Capstone Project
Machine Learning Engineer Nanodegree

Jordan Chong
May 20th 2019

I. Definition

(approx. 1-2 pages)

Project Overview

Much of the attention that is giving to deep learning's achievements has been directed towards it's ability to make sense of a complicated world using visual data such as images and video. However cameras are expensive and have a limited field of view. Audio data on the other hand is cheaper to collect and comes with more freedom in terms of positioning. It is said that a team of world class machine learning engineers with mediocre data sets can often be outperformed by mediocre machine learning engineers with stellar datasets. For this reason it appears to be important that we make certain we are making effective use of one of our cheapest and readily collectible sources of data, audio.

Part of the reason is due to the fact that audio data, particularly of the non-speech variety, is slightly more difficult to label and more susceptible to noise. And due to the wide range of sounds we can expereince and the variety of sources that create similar sounding noises we do not currently have a reliable automatic general- purpose audio tagging system. With the datasets provided by Freesound and Google's Audioset, much of the manual labor of labeling audio clips has been done and it is predicted that we may have success in developing multi-label classifiers built in the same vein as those for imagenet.

Problem Statement

The goal of this project will be to develop an algorithm to tag audio data automatically using a diverse vocabulary of 80 categories. The hope will be to design the algorithm in such a way that it might become useful for applications such as automatic labelling of sound collections to the development of systems that automatically tag video content or recognize sound events happening in real time. The model should be able to generalize to input data that may contain additional noise not specific to the particular label.

To begin, the project will entail doing some initial data evaluation, possibly some wrangling, and analysis. I will examine properties of the audio to judge potential for further feature enhancements or for clues in shaping the neural net architecture. Then benchmark models will be built using historical freesound competition winning algorithms and ensemble methods to set a baseline measure for accuracy. From there I will explore optimizations that can be made to the benchmark models or experiment with entirely new neural net architectures in order to see if accuracy can be significantly improved from the benchmark model. These architectures will likely employ some sort of temporal convolutional neural net.

Metrics

Accuracy alone should prove to be a sufficient evaluation metric for this classification task as this is how the kaggle competition will be judged regardless. The model will be evaluated on the 1.6k sample manually labeled test dataset. The accuracy will be determined by how many of these samples can be accurately predicted based only on the wave audio input.

II. Analysis
(approx. 2-4 pages)

Data Exploration

The Music Technology Group from the University of Pompeu Fra in Barcelona (MTG-UPF (https://www.upf.edu/web/mtg)) can be thanked for having compiled the Freesound Dataset FSD (https://annotator.freesound.org/fsd/) which was based on Freesound content organized with the Google's AudioSet Ontology. It contains everyday sounds such as musical instruments, human sounds, domestic sounds, and animals. This data was labeled manually by humans following a data labeling process using the Freesound Annotator platform.

This data will be supplemented by additional "noisy" tracks taken from a pool of Flickr videos taken form the Yahoo Flickr Creative Commons 100M dataset (YFCC) (http://code.flickr.net/2014/10/15/the-ins-and-outs-of-the-yahoo-flickr-100-million-creative-commons-dataset/). This data was labeled using automated heuristics applied to the audio content and metadata of the original Flickr video clips. For this reason the YFCC data should be expected to be much noisier.

The audio data is labeled using a vocabulary of 80 labels from Google’s AudioSet Ontology (https://research.google.com/audioset///////ontology/index.html). It includes various sounds such as Guitar and other Musical instruments, Percussion, Water, Digestive, Respiratory sounds, Human voice, Human locomotion, Hands, Human group actions, Insect, Domestic animals, Glass, Liquid, Motor vehicle (road), Mechanisms, Doors, and a variety of Domestic sounds.

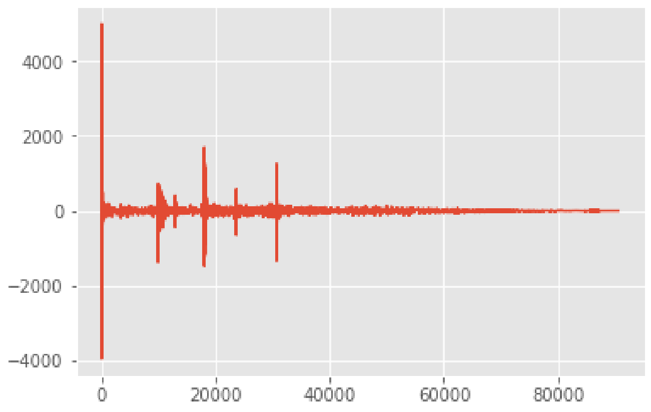
The training data is split into 2 subsets. FSD's curated subset and YFCC's noisy subset. There are about 10.5 hours worth of audio from the curated set and 80+ hours worth of data from the noisy set. For both sets the average number of labels per clip is 1.2. Clips can range from 0.3 seconds to 30 seconds in .wav format.

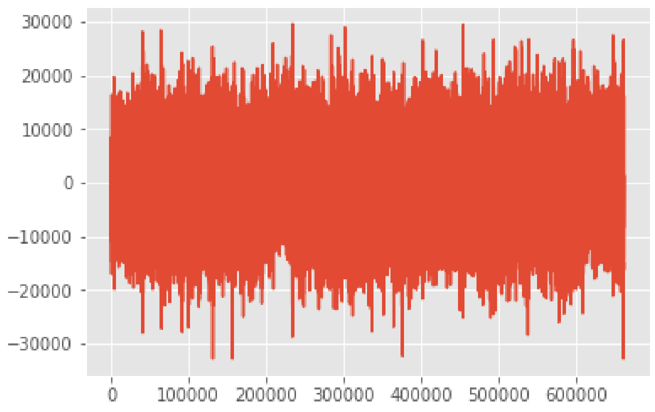
The testing data comes entirely from the curated FSD dataset.

