Audio Classification for Smart Home Audio Systems: Project Documentation

Sean Simon | Springboard Data Science Career Track Capstone 3

Introduction

Audio systems are popular forms of entertainment and communication, and are increasingly becoming integrated into wider multi-room smart home environments. With these systems of speakers and their apps, voice control and automatic content suggestions are becoming the norm. An extension of this is to not just listen to voice, but other environmental sounds and use that stimulus to not only make richer content suggestions, but to enable a deeper feature set. For example, in a smart home with an array of speakers in discreet rooms, audio content in one room could turn on or off depending on if human sounds such as footfalls are identified as leaving that room and entering another. This particular project tackles this idea by training a neural network to recognize 41 types of common sounds and ultimately proposes three smart audio customer-facing features opened up by this development: pet care, delivery/guest arrival, and a smart party DJ.

Methodology

An audio data set classified by 41 labels freely available online was used to train a convolutional neural network to identify sounds by their labels. All in all, the data set contained 11,073 sound samples split into a training set of 9,473 samples and a testing set of 1,600 samples. Each sample was between 300 ms and 30 s in length, with most files being less than 4 s in length (Fig. 1).

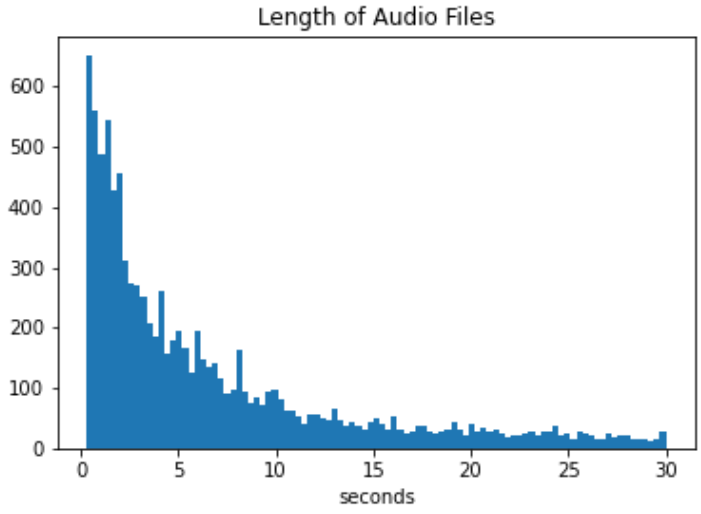

Fig. 1: Distribution of audio file length

Of the 41 labels, within the training set the maximum sample count per label was 300, and the minimum was 94 (Fig. 2)

