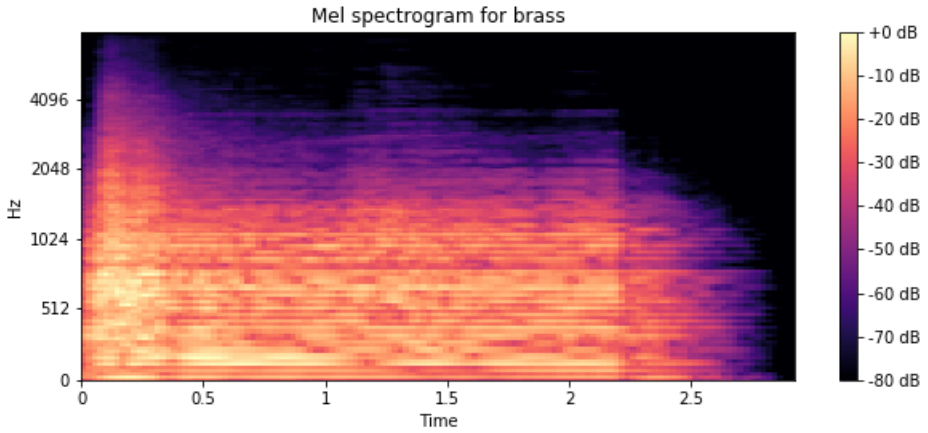
$$m = 2595 log_{10}(1 + \frac{f(Hz)}{700})$$

We opt to use a mel scaled spectrogram because we wish to mimic human perception. We can extract a power spectrogram using librosa.core.stft, which may or may not yield better results.

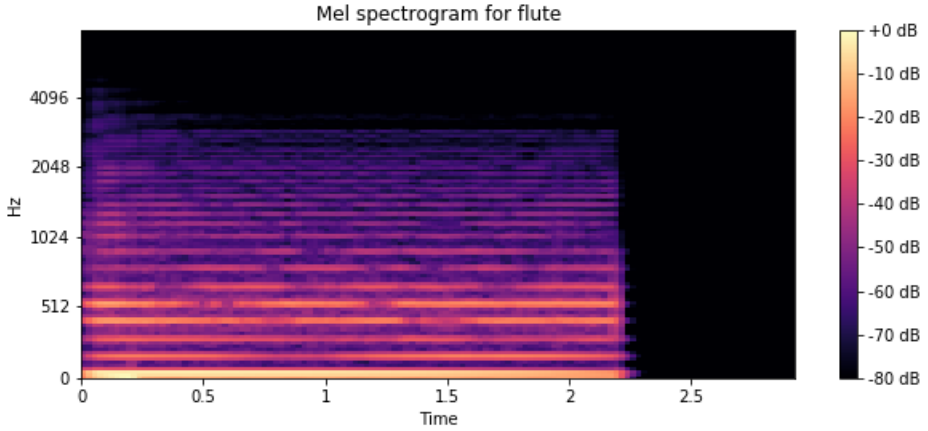
We use librosa.feature.melscpetrogram to generate mel scaled spectrograms. We focus on the *hop_length* parameter which determines the number of samples between successive transforms, this corresponds to the number of audio frames between STFT columns i.e. the allowed overlap in time. Setting a large hop length reduces the size of the spectrogram output and is useful if we want to deal with smaller data.

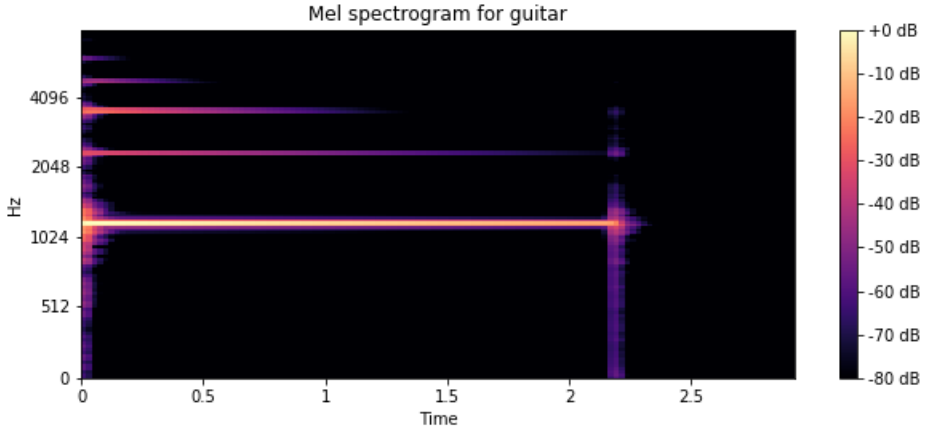
The plots below show the mel scaled spectrogram of our samples. We can clearly see distinct shapes for each instrument. This feature of our data can play an important role in classifying instruments.

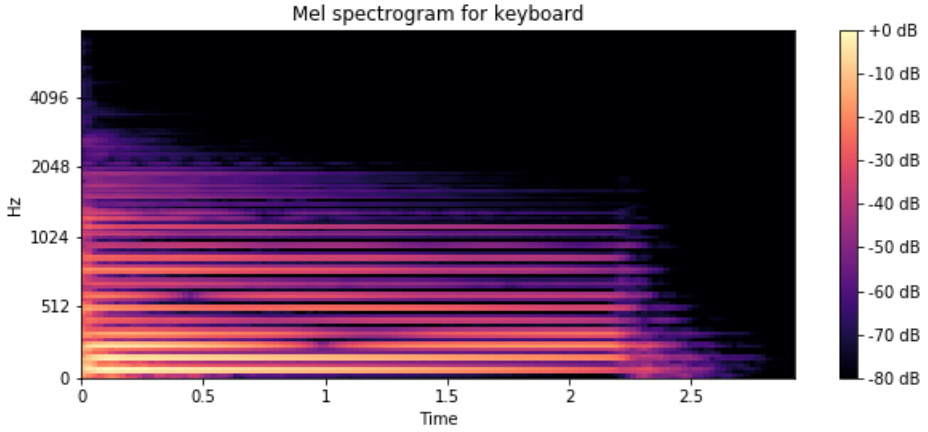




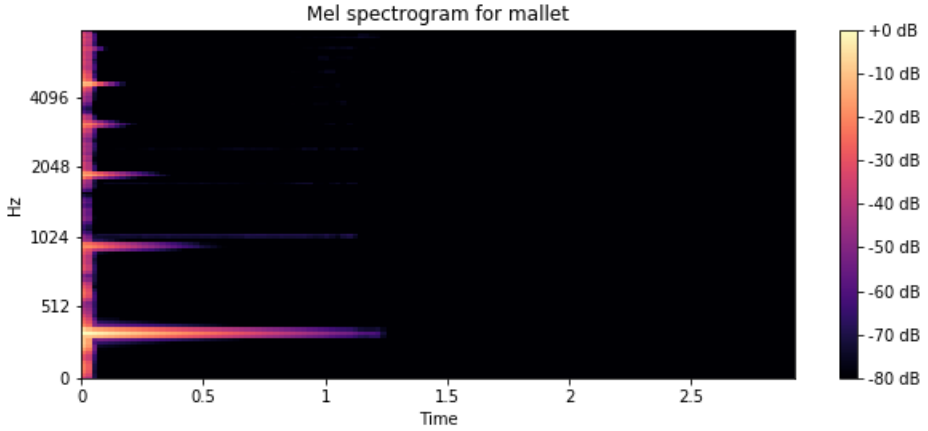
Nadim Kawwa
Udacity Machine Learning Engineer Nanodegree
Capstone Project

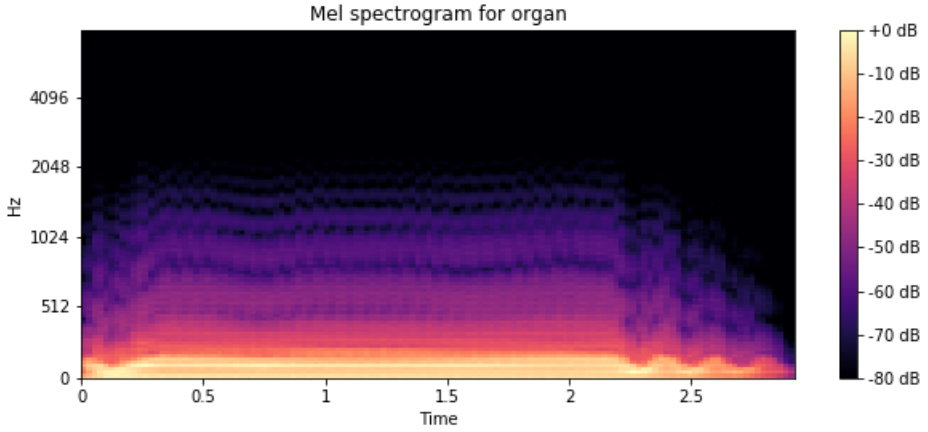


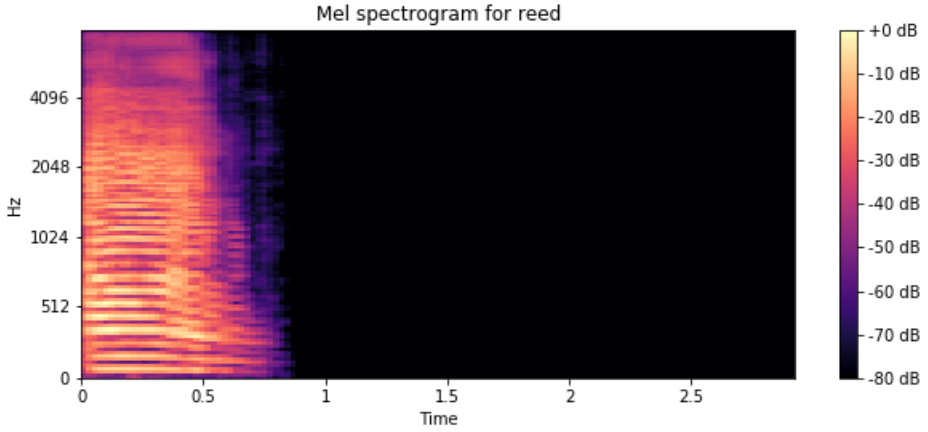


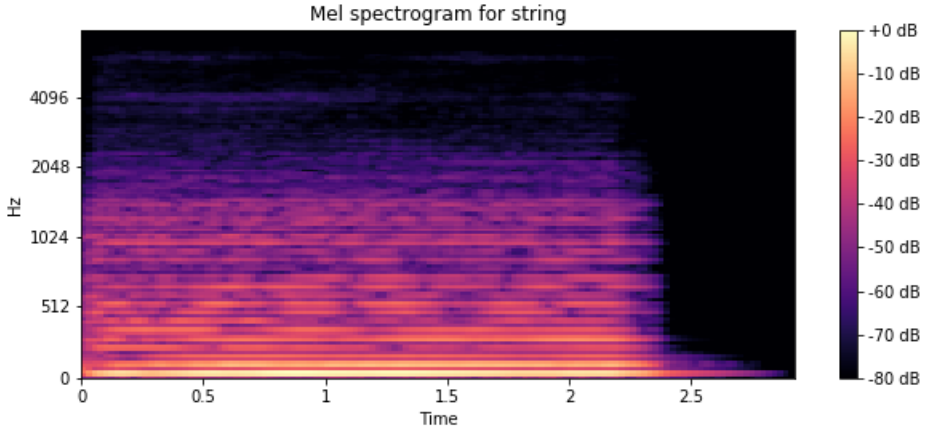


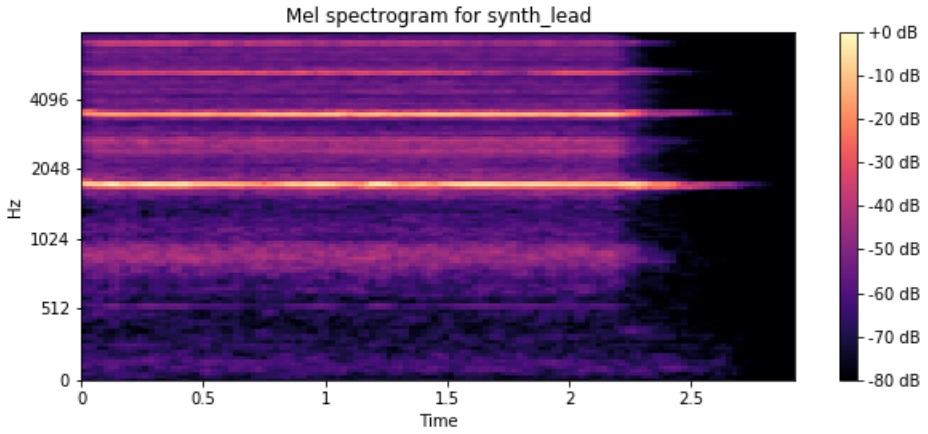
Nadim Kawwa
Udacity Machine Learning Engineer Nanodegree
Capstone Project

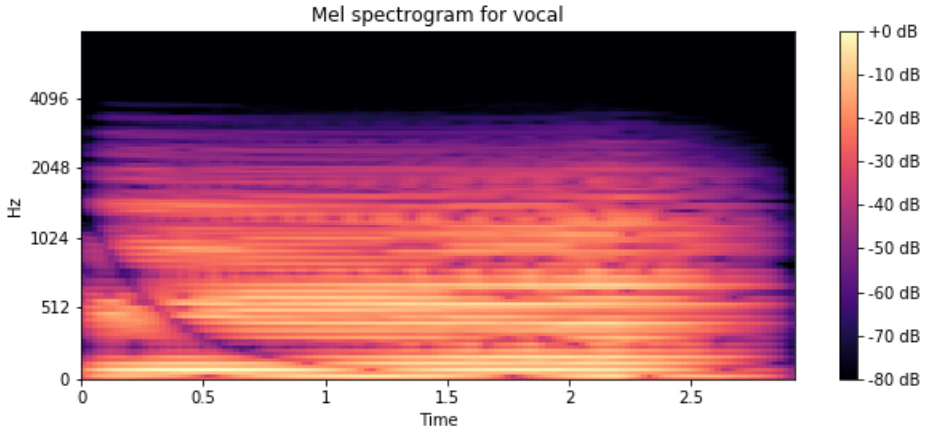












2.2.5 Mel-Frequency Cepstral Coefficients

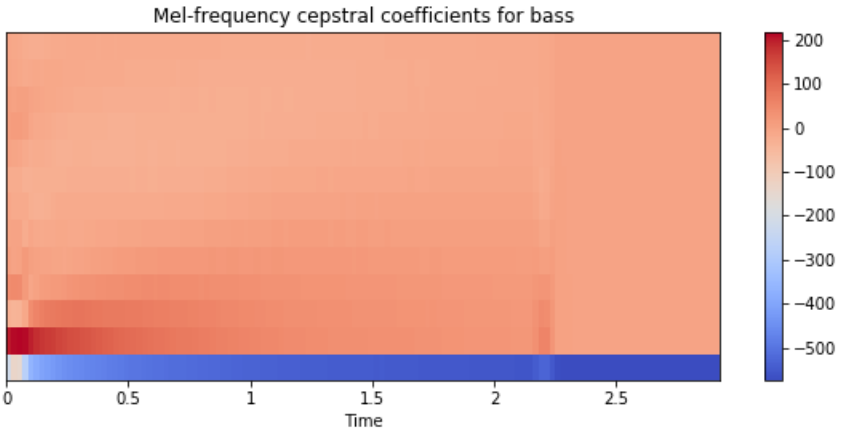
The mel-frequency cepstrum (MFC) is a presentation of the short term power spectrum of a sound, based on a linear cosine transform of a log power spectrum on a nonlinear mel scale of frequency. MFCC coefficients are the coefficients that

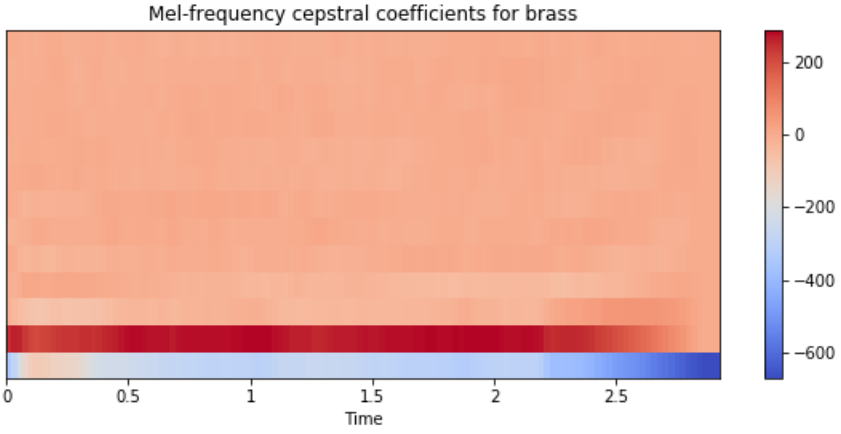
make up an MFCC. Using a mel scale means that the bands are equally scaled, which resembles the human hearing system more than linearly spaced based bands in a normal spectrum.