Exploratory Visualization

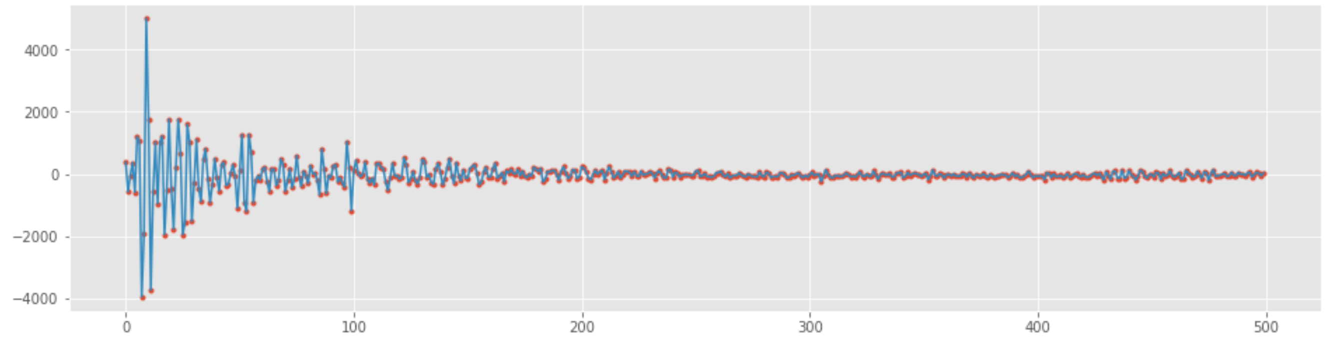
The audio data is represented digitally using PCM or pulse code modulation. They have a bit depth of 16 which means that the amplitude of each sample is a range of 2^16 (=65,536) possible values. And a sample rate of 44.1kHz which means that each second of audio is represented by 44,100 samples.

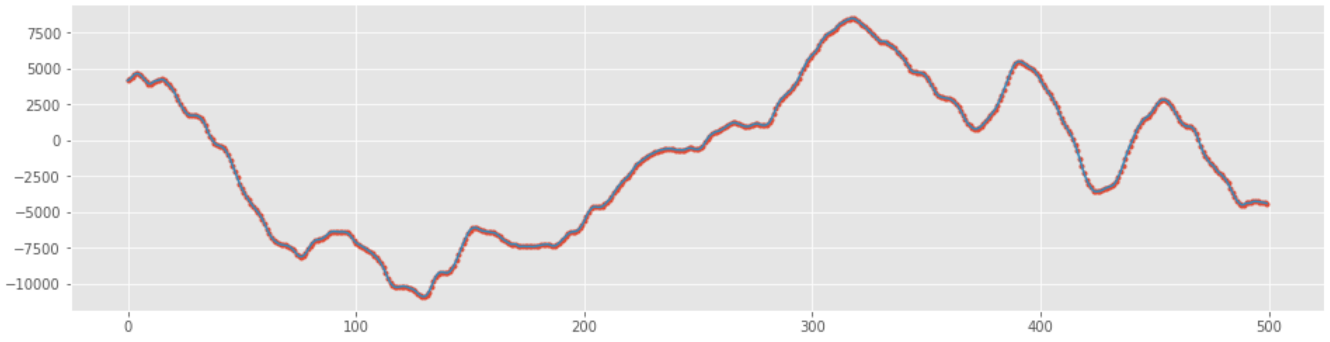
Thus each audio clip is represented as a 1D array which if plotted as a line chart will look like these two samples below.





When zoomed up to the first 500 samples taken of the audio it is clear that there are some gaps between the samples that are taken.





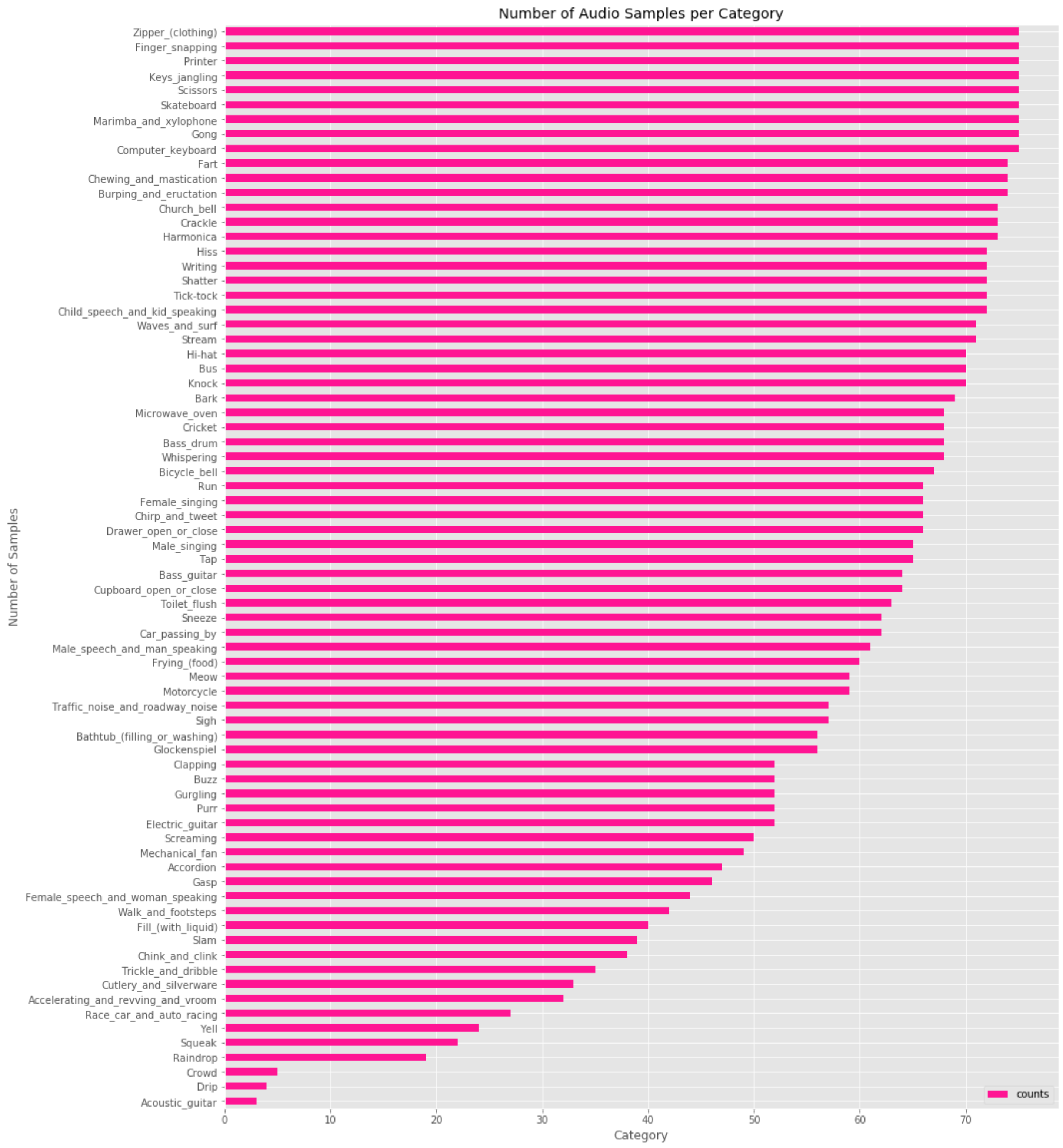
Exploratory Analysis:

In order to simplify the construction of the models, multi-labeled audio samples were ignored. This resulted in the loss of roughly 15% of all audio samples but a reduction of several thousands of independent combinations of multiple labels. Since the test data sets only contain 80 single labeled samples and the vast majority of data in also single labeled, the multi-labeled samples will be removed to allow for emphasis to be placed on building a model that is effective on those samples first.

Curated Data Sample Distribution:

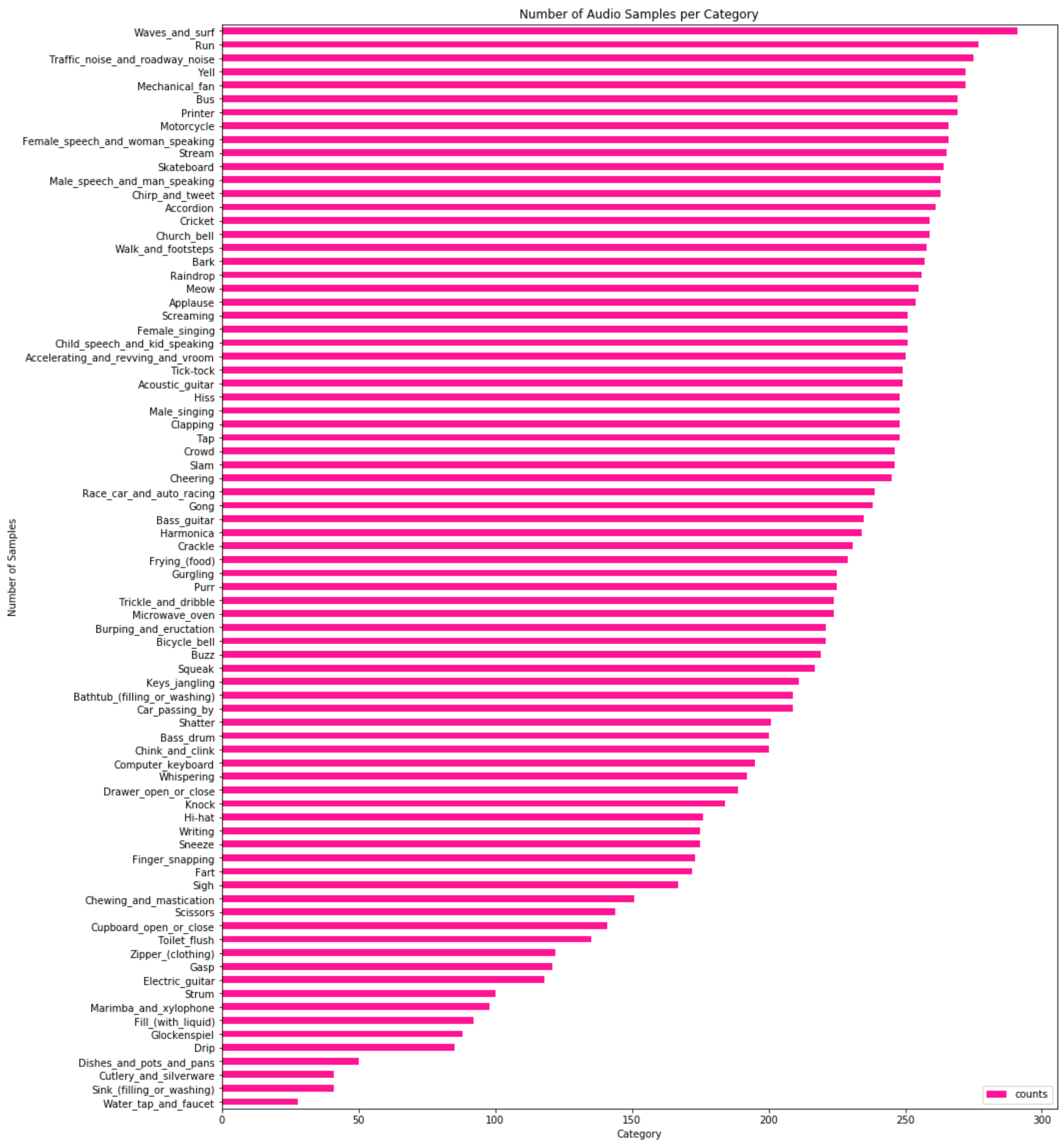
- Below is a histogram of samples for category present in the curated dataset. The set included a total of 4269 after the removal of 14.1% of the original data which included multi-labeled samples.
- The distribution is not even across all labeled samples but the majority of categories have at least 40 samples to support them.
- The least represented category has 3 samples to it and the most represented category has 75