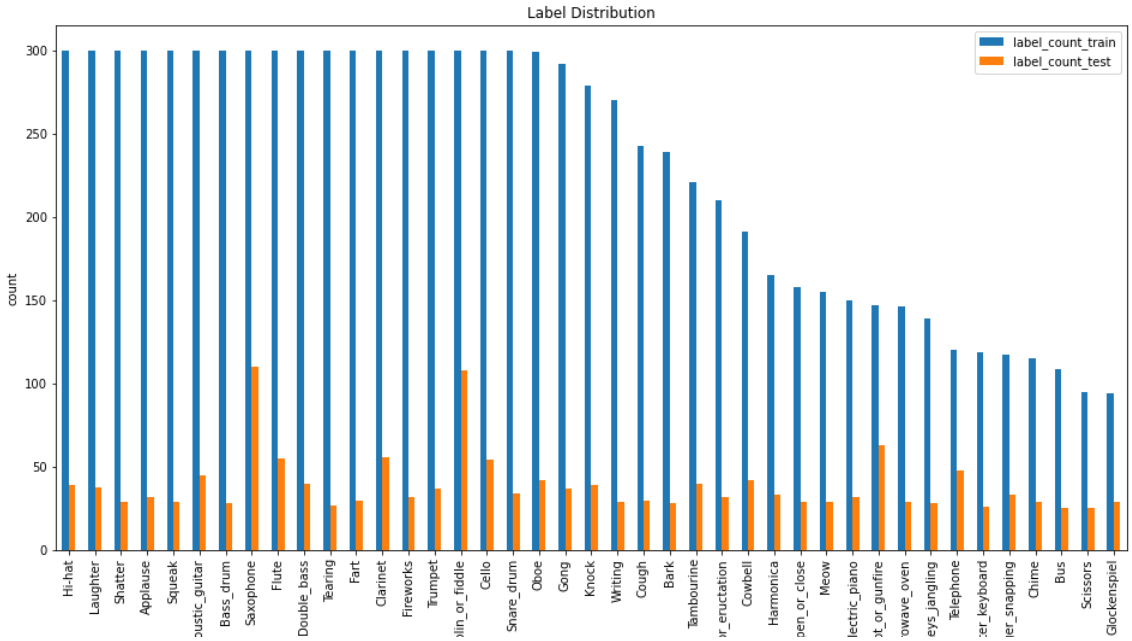

Fig. 2: Data set label distribution

The testing set featured a more or less equal distribution of labels. Additionally, only a subset of the training samples, a total of ~3.7k were manually verified as being correctly labeled while the rest featured a label accuracy estimate of 65 - 70% correctly identified.

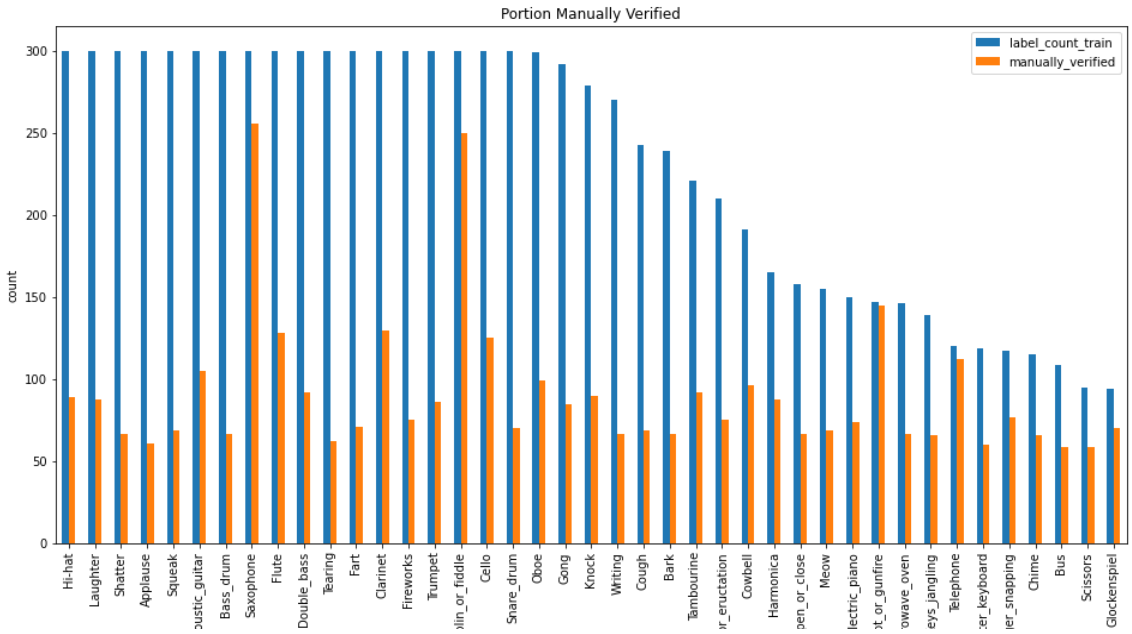

Fig. 3: Portion of samples manually verified

Audio data is represented as a series of amplitudes over time, recorded at a constant rate. The files in this dataset represented amplitude as a value between -32,768 and 32,767 reported 44,100 times per second (Fig. 4). Specifically known as mono 44.1 kHz 16-bit audio (.WAV).