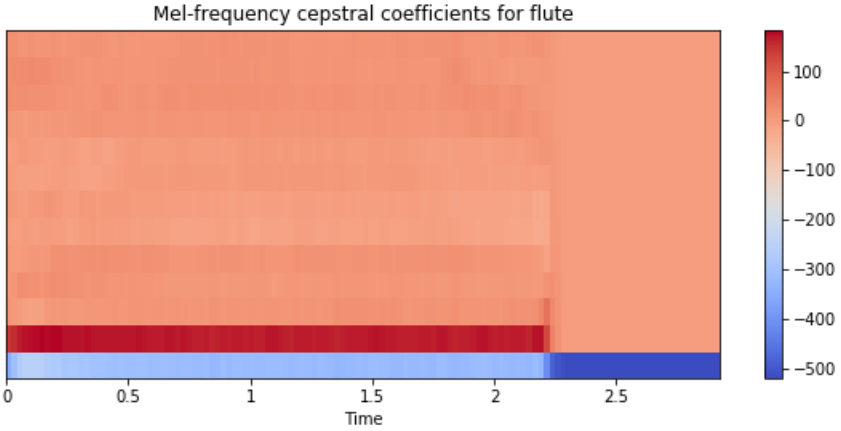
MFCCs are commonly derived as follows:

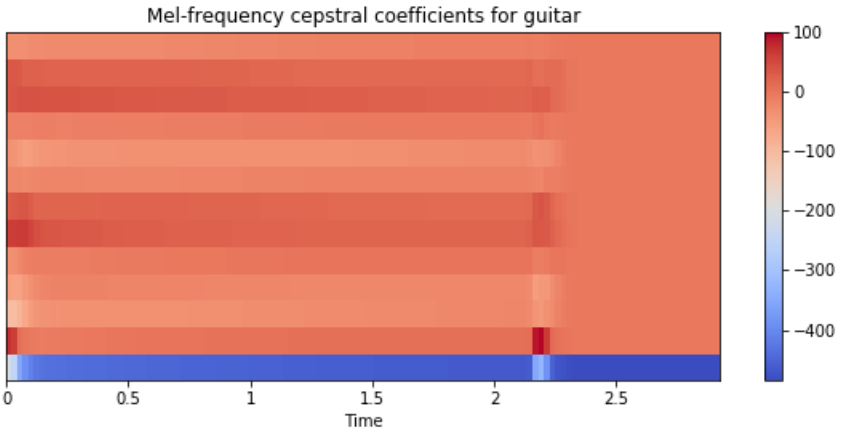
1. Take the Fourier transform of (a windowed excerpt of) a signal.
2. Map the powers of the spectrum obtained above onto the mel scale, using triangular overlapping windows.
3. Take the logs of the powers at each of the mel frequencies.
4. Take the type II discrete cosine transform of the list of mel log powers, as if it were a signal.
5. The MFCCs are the amplitudes of the resulting spectrum.

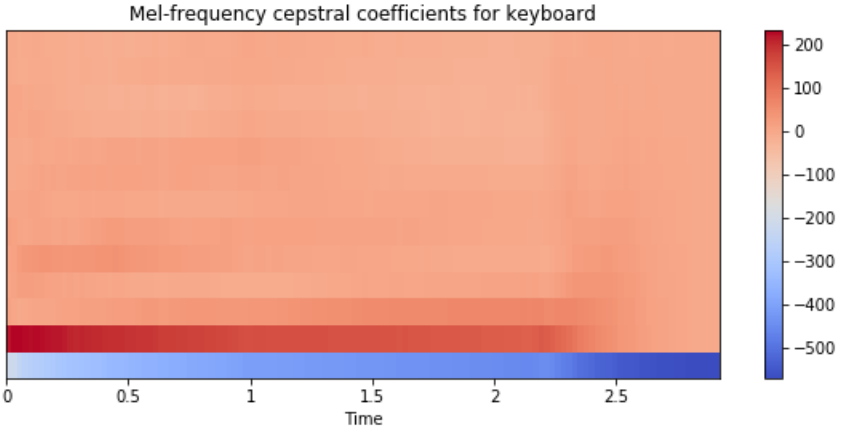
The plots below show the MFCC for each sample using librosa.feature.mfcc.

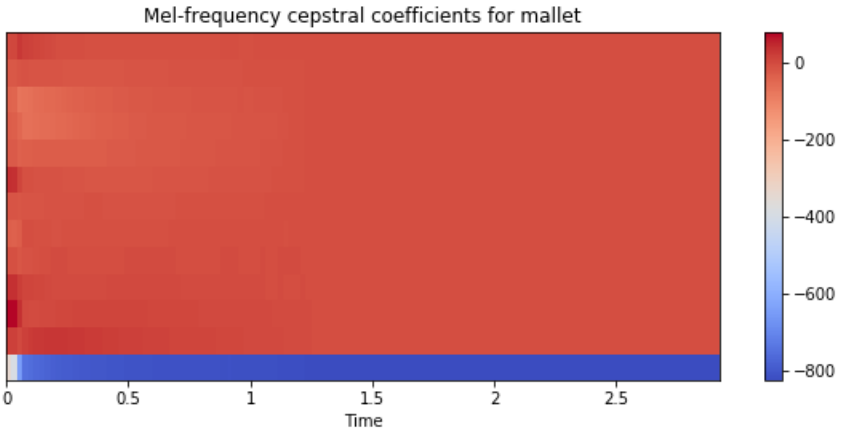


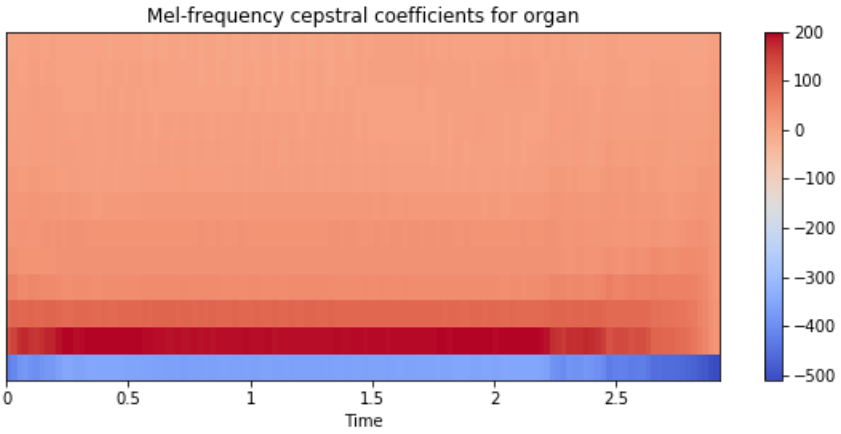


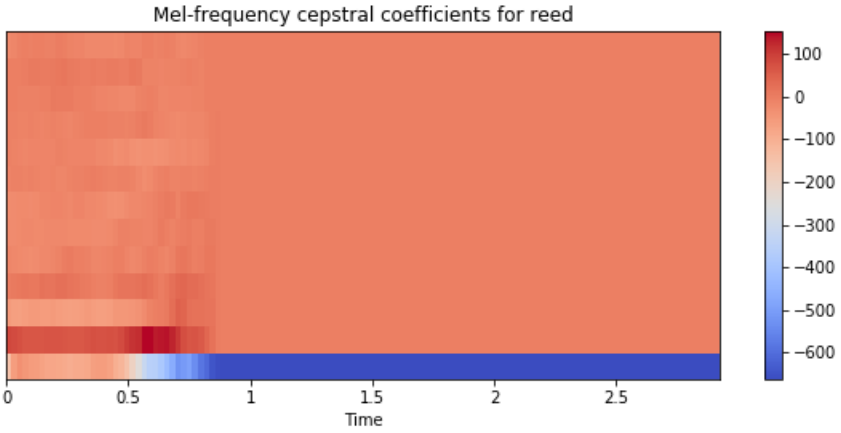

Mel-frequency cepstral coefficients for flute
100
0
−100
−200
−300
−400
−500
0
0.5
1
1.5
2
2.5
Time

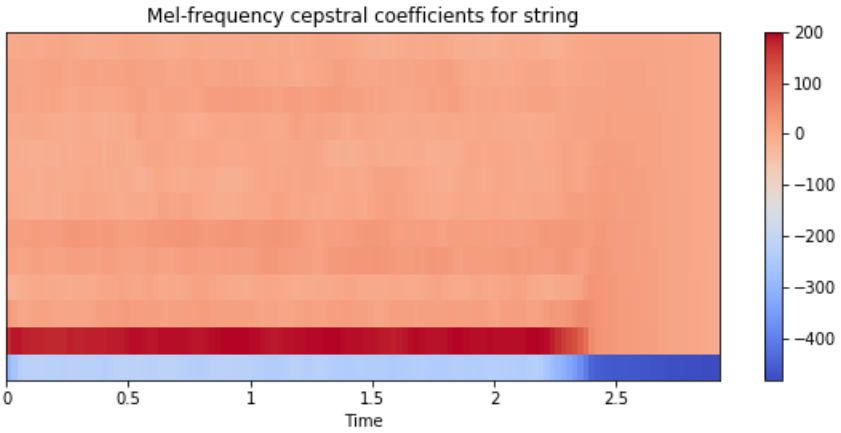

Mel-frequency cepstral coefficients for guitar
100
0
−100
−200
−300
−400
0
0.5
1
1.5
2
2.5
Time

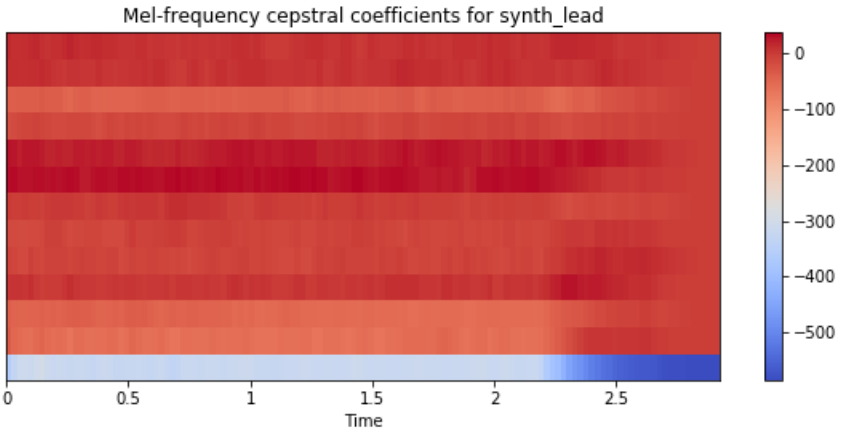

Mel-frequency cepstral coefficients for keyboard
200
100
0
−100
−200
−300
−400
−500
0
0.5
1
1.5
2
2.5
Time

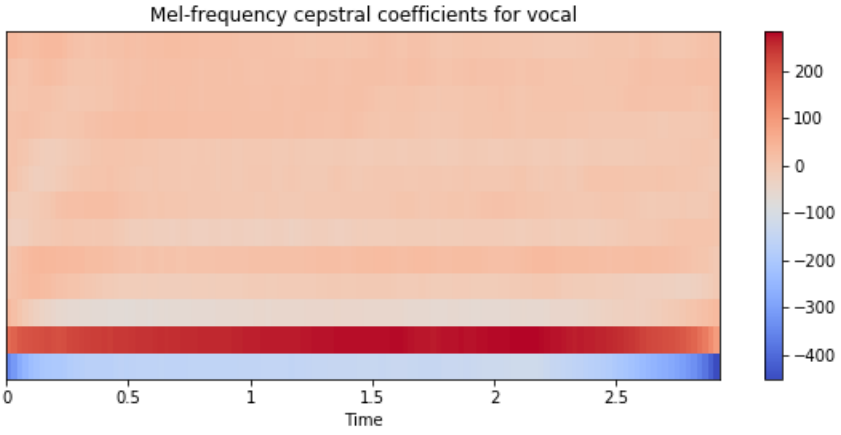

Mel-frequency cepstral coefficients for mallet
0
−200
−400
−600
−800
0
0.5
1
1.5
2
2.5
Time


Mel-frequency cepstral coefficients for organ
200
100
0
−100
−200
−300
−400
−500
0
0.5
1
1.5
2
2.5
Time


Mel-frequency cepstral coefficients for reed
100
0
−100
−200
−300
−400
−500
−600
0
0.5
1
1.5
2
2.5
Time





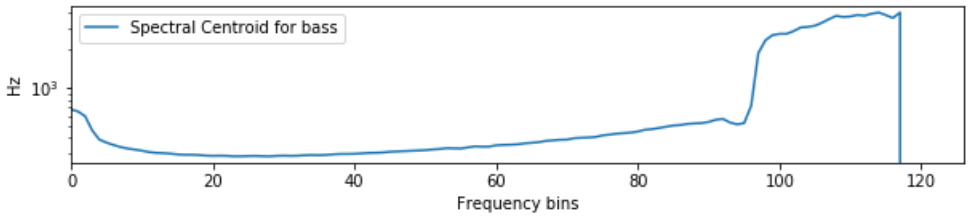


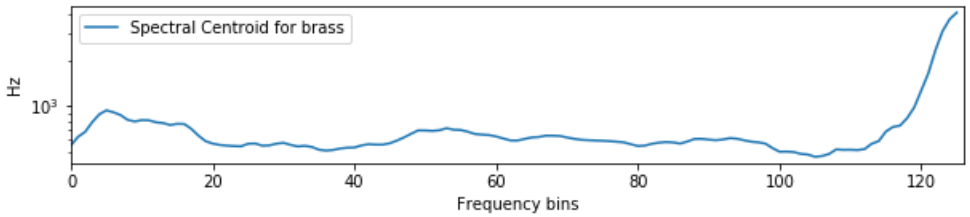
2.2.6 Spectral Centroid

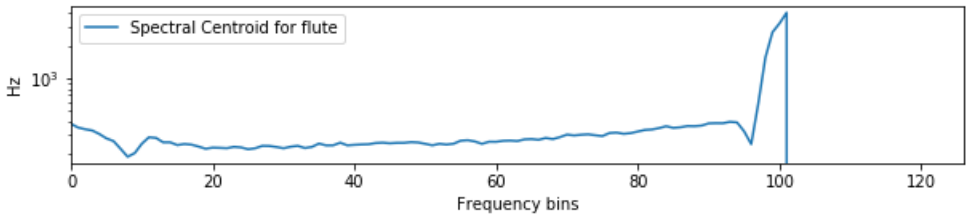
The spectral centroid is commonly associated with the measure of the brightness of a sound [10]. This measure is obtained by evaluating the “center of gravity” using the Fourier transform’s frequency and magnitude information. The

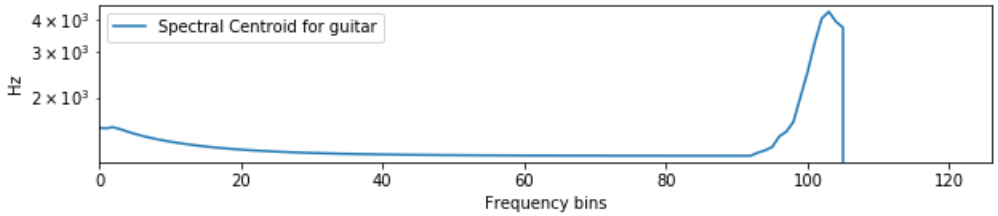
individual centroid of a spectral frame is defined as the average frequency weighted by amplitudes, divided by the sum of the amplitudes.