Noisy

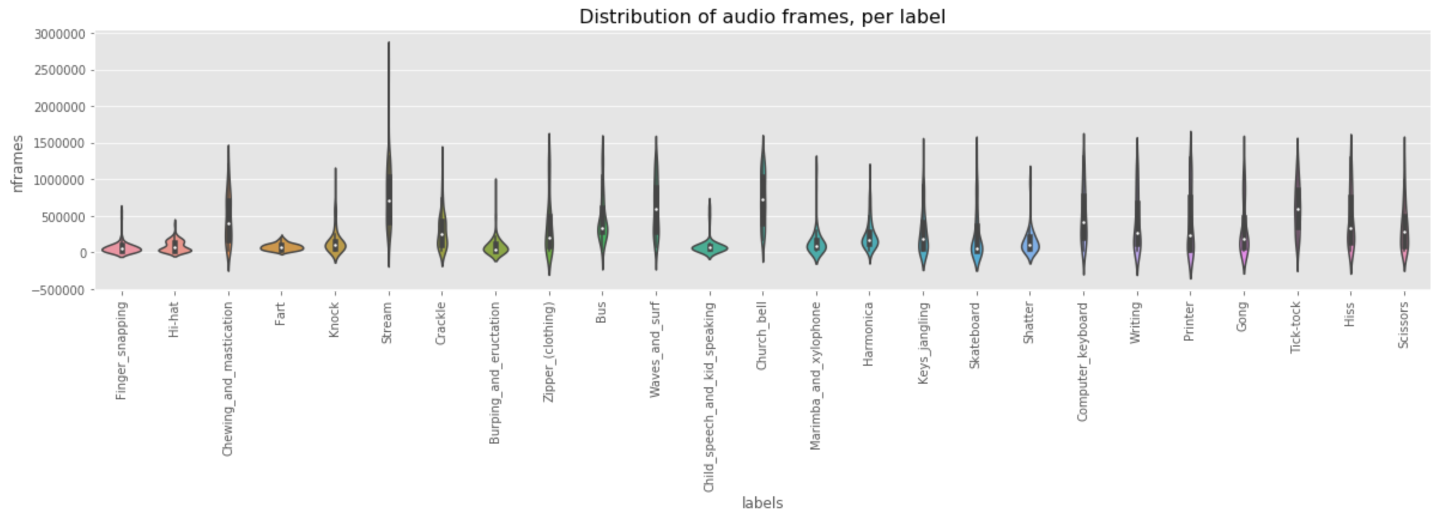


Noisy Data Sample Distribution:

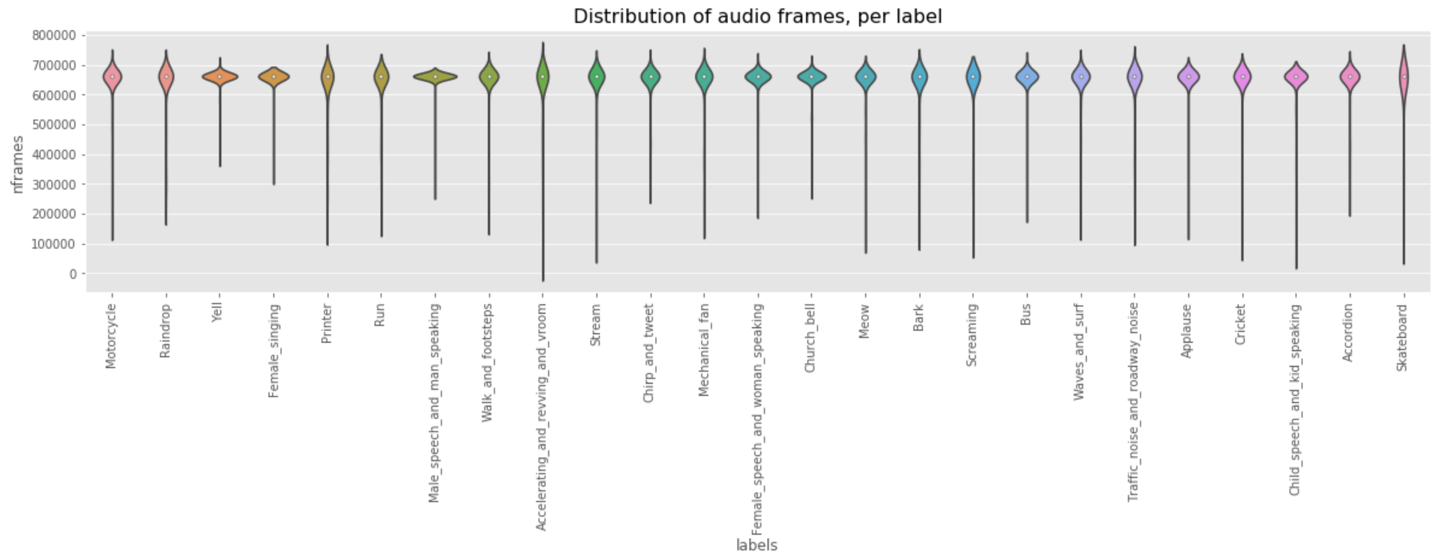
- Below is a histogram of samples for category present in the noisy dataset. The set included a total of 16,566 samples after the removal of 16.4% of the original data which included multi-labeled samples.
- The distribution is not even across all labeled samples but the majority of categories have at least 150 samples to support them.
- The least represented category has 28 samples to it and the most represented category has 291



The number of categories is large, so let's check the frame distributions of top 25 categories in the curated set. Notice that for many of the labels finger snapping and farts, the number of frames is very small with little variability. Streams and keyboard typing are examples of audio clips that tend to run a lot longer. The chart also suggest the need for at the very least about a 100,000 frames of data to accurately represent each clip. Trimming the data in order to normalize duration will be important for training, however we will need to remember that doing this may create more data loss for certain kinds of labels more than others.

