

# Musical Instrument Classification

Group 30: Tim Woodford, Jared Pham, Jonathan Lam  
{twoodfor, jspham, jhlam}@eng.ucsd.edu

*Abstract - In order to create a sound-based musical instrument sound classifier, we trained three different machine learning architectures. The dataset used to train our machine learning models was Google's Audioset, which contains labeled 10-second music clips from Youtube. After training our machine learning models, we found that a fully-connected neural network, with a reduced dataset classes, offered the best prediction accuracy of all our machine learning architectures.*

## Introduction

Machine learning for audio signal processing has attracted a large amount of attention recently for its uses in speech recognition. It has also been used in passive sonar processing, and in a few instances for more generic classification problems. In contrast to these problems, musical instrument classification is not interested in the semantic content of the sound, but rather the particular physical object that produced the sound.

In general, musical instruments consist of materials that produce certain frequencies in response to an energy input, such as air blown into pipes or a stick striking part of the instrument. Tonal instruments generally have some means of adjusting the resonant frequency of the instrument, which allows the instrument to change the note that it is playing. Instruments that are able to resonate at lower frequencies (generally the result of physically larger components) can lower notes, and vice versa. However, most of the variation in instrument sounds comes not from the range of notes that the instrument can play, but from the overtones produced by playing the instrument. The overtones present are determined by the physical characteristics of the instrument, which vary

between instruments, and are not necessarily straightforward to model.

The particular problem of musical instrument identification is useful because of the availability of a good data set. However, identifying the type of object generating a sound is a general problem with uses beyond musical instruments, including identifying wild animals and anomaly detection for mechanical systems.

## Related work

Prior work explored machine learning on sound data for a variety of applications. Only a few past works have specifically focused on identification of musical instruments. The paper, *Automatic Identification of Instrument Classes in Polyphonic and Poly-Instrument Audio*, used MIDI-synthesized audio to develop a large training set for a neural network. The F-score for this data set (mean of the precision and recall) was 0.88 using a Deep Belief Network. However, this method of generating training data does not allow for variation of sounds within a particular type of instrument, different microphones, and varying environments [Hamel].

As part of collecting its Audioset data, Google has also created a simple benchmark, albeit with a much larger set of categories. The benchmark utilized a shallow, fully-connected neural network in conjunction with a pre-trained CNN [Hershey, Gemmeke] to achieve an average precision of 0.314. This project attempts to classify wide variety of sounds, into a variety of categories. Within the realm of music, it includes categories for both the instrument(s) making the sound and the music genre. Music genres are highly abstract categories, over which humans often disagree, whereas it is relatively easy to

assign ground truth for instruments. Our project, instead of working on the broad problem of classifying many different sounds, explores the much more specific problem of differentiating between musical instruments, which is a better-defined problem.

In recent years, speech recognition has been a major application of machine learning on audio data. One of the many papers, Baidu's Deep Speech architecture was designed for the unique problems of speech recognition [Hannun]. It uses an RNN-based architecture to achieve an error rate of around 16%. Speech recognition systems in general are looking to solve a somewhat different problem than we are; in particular they depend on the context around a word to help classify the word. Thus, speech recognition models are generally not especially helpful for our chosen application.

### Dataset and Features

The sound clips we are using come from Google's Audioset, which specifies in turn references publicly-available YouTube videos. The data provided by Audioset include the video link, timestamps specifying the start and end of the relevant video clip, and the label(s) for what type of sound can be heard in the clip. Due to Audioset containing data not relevant for what we are doing, we began by removing all examples and labels not relevant to music. We then further reduced the size of the data set by removing additional data not directly specifying the instrument (i.e. music genres, music moods), with the resulting reduced dataset only containing 89 labels. Finally, we created an additional data set that grouped highly similar instruments into broader categories, such as singing, organs, drums and bells. We later ran our neural network models on this data set, and compared it to the results for the machine learning with the original 89 labels.

The Google dataset is split into three parts: balanced, unbalanced, and test. There are 59

examples per class in the balanced training set, including multi-label samples. There is also an unbalanced dataset available, with many more examples of some classes. However, Google has cautioned that some of the labels on the unbalanced set may not have been rigorously validated. The test set also has 59 examples of each class, and can be used as a test set for our machine learning results. However, some of the tools we used automatically split the training and test set for us, and in these cases we would combine Google's training and test sets and allow the tool to re-separate training and test data automatically.