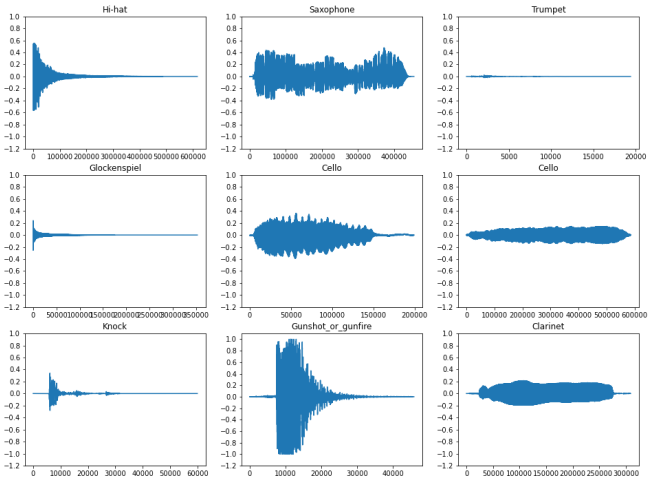

Fig. 4: Audio data waveforms (amplitude vs time), with amplitudes scaled to be between -1 and 1 (from -32,768 to 32,767)

However, this data is more useful as a spectrogram, which can be thought of as a heatmap of frequencies. Spectrograms can be acquired through a method called the Short-Time Fourier Transform which decomposes discrete segments of the audio file into its component frequencies and then plots those frequencies versus time, essentially drawing an image of frequency on the vertical axis and time on the horizontal (Fig. 5).

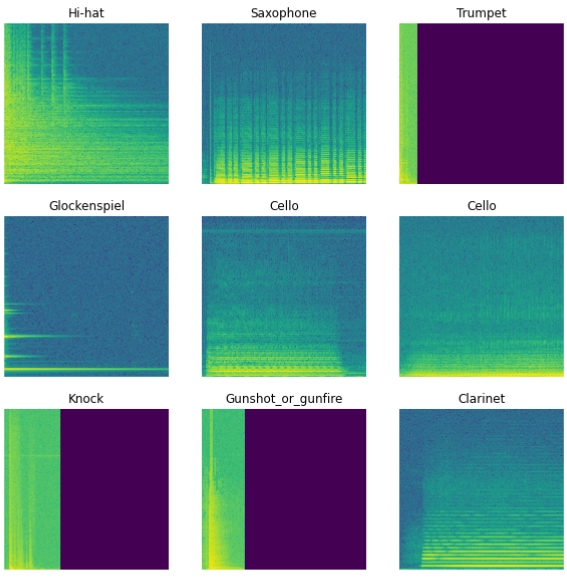

Fig. 5: Audio spectrograms, with time on the x-axis and frequency on the y-axis

It can be seen that uniquely labeled sounds have unique spectrograms. For example, cellos have a lot of low end frequencies in comparison to saxophones with higher frequencies. Certain sounds like a glockenspiel or gunshot fade to nothing faster than something that rings out, like a hi-hat. Further, within a label, spectrograms look similar as well (Fig. 6)

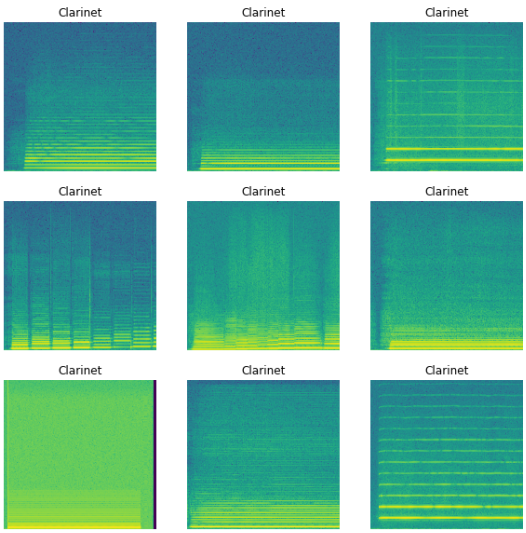

Fig. 6: In-label spectrograms

With audio files represented as images, a convolutional neural network (CNN) can be used to recognize patterns in the images and thereby learn which patterns relate to which labels. However, to do this, the images needed to be all of the same size, so all of the audio samples were made to be 4 s in duration. This was done by either cropping the longer files short, or zero-padding the short files to be 4 s long, but with no amplitude information at the end. For this project, the audio pre-processing and CNN fitting was done with the TensorFlow library, and used a CNN with 2 convolutional layers, a max pooling layer, a dropout layer, a flatten layer, a densely connected layer, and then another dropout and densely connected layer. The audio was also normalized prior to fitting. This design allows the CNN to pick out patterns, and then simplify the network for fitting.