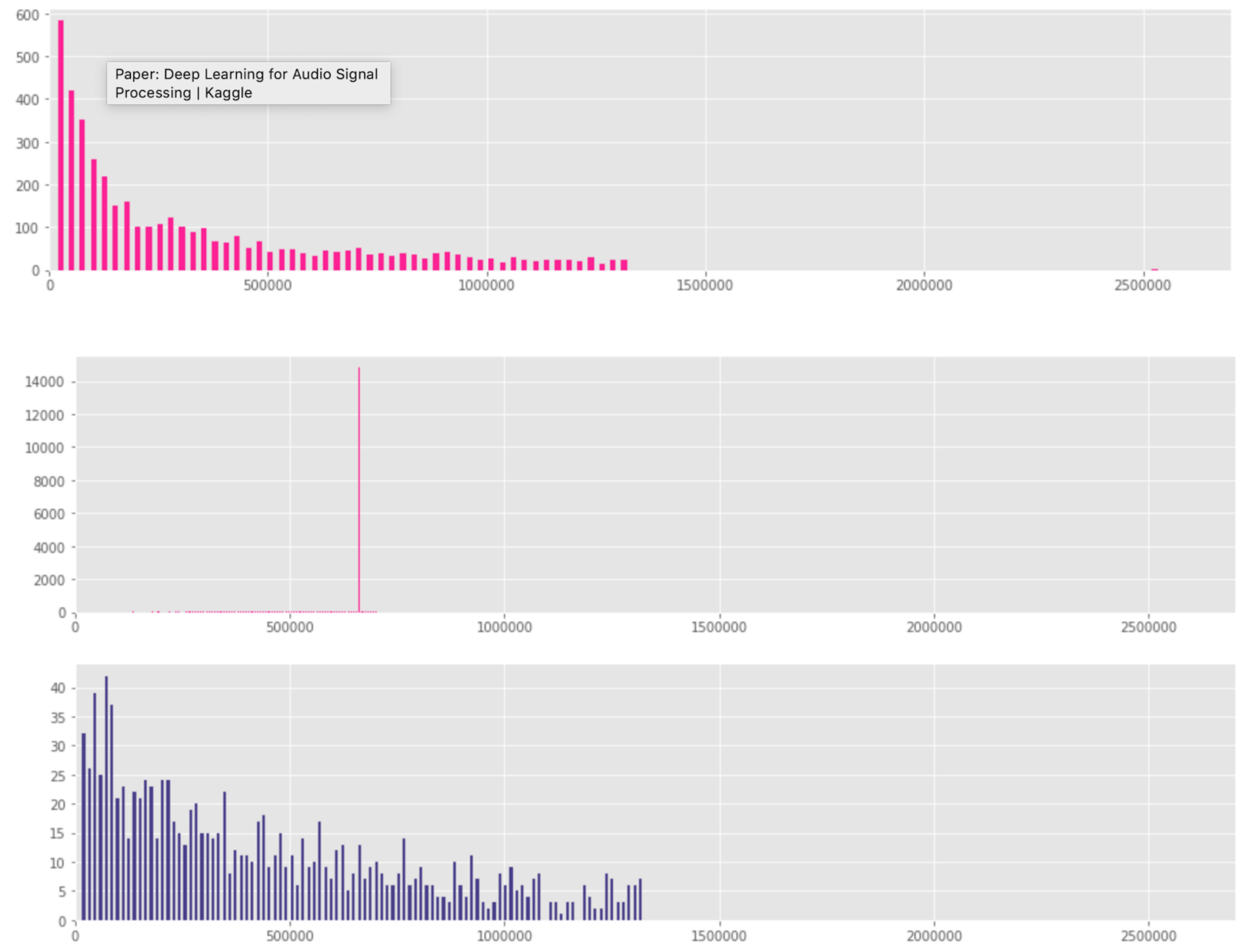


And the same violin chart for the noisy set. On the whole the noisy sets are a more than just a few seconds longer than the curated set (but note scale difference on the y-ticks). By comparison the noisy data set is much more uniform in terms of sample distribution for each label. This is likely due to the automated data retrieval process which was likely programmed to simply extract a default 15 seconds of audio from each video source.



Also comparing the frame length distributions of the two tests sets to the test sets shows that the curated set (pictured first) and the train set share the most similarities in terms of the range of frame counts. The noisy set (pictured second) is awkwardly proportioned since nearly every sample was taken for 15 seconds.

Frame Length Distribution in train and test



Algorithms and Techniques

The algorithm to solve this challenge will be a convolutional neural network. The CNN is a modern approach to classification problems that requires a large amount of data and uses sparsely connected nodes to find patterns in data. It is particularly useful for image classification due to its ability to examine sections of an image and identify patterns indicative of the label. This characteristic will prove useful if the 1D audio array format of the data does not prove as useful as the MFCC and Spectogram (aka sonogram) variations in which case 2D convolutional networks can take advantage of all the same performance enhancing features of image classifiers.

The CNN approach will involve careful selection of certain parameters including:

1. Hidden Layer depth
2. Layer Type (Convolutional 1d/2d, Maxpool, Dense, Dropout
3. Loss function (categorical cross entropy)
4. Learning rate
5. Number of epochs
6. Validation test set size

7. Optimizer (adam)

As well as several preprocessing feature choices:

1. Audio duration
2. Sampling rate
3. Bit depth
4. Input transformation Pulse Code Modulation / Mel Frequency Cepstrum Coefficiencts / Mel-Spectogram