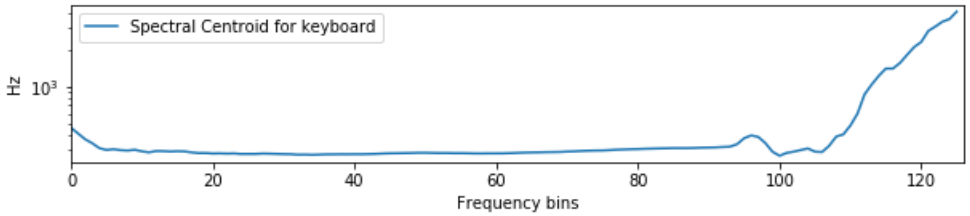
The plots below show the spectral centroids for the samples using librosa.feature.spectral_centroid. The plots look like a promising feature since each instrument shows a distinct pattern over time.

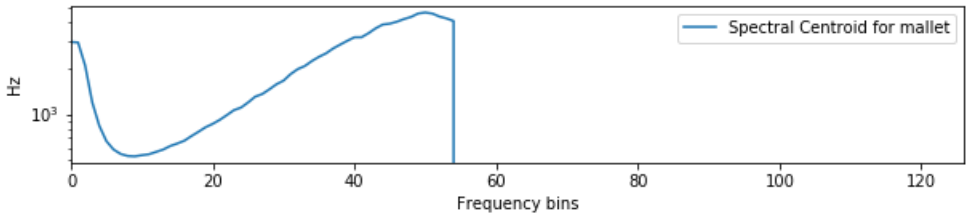


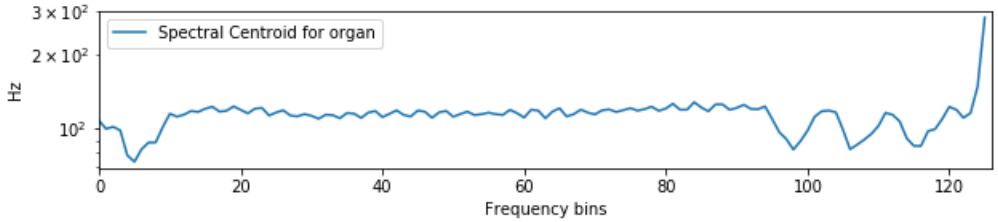


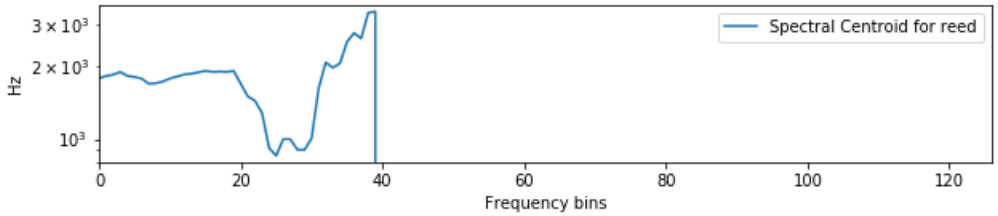


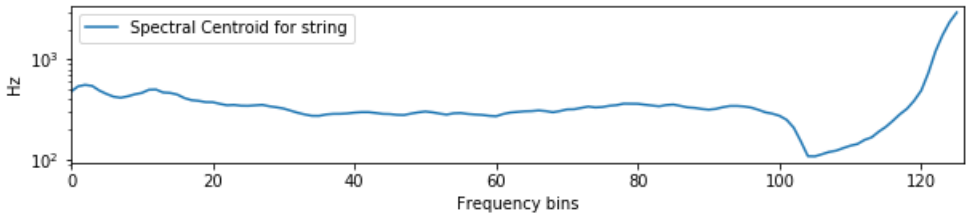

Spectral Centroid for guitar
4×10^3
3×10^3
2×10^3
Hz
0
20
40
60
80
100
120
Frequency bins

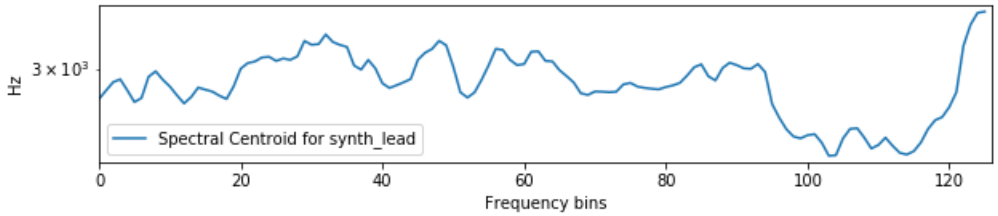

Spectral Centroid for keyboard
10^3
Hz
0
20
40
60
80
100
120
Frequency bins

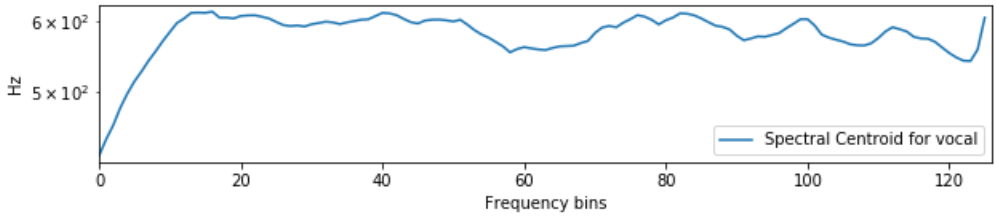

Spectral Centroid for mallet
10^3
Hz
0
20
40
60
80
100
120
Frequency bins


Spectral Centroid for organ
Hz
Frequency bins


Spectral Centroid for reed
Hz
Frequency bins


Spectral Centroid for string
Hz
Frequency bins

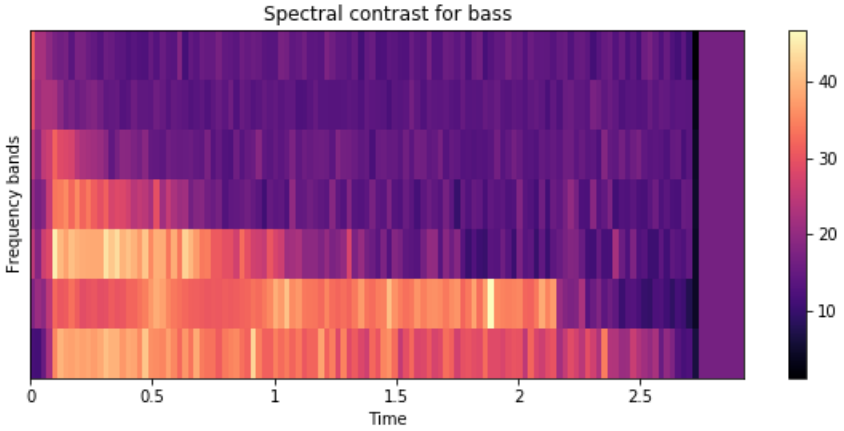


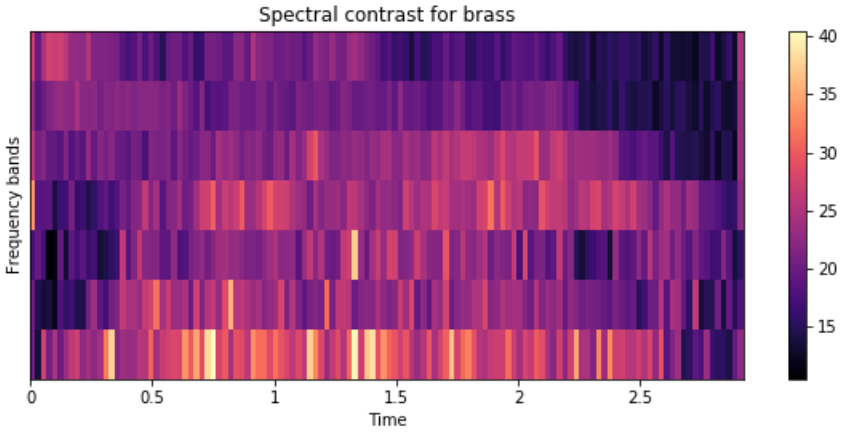


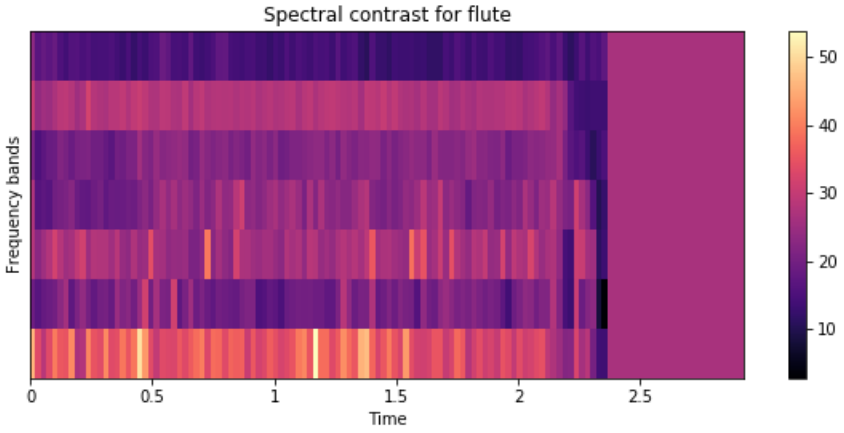
2.2.7 Spectral Contrast

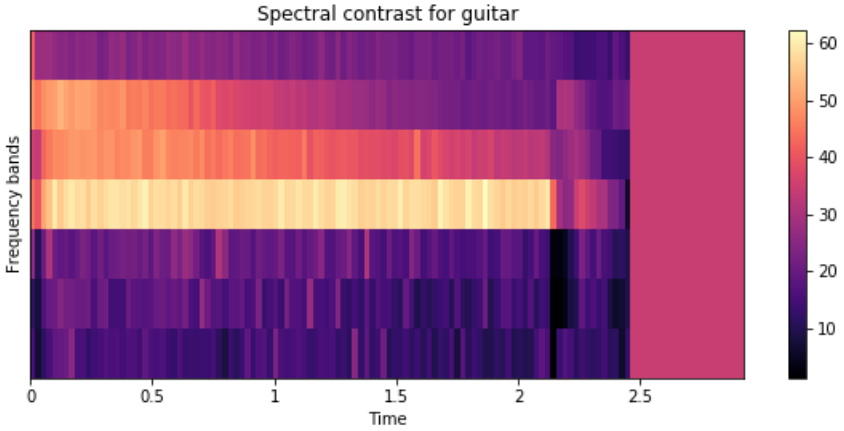
Spectral contrast is defined as the level difference between peaks and valleys in the spectrum using the method described by Jian et al[11]. Octave-based Spectral Contrast considers the spectral peak, spectral valley and their difference in each sub-band.

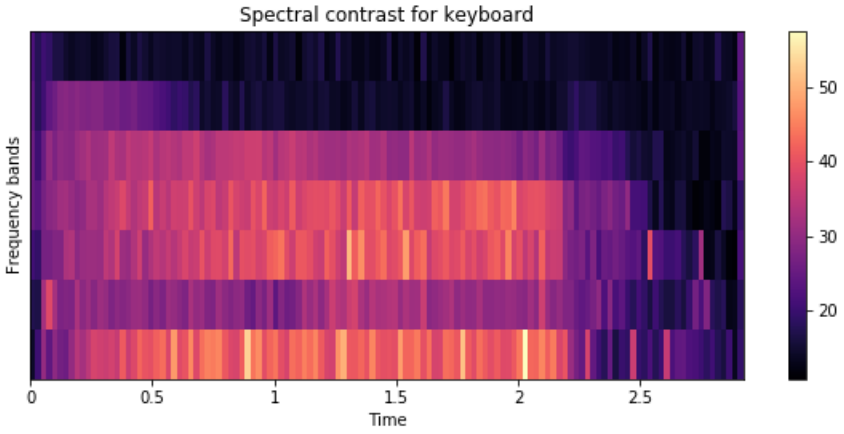
The plots below show the spectral contrasts for the samples using librosa.feature.spectral_contrast. The spectral contrast appears to be a solid way to classify instruments, given also that this feature was previously used by Jian et al to classify instruments.

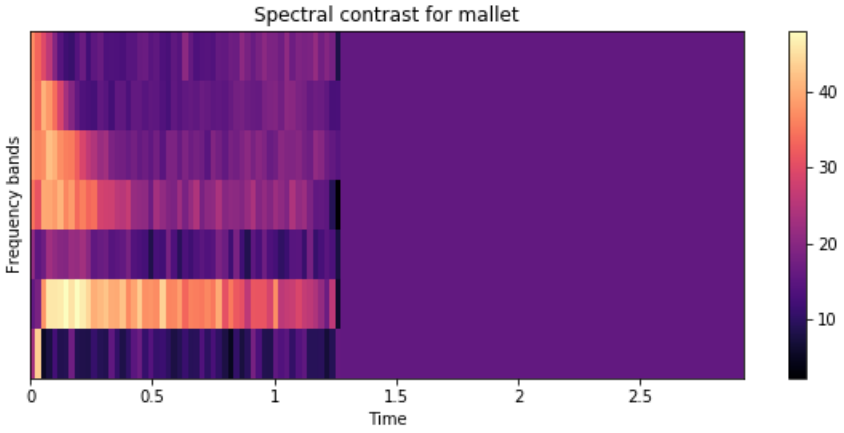

Spectral contrast for bass
Frequency bands
Time
0
0.5
1
1.5
2
2.5
40
30
20
10

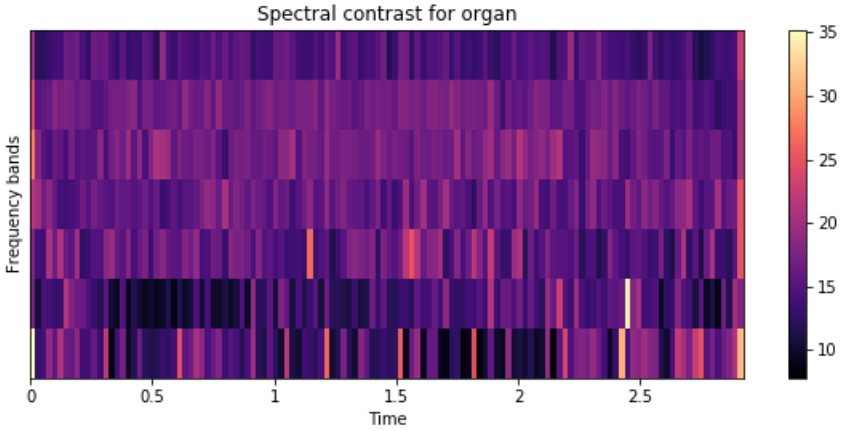

Spectral contrast for brass
Frequency bands
Time
0
0.5
1
1.5
2
2.5
40
35
30
25
20
15

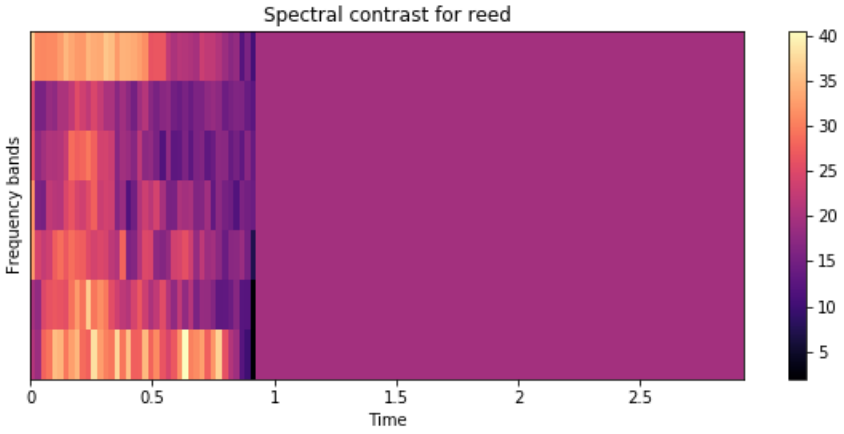

Spectral contrast for flute
Frequency bands
Time
0
0.5
1
1.5
2
2.5
50
40
30
20
10