There are a total of 128 features for every second in the clip, so in total we have 1280 features for each video input. To generate these features, we use the Mel-Frequency Spectrum transform, which is commonly used in audio applications to approximate human hearing. This transform works by taking the FFT of the signal, using a set of triangular filters placed on a log scale to resample to frequency domain, taking the log of the results, then taking a discrete cosine transform of the results. An example of how the Mel-Frequency Spectrum transform generates our features.

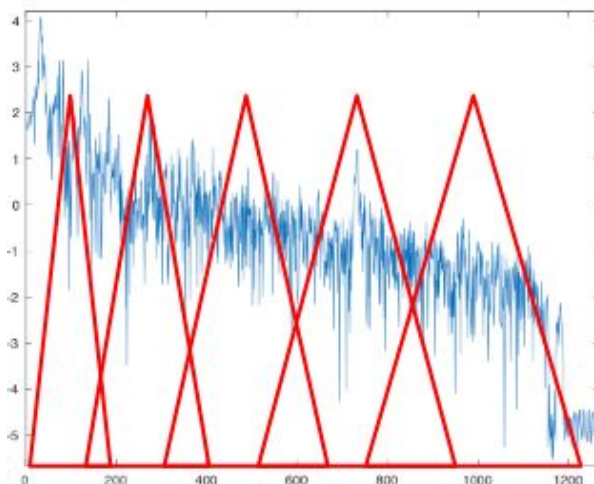


Figure 1. Mel-Frequency Spectrum.

## Methods

There were three main machine learning architectures that we trained the data in, a support-vector machine, a random forest classifier and a fully-connected neural net. All our machine learning models used the 1280 features as inputs.

**Support Vector Machines (SVM)** creates hyperplanes by finding data that can be linearly separable in higher dimensional space whenever data is not linearly separable in the current dimensions of the given data. We used a variation of an SVM that is an one versus all/rest classifier. For each class we have, a binary SVM is used to separate all members of a selected class from all other data points we have. [Aisen, Szymański]

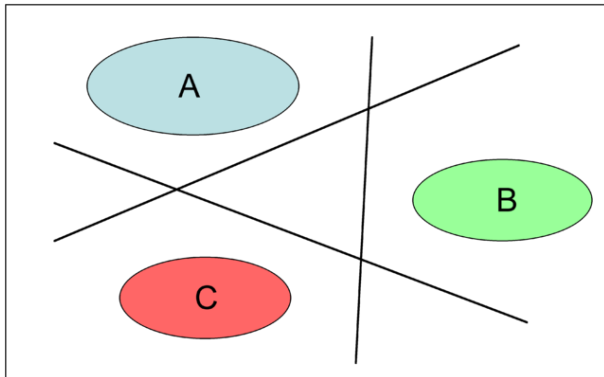


Figure 2. One vs. Rest SVM[Aisen]

We also tried a binary relevance classifier for multiclass configurations, which works by individually creating a binary classifier for each class label that disregards any other labels, and it also transformed multi-label problems into single-label by treating unique combinations as a new label [Szymański].

**Random Forests** are built from ensembles of decision trees. Each tree is built using a random selection of examples, and each node in a tree splits based on a random selection of the features. The algorithm we used selects  $\sqrt{N}$  features out of the total  $N$  features for each split. The algorithm trains an ensemble of these decision trees, with

each tree being trained differently due to the random selections. To classify an input vector, this algorithm sends the data to each tree in the forest, and determines the best output based on the consensus of the trees.

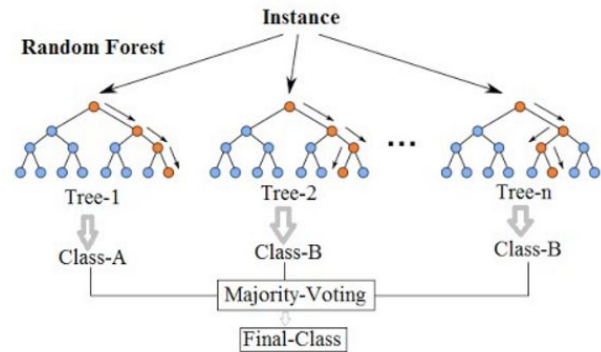


Figure 3. Model of a Random Forest[Koehrsen]

Since random forest classifiers take only single-dimensional data, we collapsed the  $128 \times 10$  matrix into a single 1280-element feature vector. We then used a randomized parameter search for hyperparameter optimization, optimizing for the maximum depth, the minimum samples per leaf, the minimum number of samples needed to split a node, the number of estimators, and the method used to determine the number of features used to determining splits in the tree. We also tested whether to use bootstrap samples for tree building, and found that this configuration was clearly disadvantageous. We ran the random search with 60 candidate sets of hyperparameter configurations, with three runs for each configuration for a total of 180 training runs. The results of random forest modelling are discussed in the following section.