Benchmark

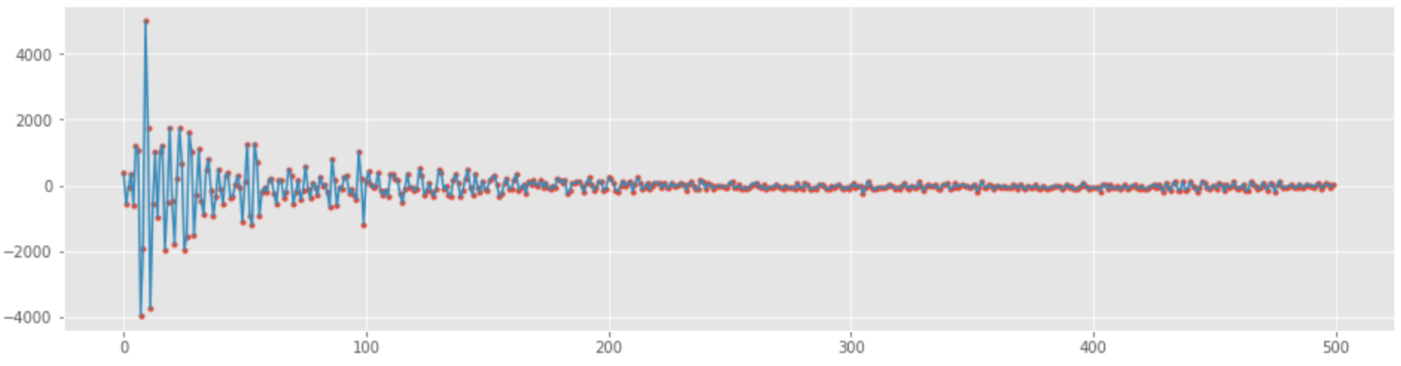
Ensemble methods such as random forest and adaboost have historically done well on multi label classification problems. For this problem I intend to start with an audio .wav based model using random forests as a starting point for a non neural net based benchmark model for comparison. This benchmark will then be tested against multiple different neural net based approaches including another pure audio based neural net, a Mel Frequency Cepstrum Coefficient based neural net, and a Mel-Spectogram based neural net. The optimal model will be chosen by measure of accuracy comparing each of the different approaches.

III. Methodology

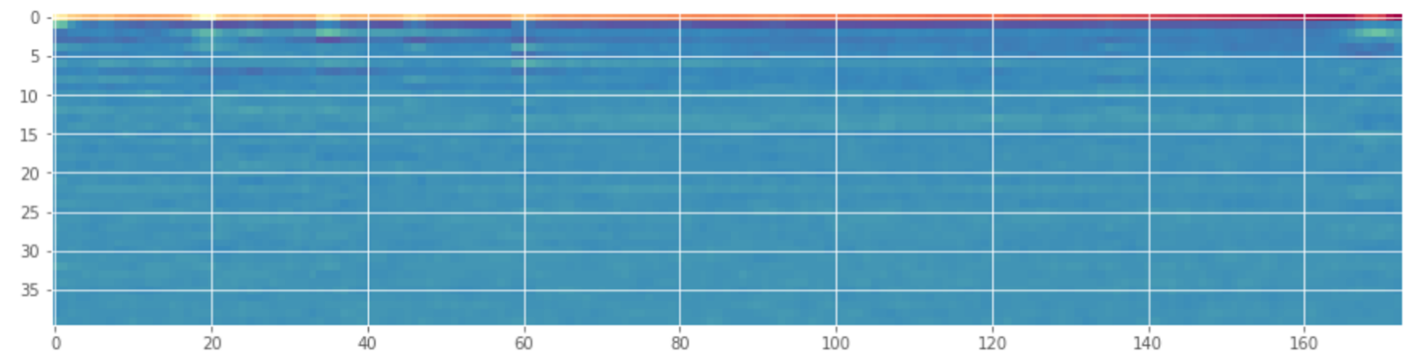
(approx. 3-5 pages)

Data Preprocessing

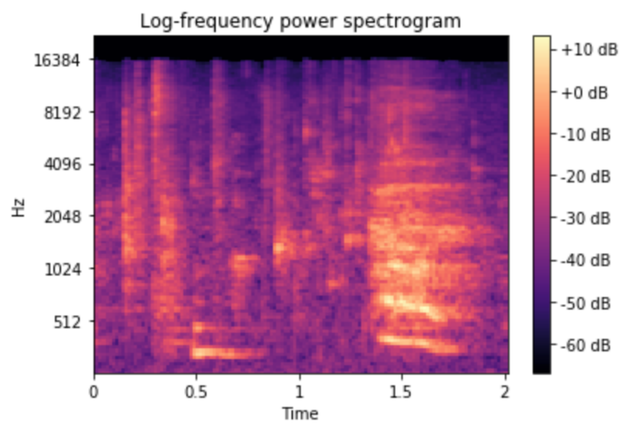
By far one of the most challenging tasks for this challenge was to determine the optimal input format to represent the audio data would be. While the 1D dimensional PCM audio array from the wav file does provide a high degree of fidelity, it does not do so in a way that is very easy for the model to now what is related audio and what is background noise. Therefore MFCC and Spectogram techniques have proven popular for further enhancing the audio data and turning them into images that can take better advantage of modern deep learning techniques such as convolutional neural nets.

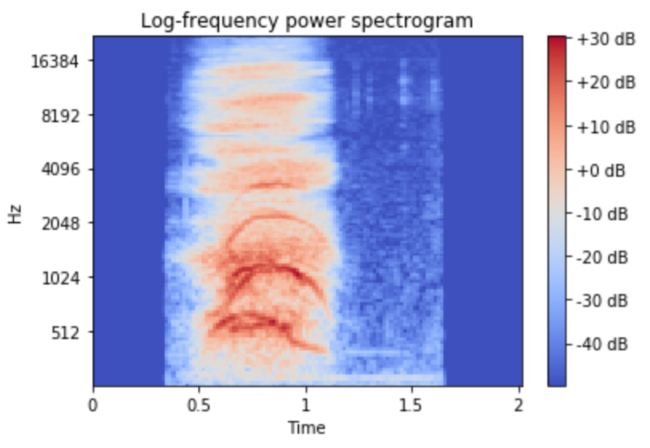


MFCC is a technique popularized in the 80’s before neural nets became popular and have since continued to be used in combination with modern deep learning to gain even greater performance in automatic speech and speaker recognition. It attempts to mathematically mimic the logarithmic perception of loudness and pitch that the human ear is capable of. In addition it has steps to help eliminate speaker dependant characteristics by excluding certain fundamental frequencies and harmonics. The Librosa python library will be used to convert the data to MFCC format.