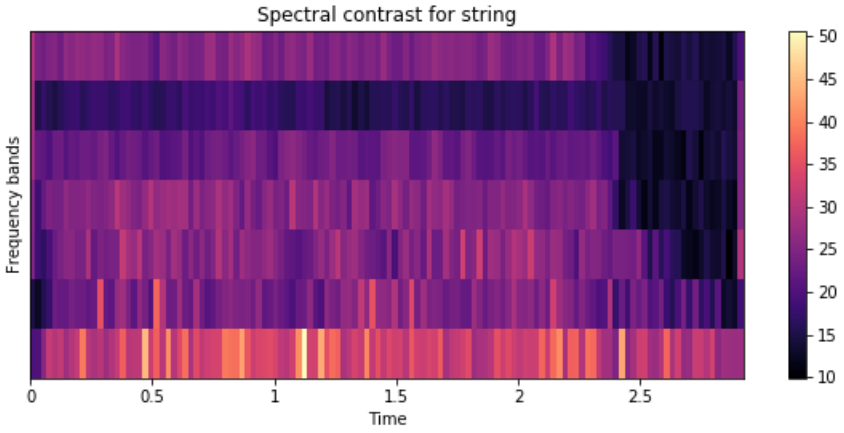

Spectral contrast for guitar
Frequency bands
Time
0
0.5
1
1.5
2
2.5
60
50
40
30
20
10

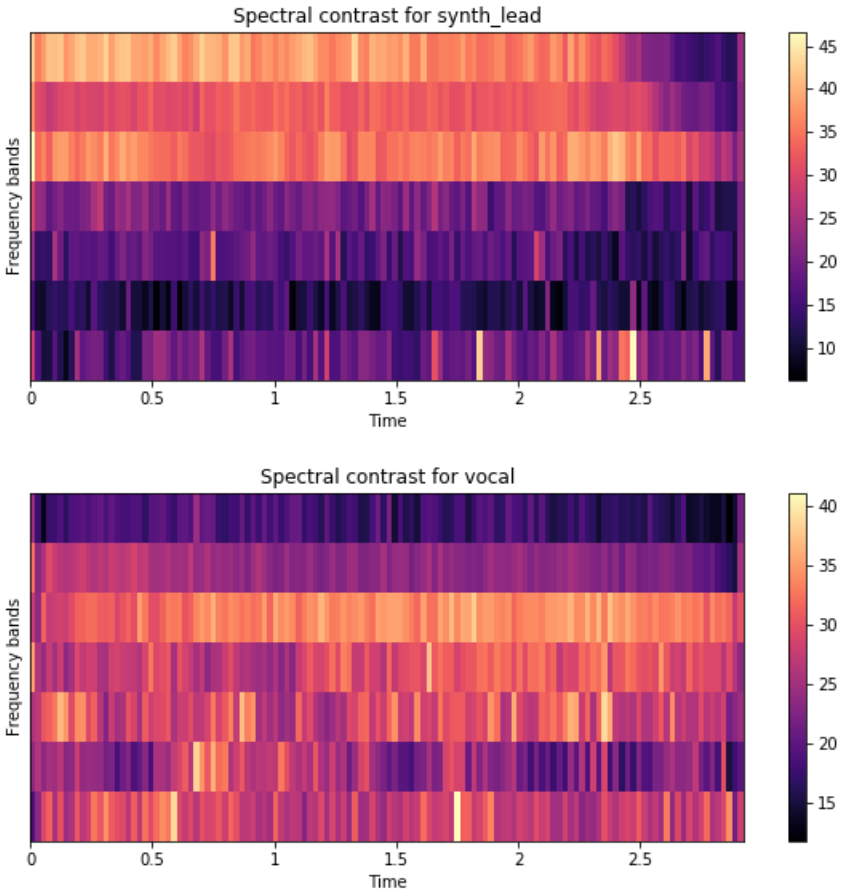

Spectral contrast for keyboard
Frequency bands
Time
0
0.5
1
1.5
2
2.5
50
40
30
20


Spectral contrast for mallet
Frequency bands
Time
0
0.5
1
1.5
2
2.5
40
30
20
10


Spectral contrast for organ
Frequency bands
Time
0
0.5
1
1.5
2
2.5
35
30
25
20
15
10


Spectral contrast for reed
Frequency bands
Time
0
0.5
1
1.5
2
2.5
40
35
30
25
20
15
10
5

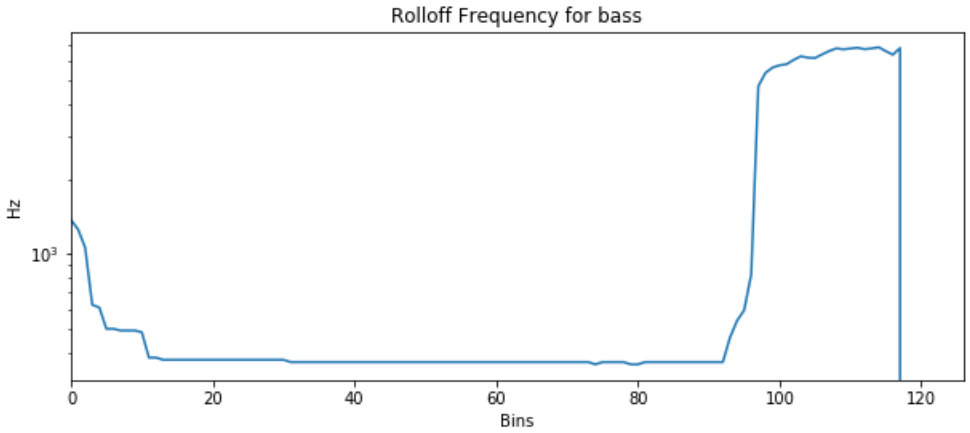

Spectral contrast for string
Frequency bands
Time
0
0.5
1
1.5
2
2.5
50
45
40
35
30
25
20
15
10

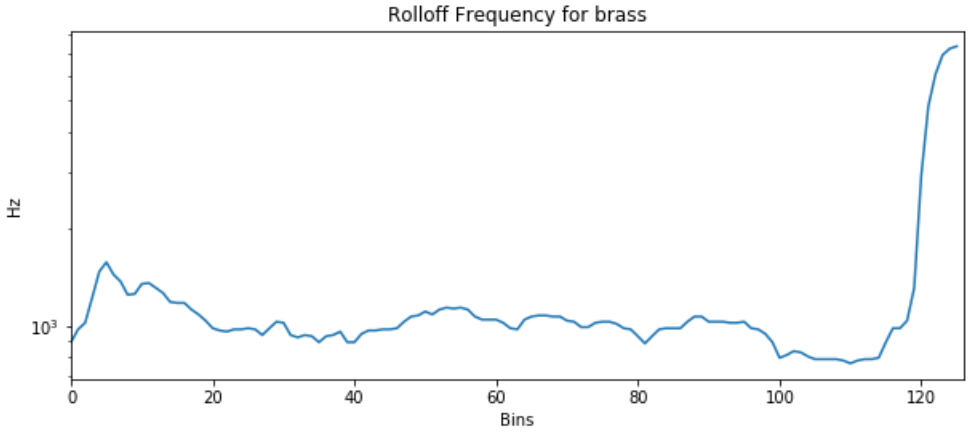


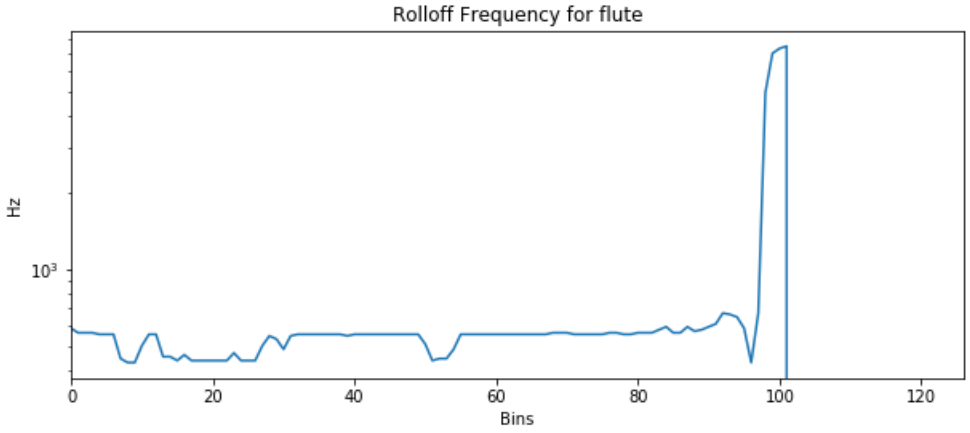
2.2.8 Spectral Rolloff Frequency

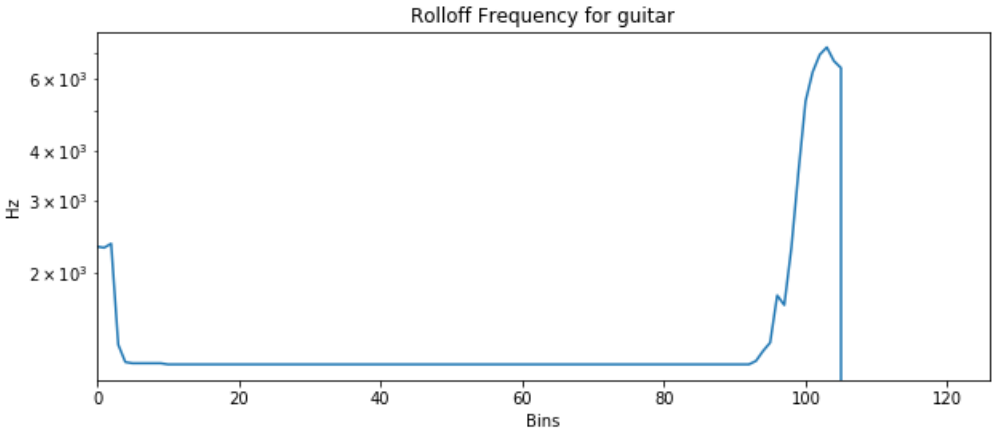
The rolloff frequency is defined for each frame as the center frequency for a spectrogram bin such that at least the roll off percent (0.85 by default) of the energy of the spectrum in this frame is contained in this bin and the bins below.

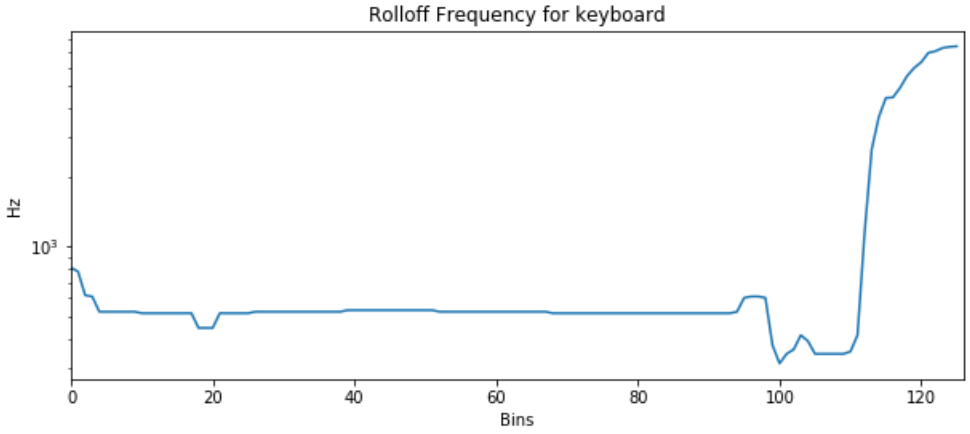
The plots below show the spectral rolloff frequencies for the instruments using librosa.feature.spectral_rolloff.

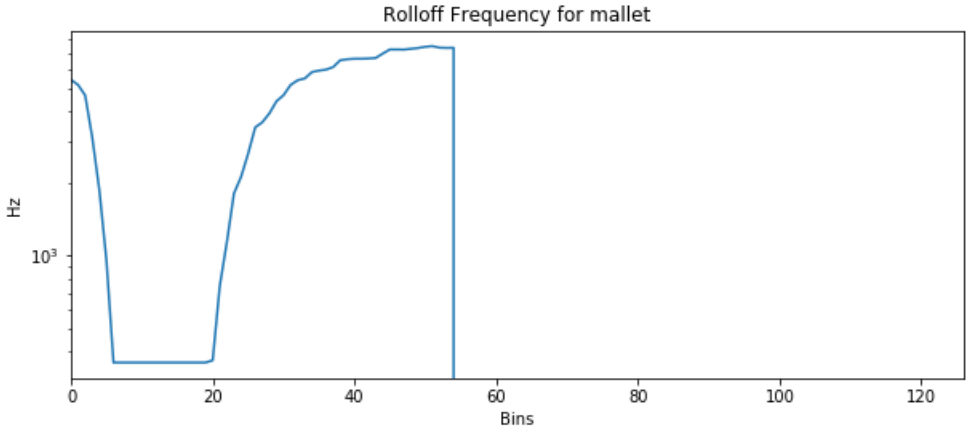

Rolloff Frequency for bass
Hz
10^3
0
20
40
60
80
100
120
Bins

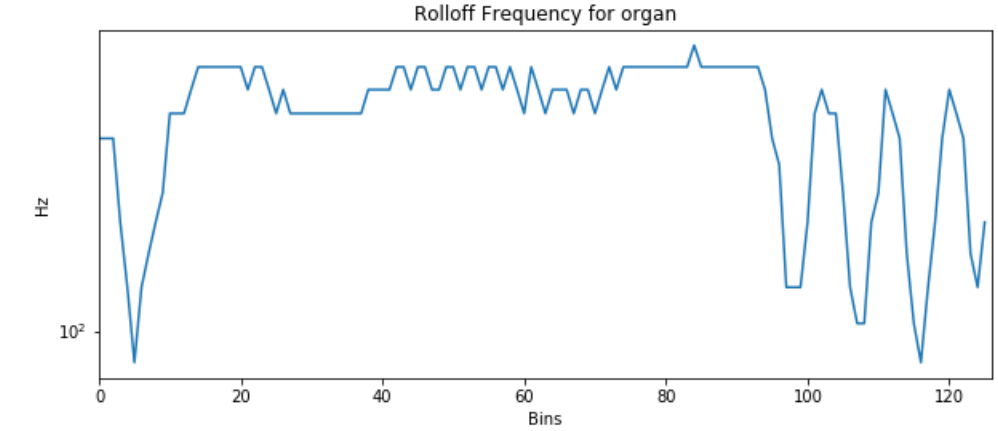

Rolloff Frequency for brass
Hz
10^3
0
20
40
60
80
100
120
Bins

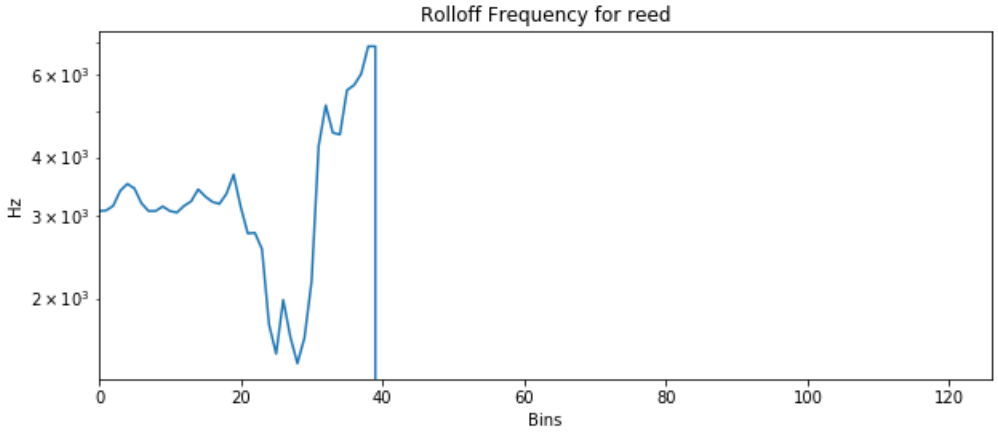

Rolloff Frequency for flute
Hz
10^3
0
20
40
60
80
100
120
Bins

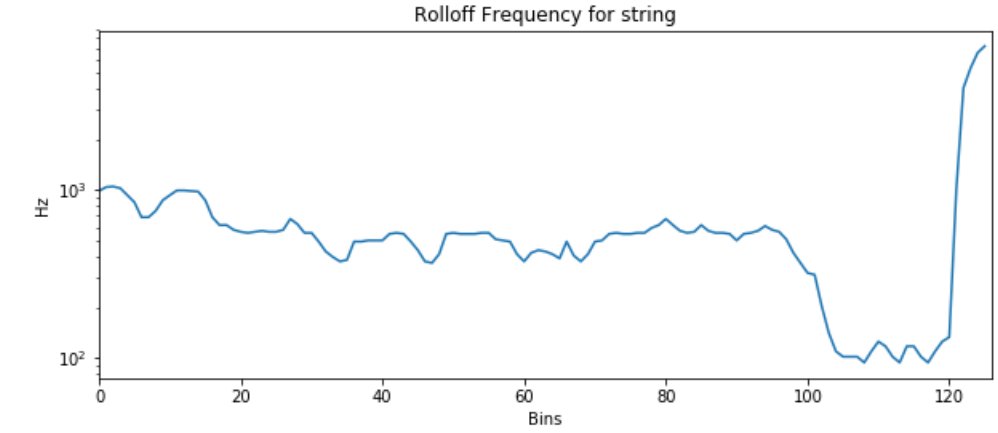

Rolloff Frequency for guitar
Hz
6 × 10³
4 × 10³
3 × 10³
2 × 10³
0
20
40
60
80
100
120
Bins