All in all, several parameters were chosen throughout this process, as summarized in Table 1.

Parameter	Value
Audio length	4 s
STFT Frame Length	555
STFT Frame Step	343
Spectrogram resizing	256 x 256 px
Convolutional layer 1 filters	32
Convolutional layer 1 kernel size	3
Convolutional layer 1 activation function	relu
Convolutional layer 2 filters	64
Convolutional layer 2 kernel size	3
Convolutional layer 2 activation function	relu
Dropout layer 1 rate	0.25
Dense layer 1 units	128
Dense layer 1 activation	relu
Dropout layer 2 rate	0.5
Optimizer	Adam
Loss	Sparse Categorical Cross-entropy
Metrics	Accuracy

Table 1: All parameters in the model pipeline

Results

All in all, the model achieved 52% accuracy on a testing set. This is actually quite good, considering that randomly assigning 1 out of 41 labels would be about 2.5% accuracy. The tendency of neural networks to overfit was apparent in the continuing decrease in loss value of the training set, but no further decrease in the validation set loss over training epochs (Fig. 7).

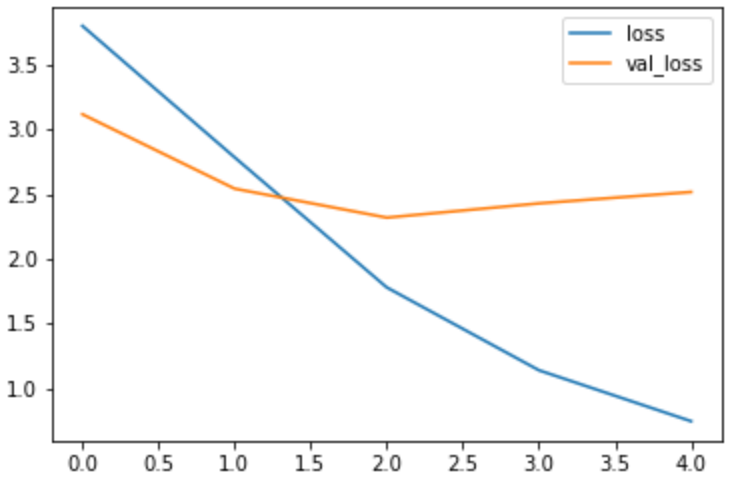

Fig. 7: Training and validation loss per epoch; after 4 epochs validation loss no longer decreased

The 52% accuracy can be visualized in a confusion matrix (Fig. 8), which shows what the label should be on the y-axis, and how it was predicted on the x-axis. This can help understand where errors are being made.