Mel-Spectogram is another format for converting audio data into a 2D image format for better feature exposure. However rather than trying to mimic specific patterns of human speech and perception is merely converts the audio into a heat map where time is the x-axis, frequency is the y-axis, and the color at each time/frequency coordinate represents the amplitude on a gradient scale. This technique though simpler then MFCC also produces a unique fingerprint of the audio data and because it is not specific to human speech characteristics is could potentially perform better on the non speech related data in the Freesound data sets.





Implementation

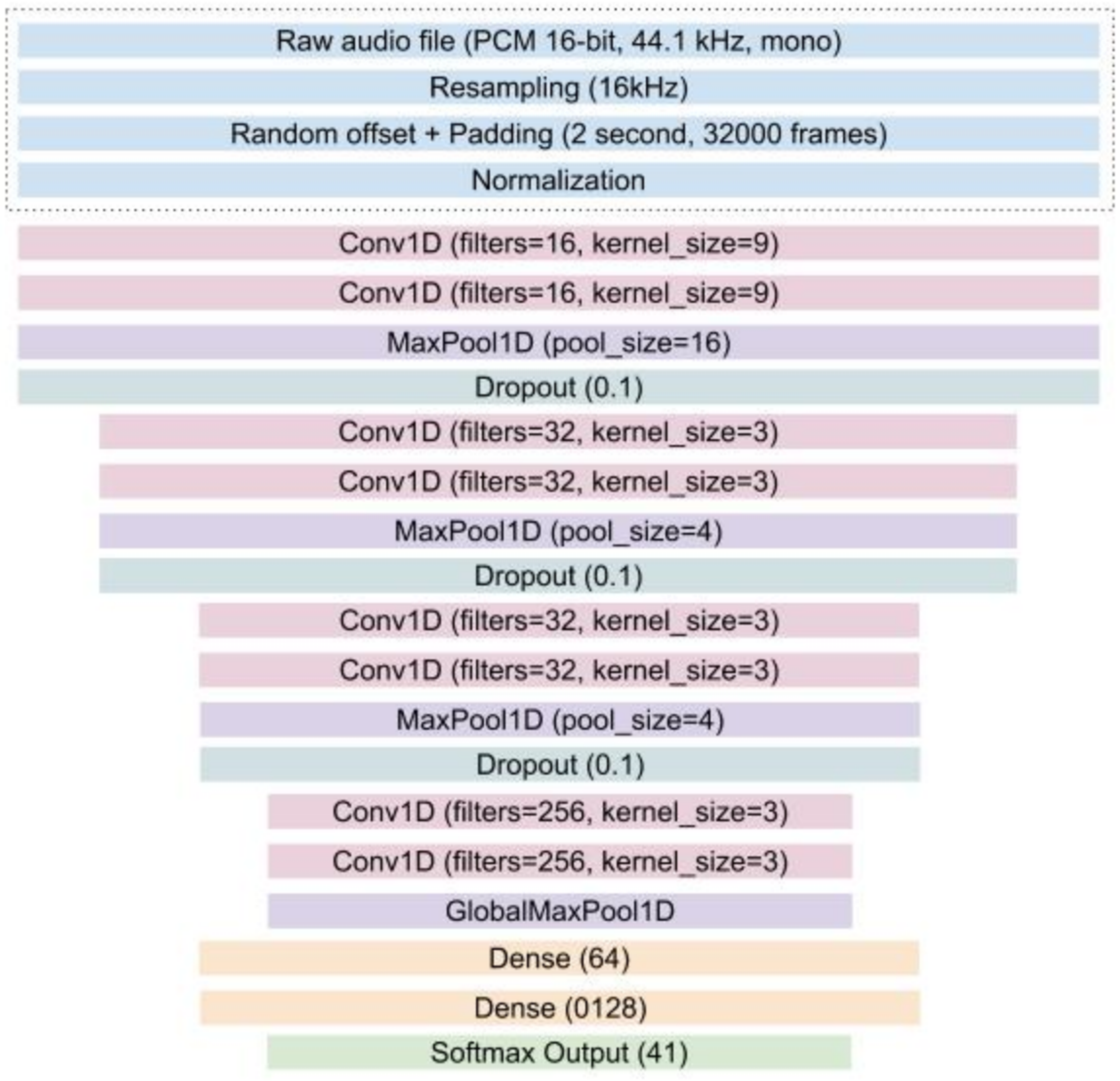
In this section, the process for which metrics, algorithms, and techniques that you implemented for the given data will need to be clearly documented. It should be abundantly clear how the implementation was carried out, and discussion should be made regarding any complications that occurred during this process. Questions to ask yourself when writing this section:
- _Is it made clear how the algorithms and techniques were implemented with the given datasets or input data?_
- _Were there any complications with the original metrics or techniques that required changing prior to acquiring a solution?_
- _Was there any part of the coding process (e.g., writing complicated functions) that should be documented?_

To accomplish this task the implementation of the model required a step for handling the audio feature extraction and training the model.

1. Load both csv's for both curated and noisy training data sets as well as the test set into memory as pandas dataframes. For each row in the dataframe a file name as well as a list of labels is provided.

2. Remove any rows for which the list of labels contains a label which is not existent in the test set (exploratory analysis reveals that the only labels not in the test set are the ones containing multiple labels and the test set)
3. Specify a configuration for model training parameters including a normalized audio duration, sampling rate, bit depth, etc.
4. Specify a neural net model architecture following the pattern show below*
   a. We use a combination of 2 convolutional layers, 1 max pool layer, and one dropout layer with a relu activation in sequence 4 times with gradually increasing layer size.
   b. Finally two fully connected dense layers
   c. A softmax output layer with 80 nodes corresponding for each label in the test set
5. To handle the rather large audio data set we need to create a data generator which inherits from keras.utils.Sequence
   a. The data generator will feed in data to the model in batches and ensure safer multiprocessing
   b. File paths will be handed to the generator and in each batch it will reach into the data folders to load the current batch's audio files into memory using librosa.load()
   c. Handle feature extraction (explained for each feature input below**)
6. Split the data into train and validation sets using k-fold cross validation across 10 folds
7. For each fold use keras' fit_generator and predict_generator functions to train and evaluate the current model.
8. Save the model's best weights as .h5 files
9. Average the predictions from each of the k-folds using the geometric mean
10. Create a submission file for predictions on the test set
    a. Select the output nodes for the top 3 highest scoring labels
    b. Save them as a space separated list onto a new dataframe
    c. Save the new predictions down to csv
11. Compare the model's accuracy and validation accuracy on the test data set across each of the 3 different feature inputs to see which performed best