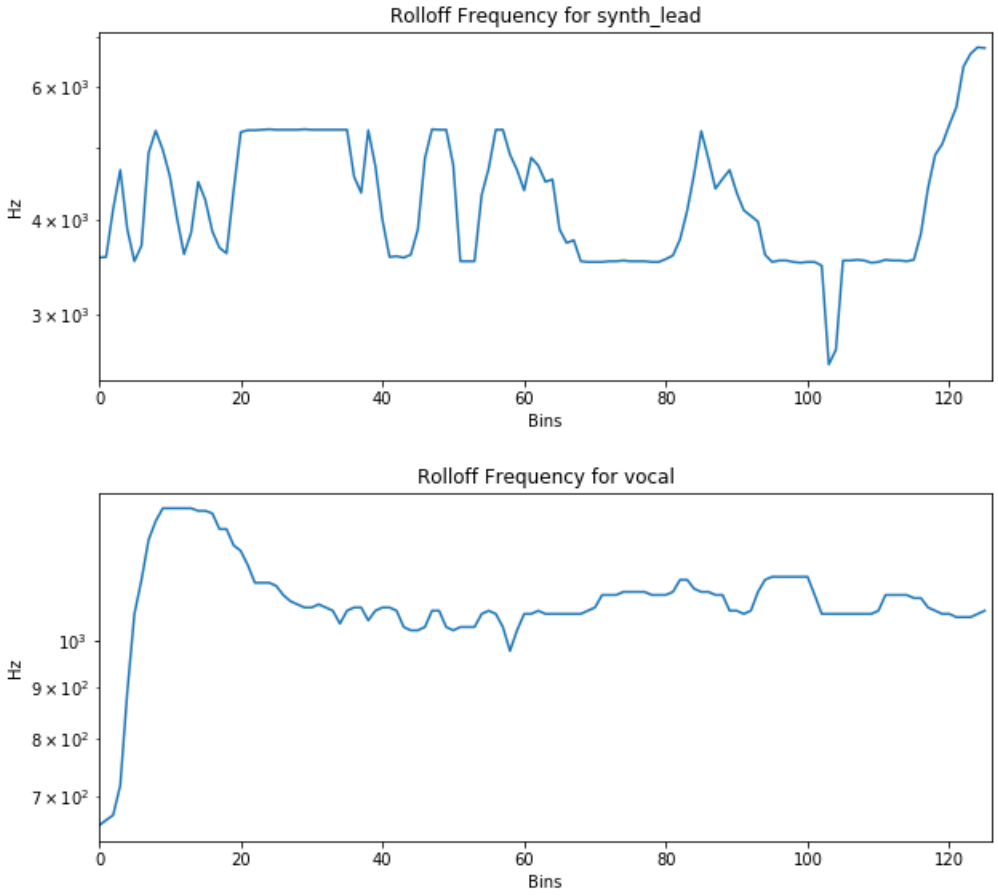

Rolloff Frequency for keyboard
Hz
10³
0
20
40
60
80
100
120
Bins


Rolloff Frequency for mallet
Hz
10³
0
20
40
60
80
100
120
Bins


Rolloff Frequency for organ
Hz
10^2
0
20
40
60
80
100
120
Bins


Rolloff Frequency for reed
Hz
6×10^3
4×10^3
3×10^3
2×10^3
0
20
40
60
80
100
120
Bins


Rolloff Frequency for string
Hz
10^3
10^2
0
20
40
60
80
100
120
Bins



2.2.9 JSON Files

We then explore the non-audio features present in the dataset json files. From the NSynth documentation, there are 14 features per Table 4 below. We will only visualize the features that we assume bring us closer to our goal. We also note that the dataset contains no missing values, therefore no data cleaning is necessary. In addition, the sections below reveal that we can't call out outliers in the features.

The exploration of the JSON files can be found here:

https://github.com/NadimKawwa/NSynth/blob/master/01_NSynth_Exploration_JSON_Files.ipynb

Table 4 Non Audio Features in json files

Feature	Type	Description
note	int64	A unique integer identifier for the note.
note_str	bytes	A unique string identifier for the note in the format <instrument_str>-<pitch>-<velocity>.
instrument	int64	A unique, sequential identifier for the instrument the note was synthesized from.
instrument_str	bytes	A unique string identifier for the instrument this note was synthesized from in the format <instrument_family_str>-<instrument_production_str>-<instrument_name>.
pitch	int64	The 0-based MIDI pitch in the range [0, 127].
velocity	int64	The 0-based MIDI velocity in the range [0, 127].
sample_rate	int64	The samples per second for the audio feature.
audio*	[float]	A list of audio samples represented as floating point values in the range [-1,1].

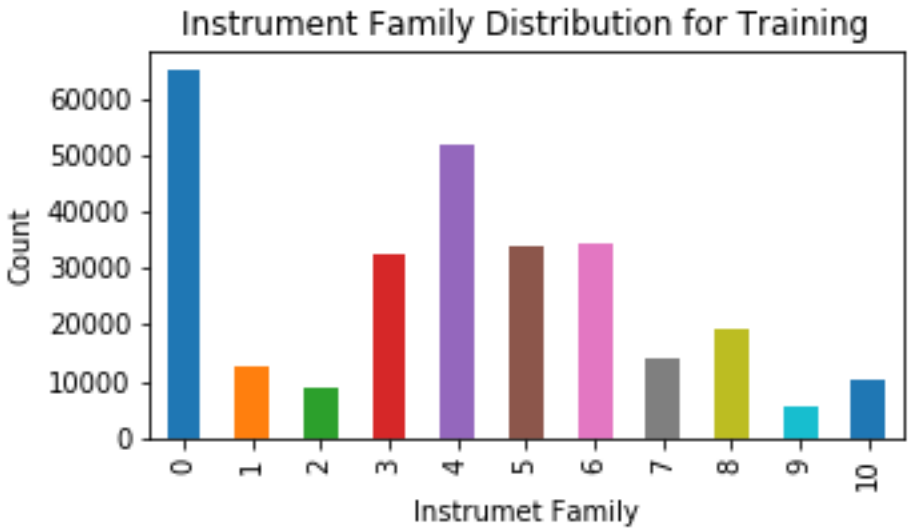
Feature	Type	Description
qualities	[int64]	A binary vector representing which sonic qualities are present in this note.
qualities_str	[bytes]	A list IDs of which qualities are present in this note selected from the sonic qualities list.
instrument_family	int64	The index of the instrument family this instrument is a member of.
instrument_family_str	bytes	The ID of the instrument family this instrument is a member of.
instrument_source	int64	The index of the sonic source for this instrument.
instrument_source_str	bytes	The ID of the sonic source for this instrument.

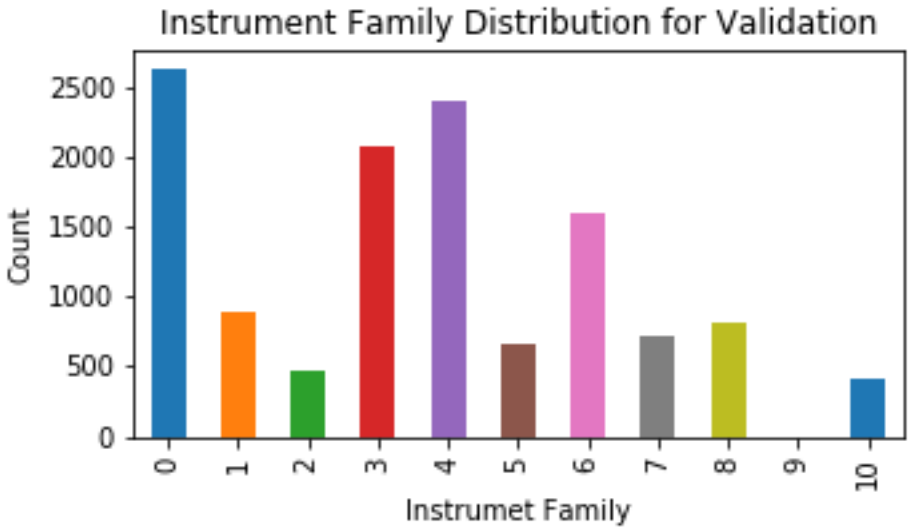
We begin by visualizing the distribution of instrument families from the plots below. We clearly see an imbalanced dataset where some instruments occur more frequently than others, sometimes by order of magnitude for the training dataset.

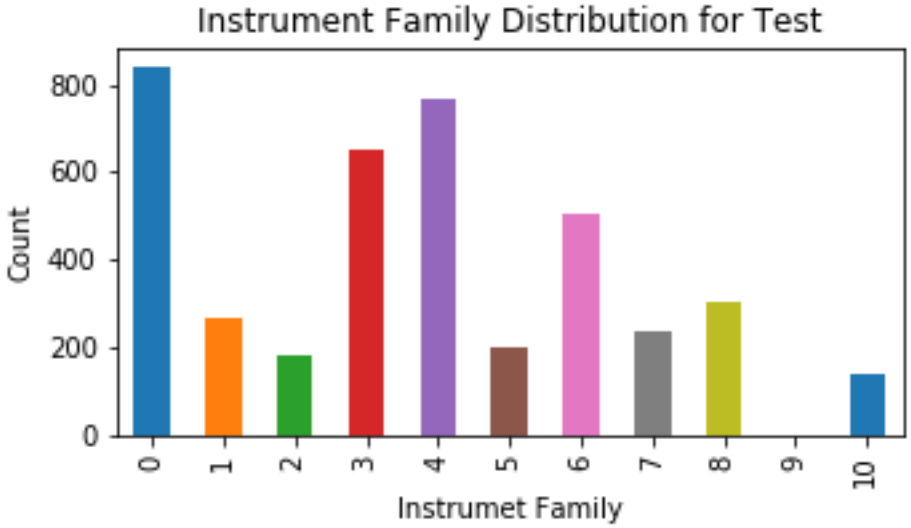
In addition, it is worth nothing that synth_lead (instrument 9) is missing in the validation and test datasets. In summary the datasets counts are distributed as follows:

- Training: 289,205
- Validation: 12,678
- Testing: 4,096

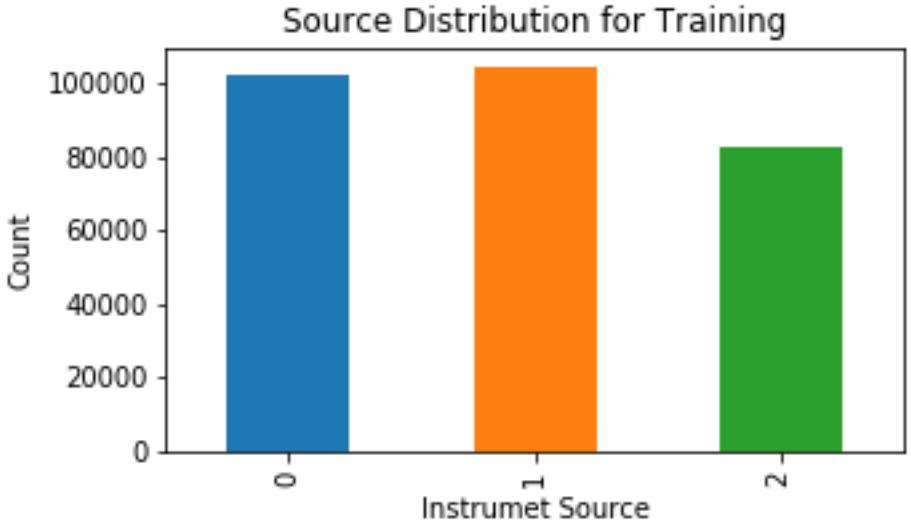
Upon further exploration of the data we also notice that all wave files possess the sample sampling rate and

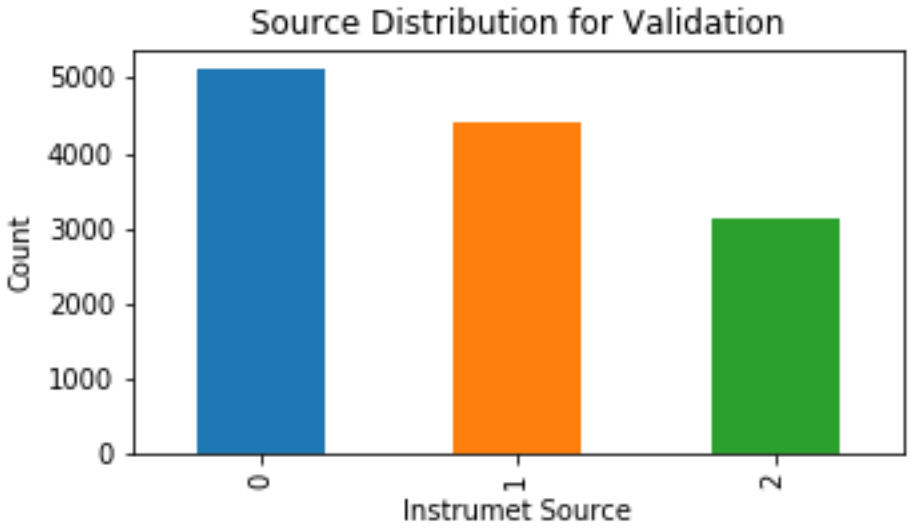


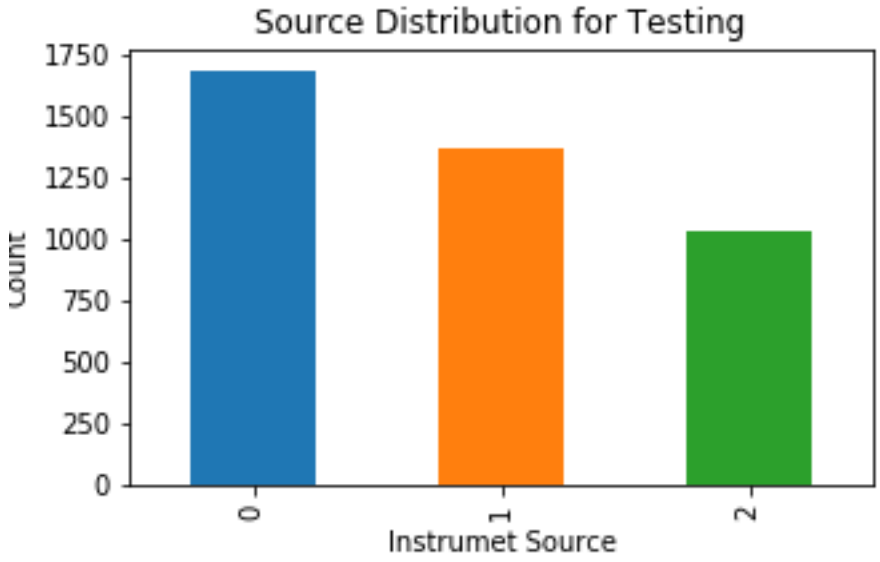




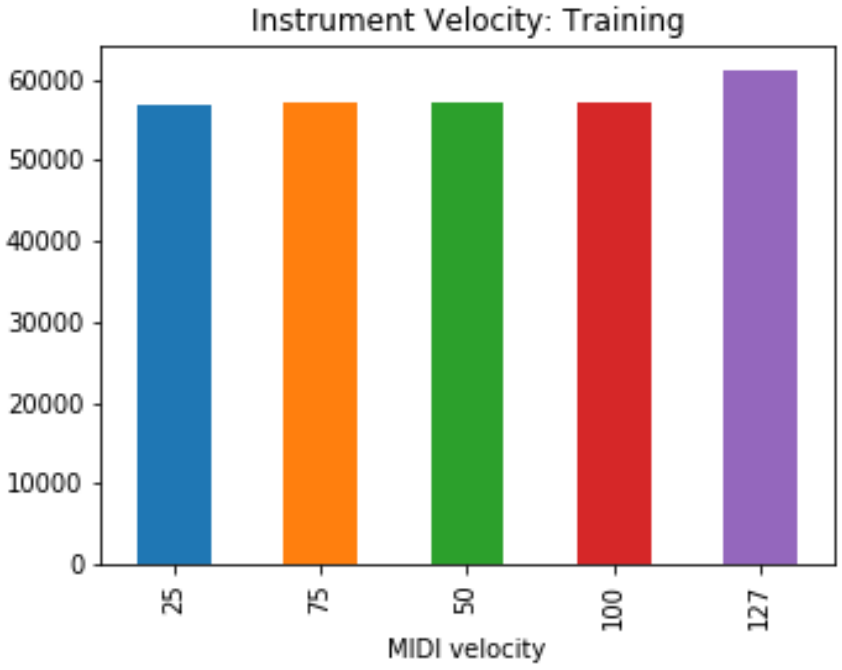
A sound can be acoustic, electronic or synthetic. The distribution is more or less balanced among the three. For the purpose of our exercise, it is not necessary to explore the method of sound production.