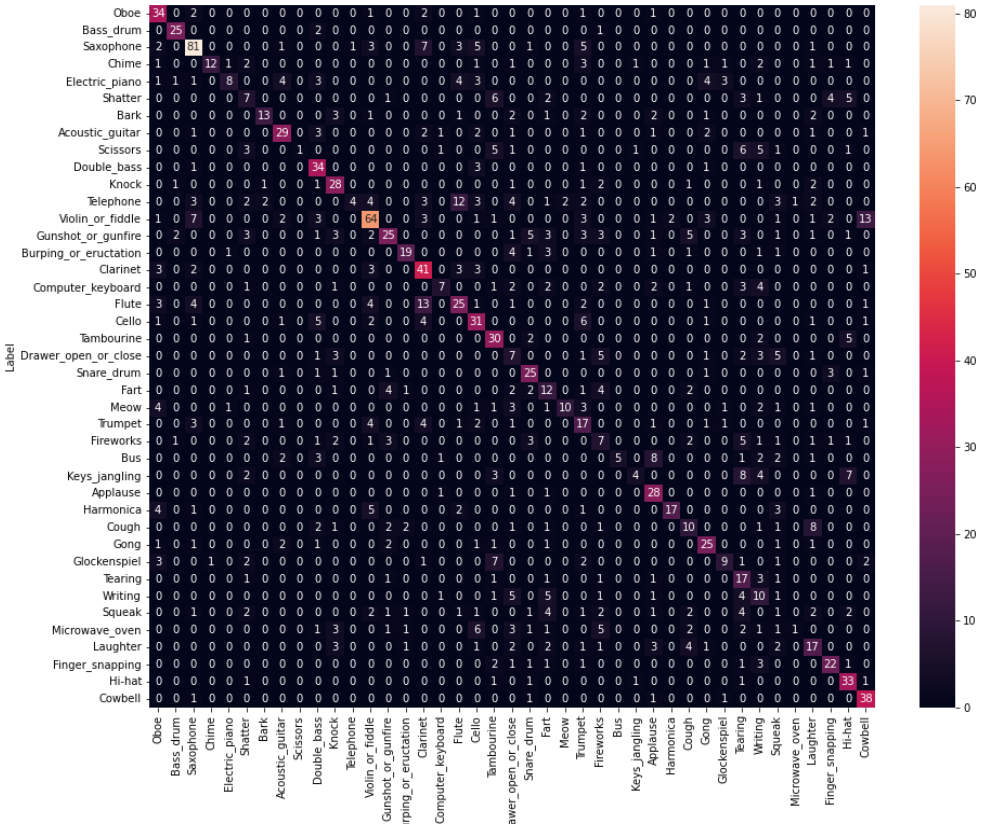

Fig. 8: Confusion matrix

Looking at the clarinet, this label is confused with oboe, saxophone, violin, flute, cello, and cowbell. 3 of the 6 are woodwind instruments, so this seems reasonable. 2 of the 6 are string instruments, and it can be seen in the spectrographs above that clarinet and cello appear rather similar.

Usage

The model works on 16-bit, mono, 44.1 kHz .WAV files. To predict a label for new audio content, simply input the path to the audio file, zip the audio file name with a dummy label, call preprocess_dataset_new() on the zipped audio/label object casted to a list, get the resulting spectrogram(s) from index 0 of the returned Dataset casted to a numpy array and then get the prediction by calling np.argmax(model.predict(), axis=1) on the spectrogram(s). The result will be indexes on which the all_labels variable can be subscripted to get the predicted label names.

Conclusion

The model makes the correct prediction half of the time. This is enough to introduce quality of life type features to smart home audio systems, where the system can make suggestions to the user based on recognized sounds, but ultimately the user has control over what happens. For example, if laughter and finger snapping are identified, a more lively and louder party playlist can gradually be cued up - however, the user can still change the music back to the original volume, change the playlist, etc. Other features that can be introduced are recognition of a dog barking and a gentle reminder to let the dog out, or playing the sound of a knock at the door across all speakers in the house to amplify recognition.

However, notably, customer facing features that would employ an on/off type response, such as alerting the police to a break-in if a window shatter or gunshot are heard would require much higher than 50% accuracy and are recommended as application targets for future work on this project.

Next Steps

A notable short fall of this methodology is cropping all input files to 4 s. This removes valuable information from longer samples, and adds false information to short samples in the form of zero-padding. Future work should investigate how to handle audio files of multiple lengths. Secondly, none of the audio files were filtered prior to converting to spectrograms. In a listening, several had a lot of white noise on top. Future work should investigate pre-cleaning the signals. Additionally, STFT parameters are often debated, as there is a trade off between high resolution in time versus high resolution in frequency - future work should strive to find the optimal tradeoff for this CNN pipeline. The CNN parameters themselves are unoptimized - future work should establish hyperparameter optimization. Lastly, the data set should be opened to more than 41

labels in order to provide a richer set of home-facing sound features. With the exploration of these steps, it is hoped that far greater than 50% accuracy can be achieved on a wider set of samples and thereby open up even more opportunities to add new features to smart home audio systems.