*Note: For the 2D Convolutional architecture the Conv1D and MaxPool1D layers are replaced with Conv2D and MaxPool2D


Raw audio file (PCM 16-bit, 44.1 kHz, mono)
Resampling (16kHz)
Random offset + Padding (2 second, 32000 frames)
Normalization
Conv1D (filters=16, kernel_size=9)
Conv1D (filters=16, kernel_size=9)
MaxPool1D (pool_size=16)
Dropout (0.1)
Conv1D (filters=32, kernel_size=3)
Conv1D (filters=32, kernel_size=3)
MaxPool1D (pool_size=4)
Dropout (0.1)
Conv1D (filters=32, kernel_size=3)
Conv1D (filters=32, kernel_size=3)
MaxPool1D (pool_size=4)
Dropout (0.1)
Conv1D (filters=256, kernel_size=3)
Conv1D (filters=256, kernel_size=3)
GlobalMaxPool1D
Dense (64)
Dense (0128)
Softmax Output (41)

**Feature Extraction steps

1. Raw WAV Audio (PCM)
    a. Use librosa to load in the raw 1D audio array
    b. Clip / Pad data
        i. Use librosa.effects.trim to remove any leading silence
        ii. For clips longer then the specified duration clip any data longer than 2 seconds after any initial leading silence
        iii. For clips less than 2 seconds pad them on the tail with zeros.
    c. For each value in the array representing amplitude, normalize it on a scale of 0 - 1
2. MFCC
    a. Use librosa to load in the raw 1D audio array
    b. Clip / Pad data (same as raw audio)
    c. Use librosa.feature.mfcc to obtain an MFCC representation of the audio
3. Mel-Spectogram
    a. Use librosa to load in the raw 1D audio array
    b. Clip / Pad data (same as raw audio)
    c. Use librosa.feature.melspectrogram to obtain a spectogram representation of the audio
    d. Convert the monochromatic image to 3 channel color

    e. Resize to square 128x128 size

Refinement

Due to the size of the noisy dataset in particular both of the 2D convolutional nets ran into massive performance problems. Even while running on an AWS GPU powered p2.xlarge instance they took about 3-4 days to train. Also they required increasing the allowed memory size to download the audio datasets from kaggle. To prevent the training from going too long several factors needed to be tweaked

1. Total number of layers and layer sizes was decreased.
2. Audio duration was capped at 2 seconds
3. Multiclass labels were ignored

Performances on the curated tended to be quite good by comparison so noisy data factors tended to drive model architecture decisions.

IV. Results

(approx. 2-3 pages)

Model Evaluation and Validation

The results of each model and feature input format results as follows:

1. Random Forest
    a. Curated data
        i. Raw Audio
            1. 97.4% accuracy on the training data
            2. 2.56% on the validation data
    b. Noisy data
        i. Raw Audio
            1. 99.5% accuracy on the training data
            2. 1.26% on the validation data
2. Convolutional Neural Net
    a. Curated Data
        i. Raw Audio
            1. 52.5% accuracy on the training data
            2. 47.9% on the validation data
        ii. MFCC
            1. 87.4% accuracy on the training data
            2. 35.6% on the validation data
        iii. Mel-Spectogram
            1. 99.9% accuracy on the training data
            2. 41.3% on the validation data
    b. Noisy Data
        i. Raw Audio

ii. MFCC
    1. 46.9% accuracy on the training data
    2. 22.5% on the validation data
iii. Mel-Spectogram
    1. 61.4% accuracy on the training data
    2. 19.1% on the validation data

Compared to the benchmark random forest, neural nets even with a relatively small number of layers and nodes dominated in performance. The random forest classifier tended to overfit to the training data set and never discovered any patterns that could be meaningfully applied to the validation set.

Breaking down the performance of the various feature inputs was much more tricky. On the whole, raw audio appeared to perform the best as it maintained a close margin of accuracy between the train and validation sets. However this is likely due to the fact the the early stopping callback was rarely ever triggered during the training of this model which meant that it was able to train for nearly twice as many epochs as any of the other models. This issue could likely be corrected by adjusting the optimizer for the 2D convolutional models. Particularly by decreasing the learning rate and adding more momentum to the model.

V. Conclusion

(approx. 1-2 pages)

Reflection

The results found strongly suggest the utility of the CNN models over the benchmark but were less conclusive for selecting an optimal feature input. It is appears possible that a raw audio format trained with more layers, stacked temporal difference frames, and more training time could indeed be the optimal format.

Considering the number of labels was 80 and the best performing model achieved an accuracy of 47.9% on unseen data I believe the design of the model represents a successful step in the right direction for achieving a solid audio tagging model.

Considering the body of literature out there that suggests reformatting audio input into MFCC or Mel-Spectrum formats I was very surprised to find the raw audio format performing better under a nearly identical model architecture. Perhaps the main advantage of such as setup would be to take advantage of transfer learning techniques that are not yet available for raw audio however this warrants additional investigation. In the future I'd be very curious exploring more ways to improve the raw audio model.

Improvement

For future improvements to the model I believe it would be worth experimenting with more techniques to overcome overfitting such as L1 or L2 regularization. In addition slowing the learning rate even further and increasing momentum will likely help to overcome some of these issues. While these are small tweaks I'll save them for a future project since obtaining the results for these models already took several days to arrive at per model given the enormity of the data set.

Applying a stochastic gradient descent optimizer as well as a non-fixed learning rate may have helped to improve model performance under the strenuous data load.

In terms of feature extraction I believe removing the color channels on the spectogram images and replacing them with previous frames to add an element of temporal difference to the model would have been interesting to try.