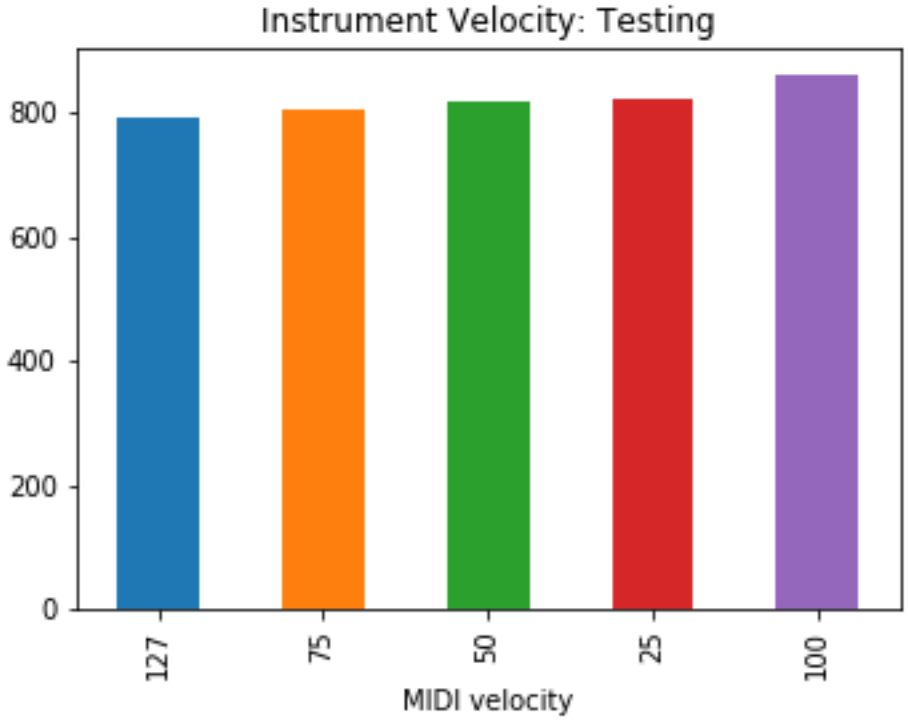

Source Distribution for Training
100000
80000
60000
40000
20000
0
Count
0
1
2
Instrumet Source

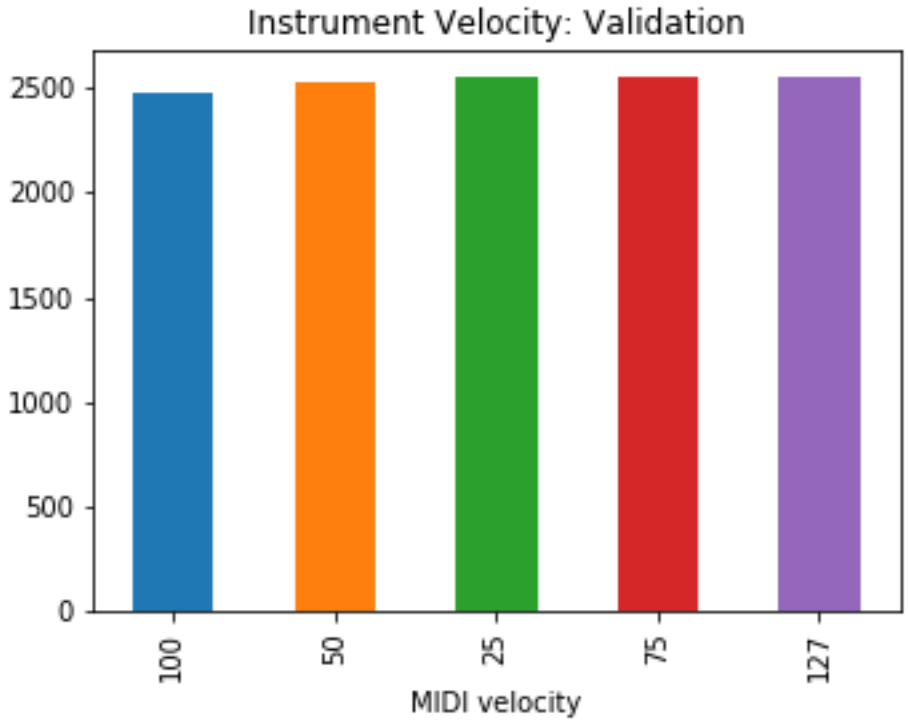

Source Distribution for Validation
5000
4000
3000
2000
1000
0
Count
0
1
2
Instrumet Source



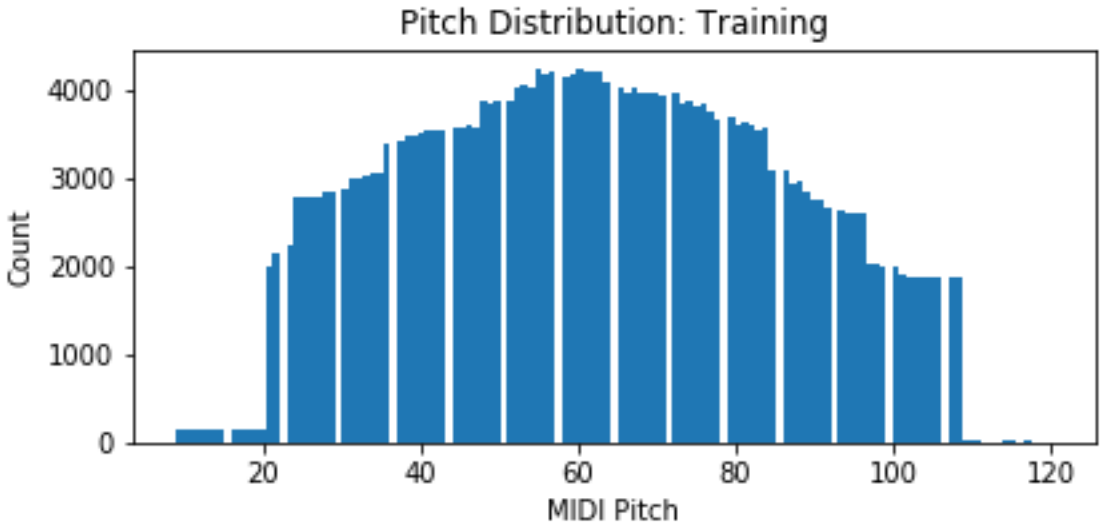
We also visualize the velocity distributions among the datasets, sorted by count. The velocities are more or less evenly distributed among the dataset. In addition, the same unique velocities are repeated across the datasets: [50, 127, 25, 75, 100].

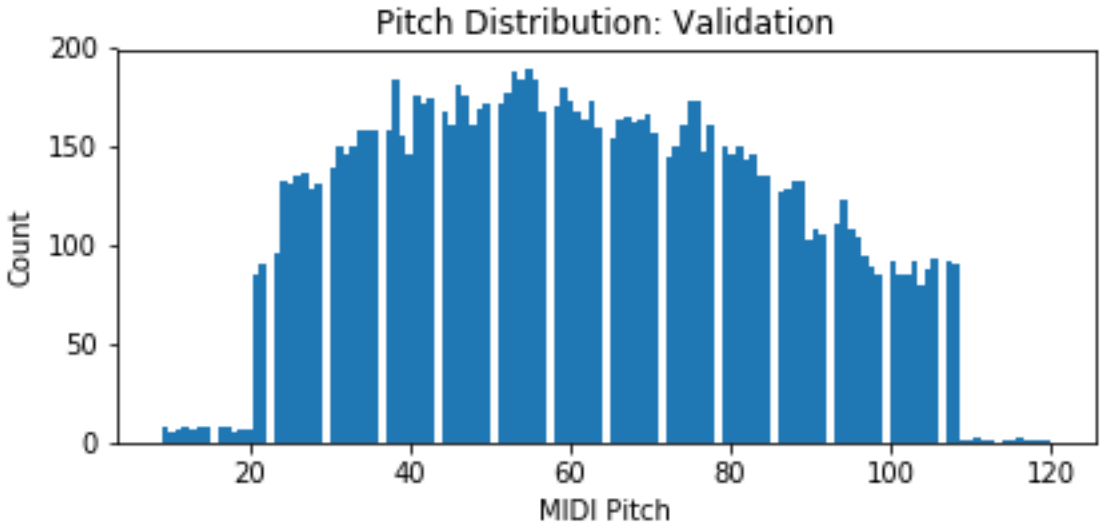


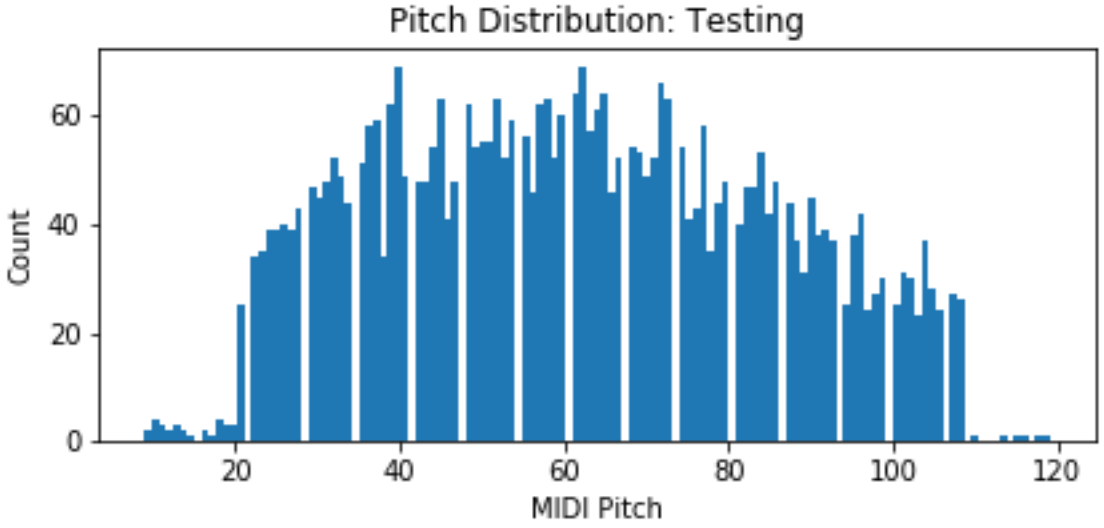

Instrument Velocity: Testing
800
600
400
200
0
127
75
50
25
100
MIDI velocity


Instrument Velocity: Validation
2500
2000
1500
1000
500
0
100
50
25
75
127
MIDI velocity

We also explore the distribution of the MIDI pitch of each dataset in the histograms below. Overall the pitch is normally distributed, albeit there some noticeable gaps for several pitches. A quick exploration using the sets module reveals that all pitches in the validation and test sets are subsets of the training dataset. Thus there are no pitches that are not represented in the training data relative to the two others. Spectulating on the missing pitches, this might due to the fact that some instruments can't replicate all the MIDI pitches.







2.3 Algorithms and Techniques

The labels of the features are already known, therefore we are trying to solve a supervised learning problem.

The first algorithm to use is Gaussian Naïve Bayes (NB); it is ideal for large datasets and high-dimensional data. NB makes the assumption of conditional independence between every pair of features given the value of the class variable. NB makes a Gaussian distribution for the data, if we discover later on that the distribution is binomial then we introduce a Multinomial Naïve Bayes (MNB) classifier. Comparing the two, MNB takes into account the average value of each feature for each class, while NB stores the average value as well as the standard deviation of each feature for each class. For this exercise we will stick to Gaussian NB.

We also will use decision trees classifiers. To avoid over fitting we might require stopping the creation of the tree too early. This is referred to as *pruning* and includes limiting the depth of the tree, limiting the maximum number of leaves, or requiring a minimum number of points in a node to keep splitting it. The advantage of trees is the capacity to visualize the importance of each feature, thus the model can be easily visualized by laypeople. Decision trees are also invariant to scaling of the data, requiring no preprocessing like normalization or standardization.

Building upon decision trees we can use Random Forests (RF), which builds many decision trees and averages the results. A drawback of random forests is that they do not perform well on very high dimensional or sparse data.

Another limitation comes from computing power that might not be handily available.

Another algorithm for supervised classification is Kernalized Support Vector Machines (SVM). SVMs create an expanded feature representation of the data points and compute the scalar products, i.e. distance, between the points. There are two ways to map data to higher-dimensional spaces: the polynomial kernel, which computes all possible polynomials up to a certain degree of the original feature, and the radial basis function (RBF) also known as the Gaussian kernel. SMVs learn how important each of the training data points is to represent the decision boundaries between classes. The main SVM parameters are *gamma* and *C*. Gamma determines how far the influence of a single training example reaches, with low values corresponding to a far reach. A low value of gamma means that the decision boundary will vary slowly, yielding a model of low complexity. The C parameter is a regularization parameter and limits the importance of each point. A small value of C means a very restricted model, where ach data point can only have very limited influence. SVMs are sensitive to magnitudes and require rescaling each feature so that they are all approximately on the same scale. In addition SVMs usually do not scale well to large datasets (10,000+), which might pose a problem later on.

For the classification tasks above, we will be using the scikit-learn library, as it allows us to instantiate the classifiers and adjust the parameters as needed.

We finally introduce a convolutional neural network (CNN) as a classification method. Previous sections have shown that a an instrument can be represented as a spectrogram, visually we can distinguish between instruments. CNNs are capable of reaching high levels of accuracy on training data but can suffer from over fitting. Thus we will introduce drop out at several layers to prevent over fitting. Furthermore, input size will influence training time and accuracy. We might opt for more compact representations of spectrograms to save up on memory and computation demands. In addition we introduce max pooling as a means of down sampling an input representation, reducing dimensionality and allowing for assumptions to be made about features contained in the binned sub-regions. CNN performance is also dependent on learning rate, batch size, number of epochs, and the optimizer. For the latter we will use Adam [12], which has been shown to be computationally efficient, hast little memory requirements, is invariant to diagonal rescaling of the gradients, and is well suited for problems that are large in terms of data and/or parameters.

To set up a CNN, we will use Keras, a wrapper for TensorFlow.

2.4 Benchmark

Consider the case of a fair coin. If we make random tosses we will eventually get as many heads as tails. For our case with eleven classes, randomly guessing should yield 1/11≈9%. If any algorithm can exceed the 10% accuracy threshold then it is not randomly guessing anymore. Since there is no setback for false predictions, the best model will be the one that scores highest in terms of accuracy.

3 Methodology

3.1 Data Preprocessing

The main issue with the data is its shear size. Extracting audio features from a total of 300,000+ wave files takes an absurd amount of the time. We therefore make the following assumption: Taking a subset of the data is representative of its characteristics. For the training data we opt to take 5000 samples from each instrument family label, leaving us with 55,000. However note that *synth_lead*, label 9, is present in neither validation nor testing sets. Would it make sense to train for a label that we will not see? Or do we keep it, and see if it pops up as a false positive?

Given the objective of the project and to reduce computation, we decide to drop *synth_lead* from the training dataset.