**Fully-Connected Neural Networks (FC-NNs)** select a category by finding the maximum element in the output vector. The output is generated by 1 or more rounds of matrix multiplication followed by a nonlinear function. In our case, we use the ReLU function for the first few rounds, and a softmax function for the final round. To train the

neural network, we need to optimize the values of the matrices used to calculate the output vector. Due to the large state space, we cannot globally optimize these values in a traditional way. Instead, we incrementally optimize the matrix values using a technique called backpropagation. Essentially, this method finds the partial derivative of the error function (which compares the actual output to the desired output) with respect to each of the matrix values. The matrix values are then modified in the direction of the negative gradient to reduce the error value.

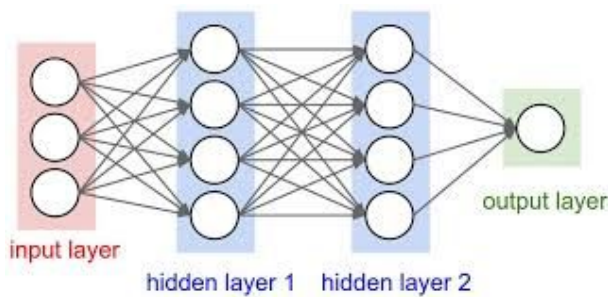


Figure 4. A 3-layer Neural Network[Karpathy]

### Experiment/Results/Discussion

Table 1 shows the accuracy results we obtained for all of the different methods that we used.

Model	Top 1	Top 2	Top 3	Top 4	Top 5
FC-NN (single, reduced)	0.68	0.82	0.87	0.92	0.93
	1	1	1	1	1
FC-NN (multi, reduced)	0.70	0.83	0.88	0.91	0.93
	1	1	1	1	1
FC-NN (single)	0.46	0.61	0.69	0.73	0.76
	0.98	0.99	1	1	1
FC-NN (multi)	0.52	0.63	0.70	0.75	0.79
	0.92	0.97	0.98	0.99	1
FC-NN	0.63	0.75	0.81	0.85	0.87

(multi, unbal)	0.68	0.80	0.86	0.90	0.92
Random Forest (reduced, single)	0.61	N/A			
	1				
SVM One vs Rest	0.38	0.50	0.56	0.61	0.64
	1	1	1	1	1
Binary Rel. SVM	0.54	0.65	0.72	0.75	0.78
	0.96	0.99	1	1	1

Table 1. Accuracy Results

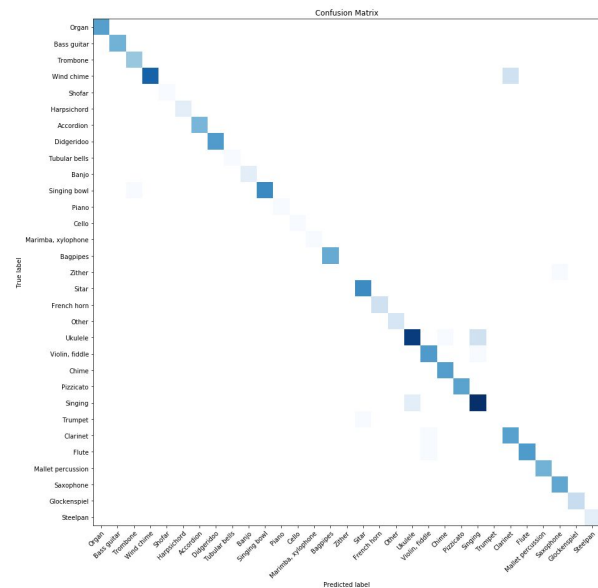


Figure 5 Confusion Matrix of the Reduced Dataset from the FC-NN

From our results we can deduce the following for each machine learning model:

**SVM:** The results of our SVM was the worse of all implementations we had tested. With a 38% Top 1 accuracy and a 64% Top 5 accuracy, this indicates that SVM is not well suited for our uses. It shows that audio clips are not easily linear separable by a SVM with just the Mel-Frequency data we have.

*Random Forest:* Unlike the other machine learning models, random forest can only give us 1 prediction instead of top 5 results. The prediction that is given to us is comparable to the results we got from our FC-NN. Random forest predicting this high is understandable due to the patterns that can be predicted due to musical instrument patterns.