Another consideration is the output shape of some features. The sound waveform is represented as a 2D numpy array. All wave files are of the same length and have the same sampling rate, and therefore our application does not account for sounds of different lengths.

To determine if a sound is harmonic or percussive, we compare the average amplitude of percussive and harmonic waveforms. Based on the plots in the data exploration section, the average amplitude of the instrument type should have a higher value.

Other features we seek to extract are:

- Mel scaled spectrogram
- Mel frequency cepstral coefficients
- Chroma energy
- Spectral contrast

Given an a sound of size S of size n, the transformation for feature extraction f is defined as:

$$f: \mathbb{R}^n \rightarrow \mathbb{R}^{m \times k}$$

The output shape of the list above is an mx*k* array, where the shape size is dependent on user inputs. *k* is dependent on the duration of the sound, while *m* is dependent on the transformation parameters such as hop length and number of mels.

For the supervised learning algorithms we will need to make the features time independent by averaging each output over the rows. We then obtain a stacked representation of these features as input for our algorithms.

We use the JSON file to randomly extract 5000 of each sample and then drop synth lead. The data wrangling code can be found here:

https://github.com/NadimKawwa/NSynth/blob/master/Data_Wrangling.ipynb

For the CNN, we opt to use exclusively the mel spectrogram representation of a wave file, believing it to be enough training data. We also wish to use the smallest 2D representation possible to save on training time and computing costs (AWS is not free).

3.2 Implementation

The supervised learning implementation can be found here:

https://github.com/NadimKawwa/NSynth/blob/master/SupervisedLearning.ipynb

For Gaussian Naïve Bayes, the implementation is straightforward: Set the targets on one side and dump all the features in the training set. We use scikit-learn's GaussianNB class to predict instrument label.

The next algorithm is RandomForestClassifer, and again there is no need to preprocess the data. We need to set up parameters such as the number of estimators *n_estimators*, the maximum depth of the tree *max_depth*, and set *warm_start* to true so that the estimator reuses the solution of the previous call to fit and add more estimators to the ensemble.

For the last supervised algorithm, we use SVMs to predict the labels. For this algorithm it is highly advised to transform our data via scaling or normalization. For this exercise we scale each feature on a Min Max scale in the range [0,1] by setting up the *feature_range*. In addition SVM performance is dependent on the penalty parameter *C*, the choice of kernel, and the kernel coefficient *gamma.* We set C=0.1 to decrease the effect of noisy data and keep gamma to its default value. There are other parameters that we can tune however they play a lesser role.

For the CNN classifier we first convert the class vector to binary class matrices. For this exercise we limit ourselves to spectrograms. To reduce training time we decrease the number of mels to 100 and increase hop length to 1024. The CNN we use rather simple and is a repetition of the following sequence:

- 2D Convolution
- Relu activation
- 2D Convolution
- 2D Max pooling
- Dropout

When setting out the learning rate for the optimizer we start with 10^{-4}, slightly lower than the default value. The latter parameter tells the optimizer how far to move the weights in the direction opposite of the gradient for a given batch. A high training rate can cause the training to diverge because of unreasonably high weight changes. Too low however and optimization will take too much time because we are making infinitesimal changes to the weights.

Given that we have a multiclass problem, the best loss function appears to be categorical cross entropy. Finally the number of epochs can't really be determined beforehand. To address this issue, we will compare training and validation sets in two plots:

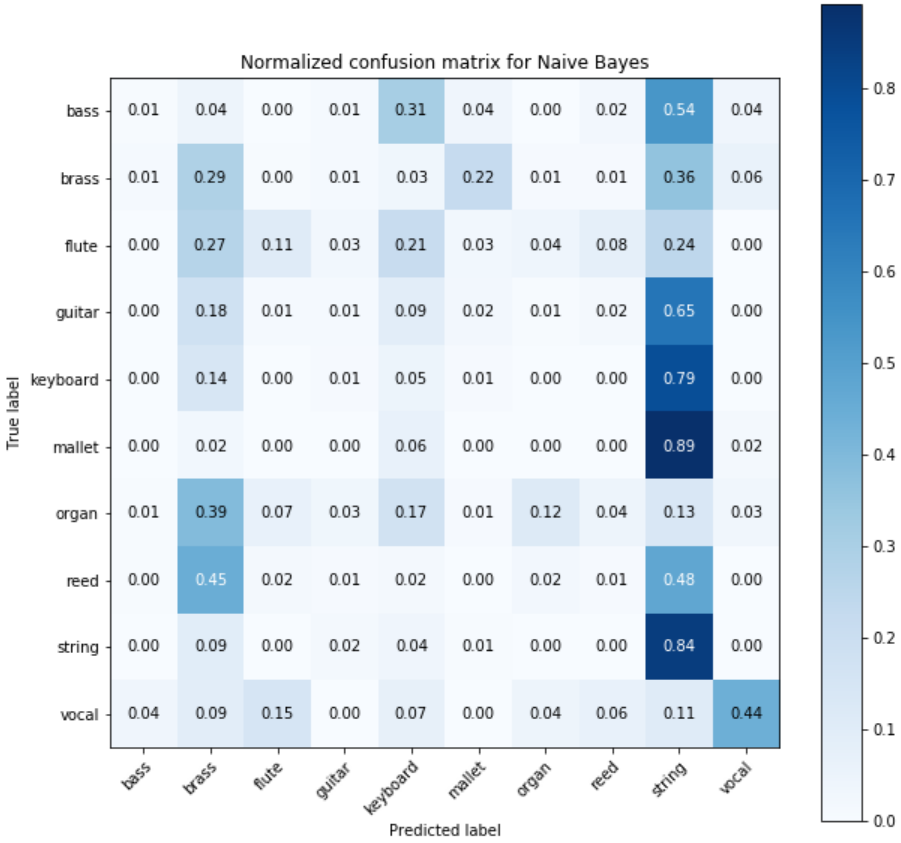
- Accuracy as a function of epoch
- Loss as a function of epoch

The implementation of the CNN can be found here:

https://github.com/NadimKawwa/NSynth/blob/master/CNN_Spectro.ipynb

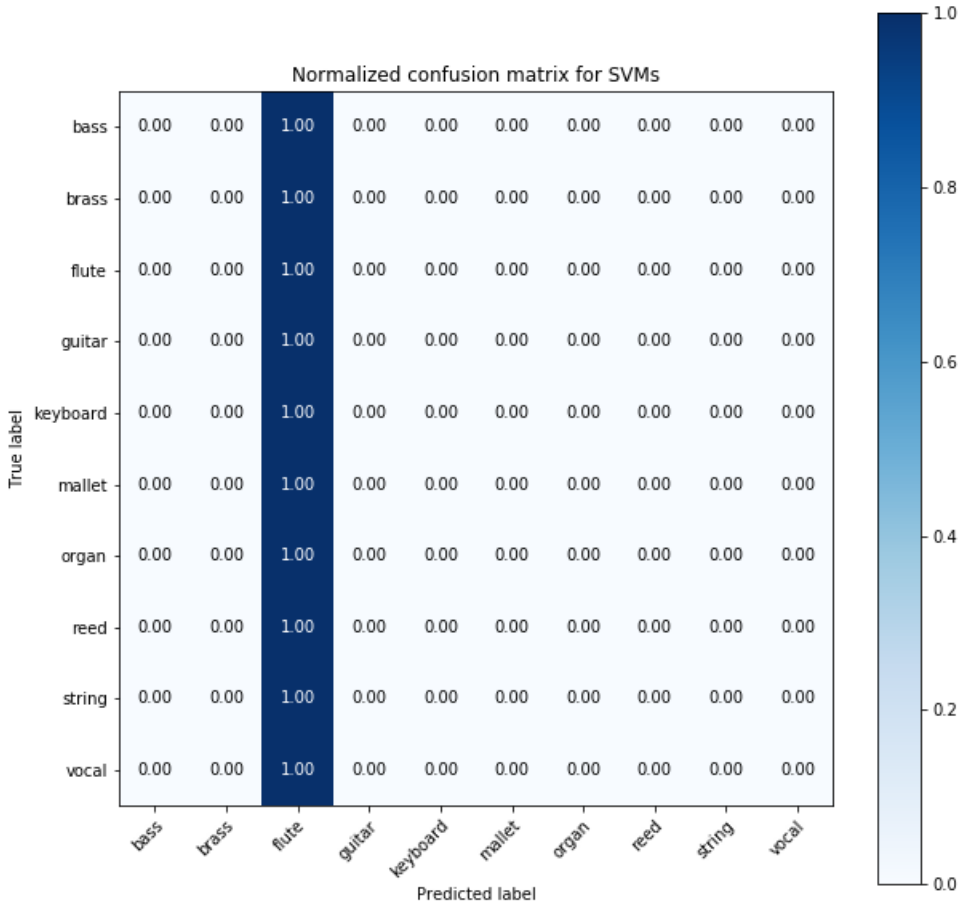
3.3 Refinement

For Naïve Bayes no refinement was necessary. However must discard the model because the accuracy is very low, in the range of 11%. The figure below shows the confusion matrix for Naïve Bayes.



NB is slightly better then a coin toss. The reason why it performs so poorly on the dataset is because it is founded on conditional probability: the features are independent of each other. However we can ascertain from the data that each feature is highly dependent on a series of others.

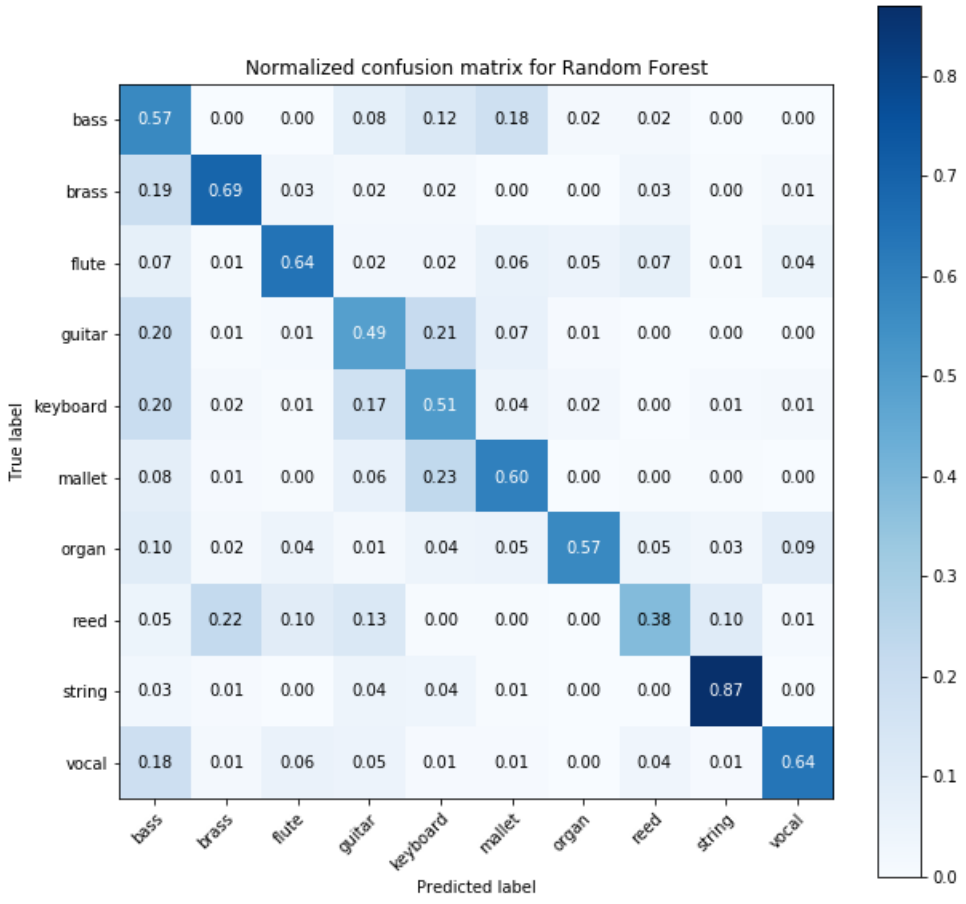
In addition, SVMs perform at an accuracy of 4.39%, predicting all instruments as flutes. This indicates hat SVMs are not suitable for the way we have set up the feature space.



At this point it is necessary to explain the concept of time complexity, known as big *O(n)*, given the size of the data. SVMs require a significant time to fit the data, the quadratic programming (QP) solver's complexity is dependent on the data and varies between:

$$[n_{features} \times n_{samples}^2, n_{features} \times n_{samples}^3]$$

Consider the training data which has 166 features and 50,000 samples, and the time complexity becomes too large. Compare that to random forests where the time complexity is $O(n_{features} \times n_{samples} log(n_{samples}))$ and the difference is striking. We can project from early on that random forests will likely be a better solution. With RF the accuracy is 57.37%, the figure below shows the confusion matrix.



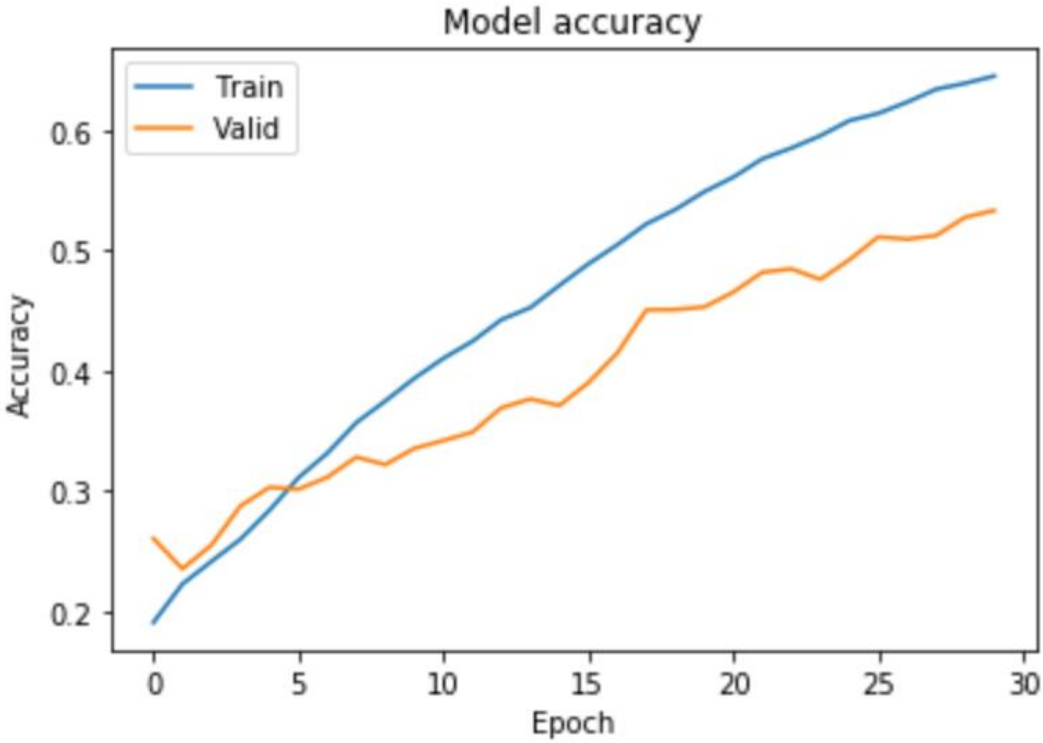
To optimize random forests we explore some hyperparameters:

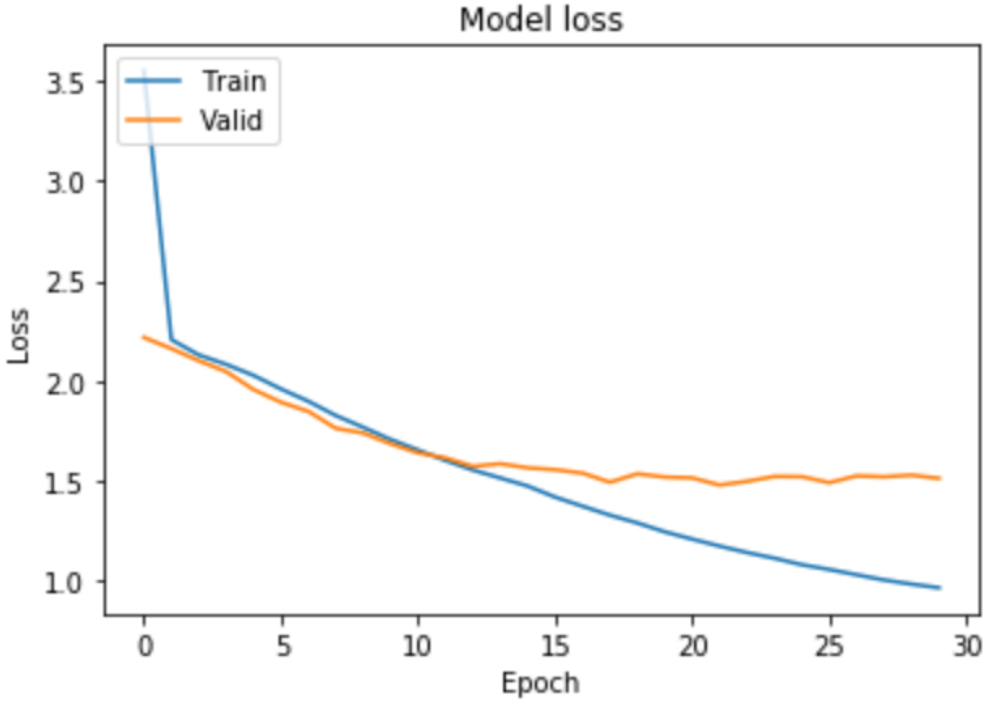
- *n_estimators*
- *max_depth*
- *max_features*
- *min_samples_split*
- *bootstrap*
- *criterion*

We can compare a randomized search or a grid search. The latter is simply a less thorough grid search where not all parameter combinations are tested out. The documentation states that computation time for randomized search is drastically lower and performance is slightly worse. We decide to

implement RandomizedSearchCV and test accuracy improves to 62.23%, the confusion matrix below bears witness to this improvement:

With CNNs on the spectrogram we obtain a test accuracy of 54.5% at the cost of 12hours of training. The plots below show the difference in accuracy and loss for training and validation datasets. We notice a discrepancy in accuracy between the training and validation datasets throughout the training. In addition, the loss diverges after epoch 15 and we experience and diminishing return from the validation set.


Model accuracy
Train
Valid
Accuracy
Epoch
0.2
0.3
0.4
0.5
0.6
0
5
10
15
20
25
30


Model loss
Train
Valid
Loss
Epoch
1.0
1.5
2.0
2.5
3.0
3.5
0
5
10
15
20
25
30

At this point we can explore using transfer learning to optimize CNN results. However that does not seem feasible as the trained kernels are not optimized for pictures of spectrograms. Take for example Google's imagenet, it is trained on pictures of animals, plants, and natural objects but not spectrograms or even wave plots.

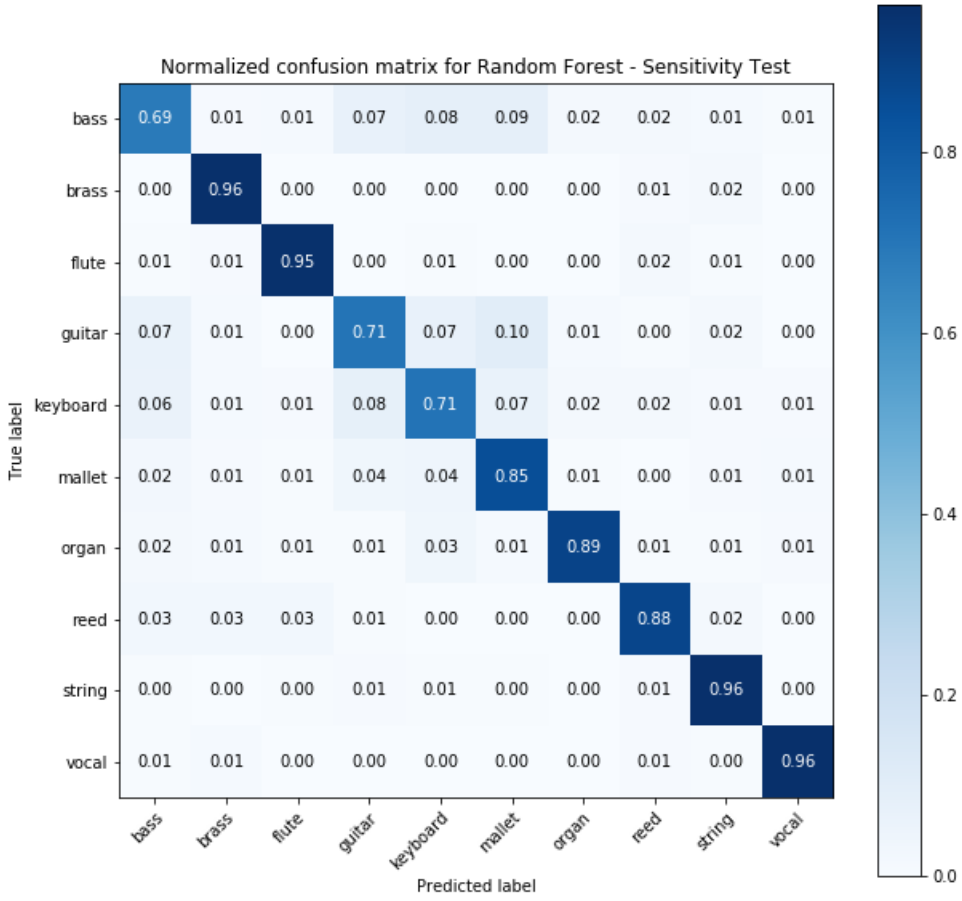
4 Results

4.1 Model Evaluation and Validation

The ideal model appears to be the random forest given that it performs a lot better than the others in terms of accuracy. The best estimator is described *as:*

RandomForestClassifier(bootstrap=False, class_weight=None, criterion='gini', max_depth=20, max_features=9, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=3, min_weight_fraction_leaf=0.0, n_estimators=80, n_jobs=1,oob_score=False, random_state=None, verbose=0, warm_start=False)

To test the sensitivity of our model we can use unused training data and see how it performs on it. This serves as an external validation mechanism since it counts as data that our model has not seen yet. We find out that our accuracy is a whopping 85.67% and the confusion matrix is shown below:



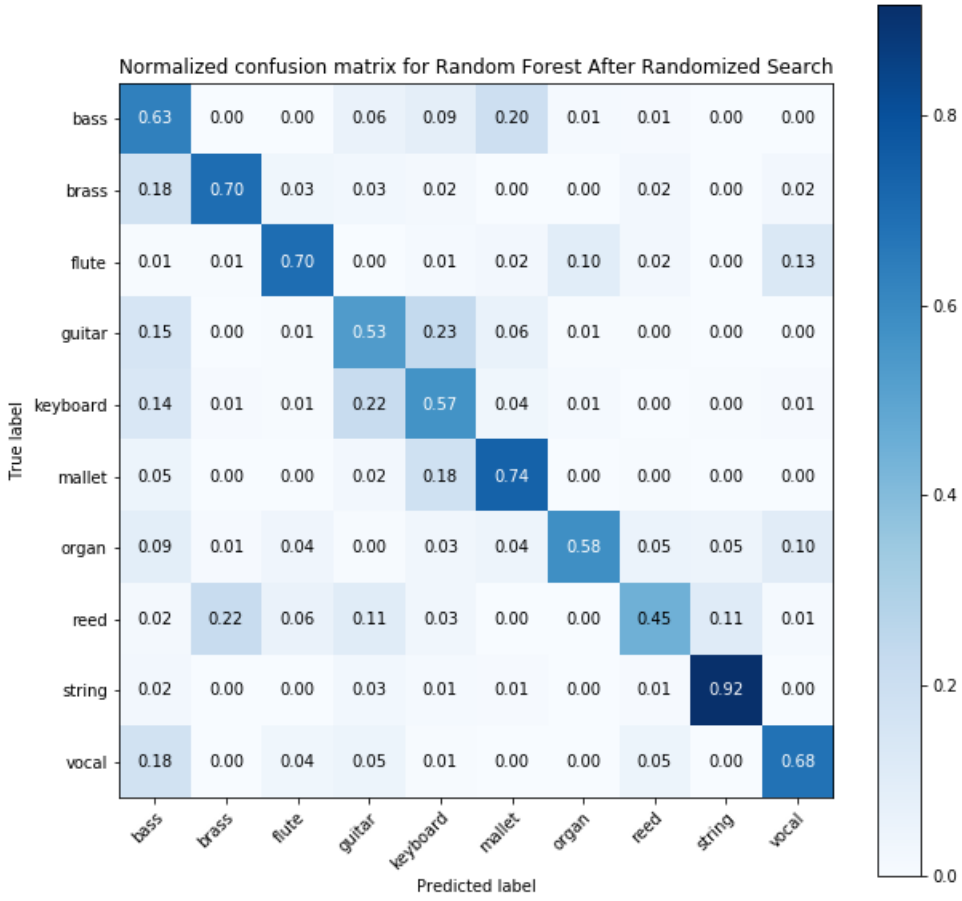
The results are starkly different from the base case. This could be due to the fact that features in the training set are more or less very similar to each other. Another indirect sensitivity test is the CNN where we opted for a smaller feature space and obtained a smaller accuracy. This shows that our model can deliver high levels of performance but is nowhere near ideal.

The sensitivity test can be found here:

https://github.com/NadimKawwa/NSynth/blob/master/Sensitivity_test.ipynb

4.2 Justification

Our final model is the random forest, with the confusion matrix shown below:



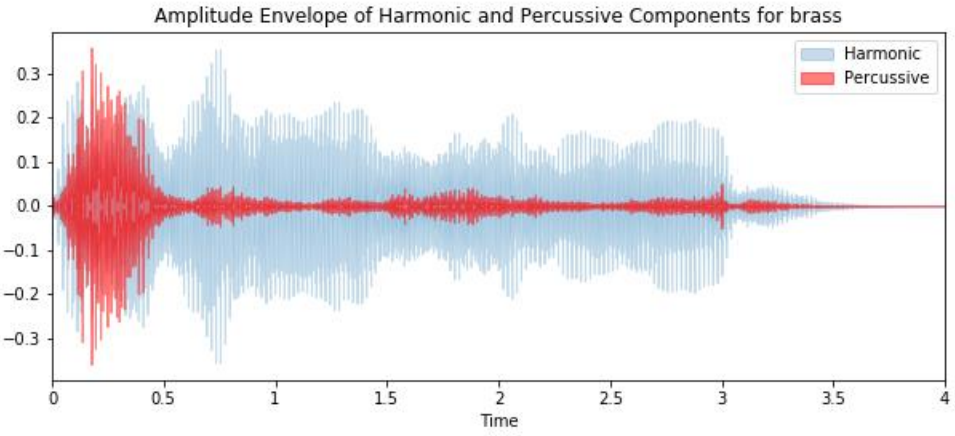
We can see that the model performs significantly better on classifying some features compared to others, for example string versus guitar. Our main benchmark is accuracy and the reported accuracy is upwards of 60% thus passing our benchmark. However it is revealed that the classifier is very sensitive to changes in the training data, given that it is all drawn from the same pool. In addition the optimal parameters are obtained by a random grid search, given that a massive dataset coupled with limited computation power dictates a judicious balance between exploration/exploitation. There definitely exists a better combination of parameters out there for random forests, or a better way to set up at a CNN.

However given that we are trying to deliver results better than random guessing, this solution defines a path forward for improving on current results.

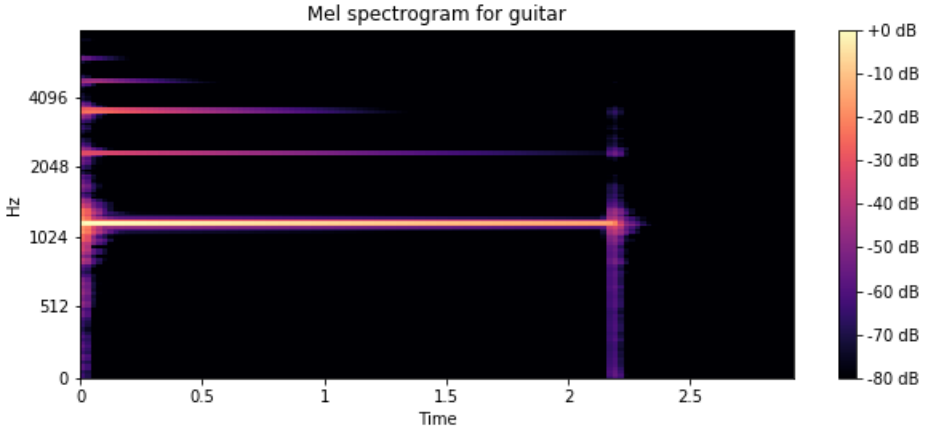
5 Conclusion

5.1 Free-Form Visualization

Going back to our original hypothesis, every sound can be represented as a wave plot as show in the figure below.



We used this representation to engineer a features, one them called the mel scaled spectrogram, shown in the plot below.



The objective was to classify the sound instrument given a massive dataset and limited computation power. Thinking ahead, there might be more features or better setups of algorithms that can deliver better results.

However the fundamental representation of a sound is a wave and how we decide to model it. Based on data exploration, it appeared that the spectrogram provided the most clear-cut way to distinguish instruments with the naked eye.

5.2 Reflection

The first step of the project was to explore the features presented to us and shed out those that would not aid in classification. Keeping the end goal in mind is essential in any project. Although I am familiar with Fourier Transforms and representations of frequency, I had to learn how music is read at a machine level. Indeed early on in the project, the sampling rate proved to be a critical feature: Is a lower sampling rate better? Do we stick to the original sampling rate?

In addition, Librosa offers a whole range of feature extraction tools and not all of them were used. Using unnecessary features might bog down the analysis and lead us away from a lean solution.

What surprised me most was random forests beating CNNs. Given that neural nets are very popular in machine learning applications, they must be set up in a proper manner. Moreover SVMs in the way they were set up turned out to be very poor performers, and required long periods of time to fit the data.

The final solution still requires improvements and is far from perfect, given that I had initial hopes of achieving upwards of 90% accuracy with a CNN model.

5.3 Improvement

The model can be improved by changing the feature representation of the wave files. For examples we could opt for a higher sampling rate, increase the number of mels. The spectral rolloff frequency might be useful feature for classification. Moreover some overlooked features such as Tonnetz centroids might improve our model. In addition we could train the CNN on the chroma energy or a log power spectrogram without mel scales.

We can use a multi layer perceptron (MLP) on the features and see how it performs compared to the CNN in terms of accuracy and efficiency. Moreover we decided to overlook K nearest neighbors since it performs poorly with large datasets. We do not know how representative our subset of the training data is, and if that skewed our results one way or another. Rainbowgrams are also a visualization tool used by one of the NSynth authors and is a potentially a richer feature representation that our current model.

6 References:

[1] Jesse Engel, Cinjon Resnick, Adam Roberts, Sander Dieleman, Douglas Eck, Karen Simonyan, and Mohammad Norouzi. "Neural Audio Synthesis of Musical Notes with WaveNet Autoencoders." 2017.

[2] Learning Multiple Layers of Features from Tiny Images, Alex Krizhevsky, 2009.

[3] Introduction to machine learning with Python: a guide for data scientists

Andreas Müller-Sarah Guido - O'Reilly. – 2017

[4] https://hypertextbook.com/facts/2003/ChrisDAmbrose.shtml

[5] http://www.electropedia.org/iev/iev.nsf/display?openform&ievref=801-30-04

[6] https://ccrma.stanford.edu/CCRMA/Courses/152/percussion.html

[7] Fitzgerald, Derry. "Harmonic/percussive separation using median filtering." 13th International Conference on Digital Audio Effects (DAFX10), Graz, Austria, 2010.

[8] Meinard Müller and Sebastian Ewert "Chroma Toolbox: MATLAB implementations for extracting variants of chroma-based audio features" In Proceedings of the International Conference on Music Information Retrieval (ISMIR), 2011.

[9] https://ccrma.stanford.edu/~jos/st/Spectrograms.html

[10] Roger B. Dannenberg, Introduction to Computer Music Spectral Centroid

[11] Jiang, Dan-Ning, Lie Lu, Hong-Jiang Zhang, Jian-Hua Tao, and Lian-Hong Cai. "Music type classification by spectral contrast feature." In Multimedia and Expo, 2002. ICME'02. Proceedings. 2002 IEEE International Conference on, vol. 1, pp. 113-116. IEEE, 2002.

[12] arXiv:1412.6980 [cs.LG]