*Fully-Connected Neural Network:* We used Talos [Kotila] for hyperparameter optimization which allowed us sample a random 10% of permutations (~500) of the different parameters whose results will be attached as CSV, and we chose the combination with the highest validation accuracy. After testing several variations of our dataset (single label only, multi label only and reduced variants), we obtained varying qualities of Top-N accuracies. When we just fed the FC-NN unmodified data from the balanced set, our results were not that great for Top-1 accuracy. Only when we expand to Top-5 accuracy is where accuracy scores above 75%. Our training set sits at an accuracy of 100%, which shows that our model is overfitting for the training data set. Unfortunately, even using dropout did not seem to substantially improve our results on the larger data set. We tried grouping all similar music classes (like female singing, male singing, child singing, etc.) into a single class to see the improvements in accuracy. Accuracy does improve from this reduction in the number of classes, but we still have overfitting issues since training is still sitting at 100% accuracy.

In Figure 5, we can see the confusion matrix for our reduced multi-label tests. The errors are mostly uniform, except for a slight uptick in confusion between singing and ukulele. We suspect that the large number of videos containing both ukulele and singing together probably contributed to this confusion.

After we obtained all the results, we questioned if the varying different audio quality had an effect on the classification. When analyzing

a select few of the videos from the balanced dataset, we noticed that lots of the audio clips have varying mic quality and compression that can affect the training of our machine learning models. With training accuracy being very high, it could also mean that our neural network is also learning the actual musical pieces that were being played and also attempting to classify instruments that match closely to those musical patterns.

Afterwards we did try incorporating the large unbalanced dataset by using a stratified train test split of all the data combined, and this extra data ended up making training and testing Top-N accuracy much closer, therefore less overfitting. However, this method is probably less desirable if our chosen metric or goal was different because the unbalanced dataset has a lot more samples of certain popular classes like guitars or singing, and if it gets those right performance improves.

## **Conclusion/Future Work**

In conclusion, we trained three separate machine learning algorithms, which were a SVM, Random Forest and a FC-CNN. The data we were passing into these algorithms are 10 second Youtube clips provided by Google's Audioset. From these Youtube audio clips, we extract 128 features every second through the use of a Mel-Frequency Spectrum transform. In total, each input contains 1280 features that are considered when training our classification algorithms. Out of the three algorithms we tested, the FC-CNN gave us the best results for Top-N accuracies. The SVM model performed the worst and Random Forest gave us results that were close to the FC-NN.

If future work were to continue for this project, we would create a CNN as another architecture to compare our models to. We have read about the successes of a CNN in sound classification because they found that using variations of image classification networks work very well with audio classification.[Hershey] We could also try different preprocessing methods for

the input data. The Mel-Frequency Cepstrum technique is a good general-purpose tool for audio classification, but a method specifically designed to detect the relative levels of the fundamental tone and harmonics may produce better results for our specific case. Filtering the audio clips could also be another option.

### Contributions

Most of the work for this project was done while the group was in the same location, working on the project together. This included most of the initial work of finding and preprocessing data. Jared did hyperparameter tuning for the NN classifier and some extra data processing. Tim tuned the random forest classifier. Jonathan and Tim did most of the report writing.

### References

1. Aisen, Ben. "A Comparison of Multiclass SVM Methods." A Comparison of Multiclass SVM Methods. N.p., 2006. Web. 14 June 2019.
2. Gemmeke, Jort F., et al.. "Audio Set: An Ontology and Human-labeled Dataset for Audio Events." *Google AI*. N.p., 2017. Web. 15 June 2019.
3. Hamel, Philippe, et al. "Automatic Identification of Instrument Classes in Polyphonic and Poly-Instrument Audio." *ISMIR (2009)*.
4. Hannun, Awni. "Deep Speech: Scaling up End-to-end Speech Recognition." *ArXiv.org*. N.p., 19 Dec. 2014. Web. 15 June 2019.
5. Hershey, Sourish, et al. "CNN Architectures for Large-Scale Audio Classification." *International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE (2017)*.
6. Karpathy. "Convolutional Neural Networks for Visual Recognition." *CS231n Convolutional Neural Networks for Visual Recognition*. N.p., n.d. Web. 14 June 2019.
7. Pedregosa, Fabian. "Scikit-learn: Machine Learning in Python." *Journal of Machine Learning Research*. N.p., 2011. Web.

<<http://jmlr.csail.mit.edu/papers/v12/pedregosa11a.html>> .

8. Koehrsen, Will. "Random Forest Simple Explanation." *Medium*. Medium, 27 Dec. 2017. Web. 14 June 2019.
9. Kotila, Mikko. "Talos." *Github.com*, Web.
10. Szymański, Piotr, and Tomasz Kajdanowicz. "A scikit-based Python environment for performing multi-label classification." *arXiv preprint arXiv:1702.01460* (2017).