Urban Sound Classification:

With Random Forest, SVM, DNN, RNN, and CNN Classifiers

Chih-Wei Chang
Harvard Univerisity
Cambridge, MA
USA
karen.chiwei@gmail.com

Benjamin Doran
Harvard University
Cambridge, MA
USA
BenjaminDoran@g.harvard.edu

ABSTRACT

In this paper we describe two methods of extracting features from sound data, one focusing on maintaining the time-series nature of the audio sequence – the other expanding signal characteristics. We go on to detail the effectiveness of different models on each method, including tests of Random Forests, Naïve Bayes, Support Vector Machines, and Neural Network architectures such as deep neural network, convolutional neural network, and recurrent neural network. Additional steps such as feature engineering, and future research steps are touched on briefly.

CCS Concepts

- **Computing Methodologies Machine Learning Machine Learning Algorithms Feature Selection**
- **Computing Methodologies Machine Learning Machine Learning Approaches Neural Networks**

Keywords

Mel Banks Cepstral Coefficients (MFCC); Filterbanks; Sound Classification; Feature Extraction; Neural Networks

1. INTRODUCTION

Automatic sound classification has been a field of growing research. In particular, the sonic analysis on environment sounds has generated increasing researches because of its multiple applications to large-scale content-based multimedia indexing and retrieval. However, sonic analysis researches have focused mostly on music or speech recognition. Not only are there scarce works on environment sound, but also very few database for labeled environment audio data. One of the few free large sound data is the UrbanSound dataset created by Justin Salamon, Christopher Jacoby and Juan Pablo Bello in 2014. The UrbanSound dataset is unique for its classification is not just based on the auditory scene type such as nature, human, animal, but on the sources of sound, such as dog bark, car horn.

One of the main challenges facing sound data classification problem is feature extraction. The features of sound data cannot be expressed in vector forms like other type of data such as images and texts. So the feature extraction for sound data is less straightforward. In this project, we applied different feature extraction techniques and compared the model performances on different feature sets. We applied two categories of feature extraction techniques: signal characteristic feature extraction and time series feature extraction. The first method of extraction approaches is expansion and isolation of important "characteristics" from each sample, such as Mel Frequencies Cepstral Coefficient (MFCC), spectrograms, spectral contrasts, and tonal centroid features. The second category of feature extractions are the filterbank, and log-filterbank methods. These extraction techniques filter important information in each of the smaller time segments (frames) of the original signal, thus preserving the time-series format of the raw data.

We applied general classifiers such as Random Forest, Naïve Bayes, Support Vector Machine, as well as neural network models such as deep neural network, convolutional neural network, and recurrent neural network.

Section 2 of this paper describes the UrbanSound dataset in more detail. Section 3 discusses different feature extraction techniques. Section4 presents the models, and section 5 concludes this paper.

2. DATASET

We use the UrbanSound 8k Dataset by Justin Salamon, Christopher Jacoby, and Juan Pablo Bello. [1] The UrbanSound 8k dataset contains 8723 real-field recording samples from 10 classes of different sound sources: air conditioner, car horn, children playing, dog bark, drilling, engine idling, gun shot, jackhammer, siren and street music. Most of the samples have durations of 4 seconds. But some of them can be as short as 2 seconds. The total length of the 8732 samples is 8.75 hours. Counts of samples in each class are displayed in Figure1.

The creators of UrbanSound8k has pre-divided the 8732 samples into 10 subgroups for 10-fold cross validation. Because the 8732 sound excerpts are cropped from a smaller number of longer recordings, randomly partitioning the 8732 samples for cross validation can result in over optimistic results because excerpts from the same original recording can appear in both the training and validation/test dataset. The pre-divided 10 folds avoid the above issue and make classes in each fold roughly balanced.

The downloaded UrbanSound dataset contains 8732 .wav format files of different length and resolutions. We parsed through all the .wav files and use the python Sound File library (https://pypi.python.org/pypi/SoundFile/) to convert the .wav files into matrixes of numbers.

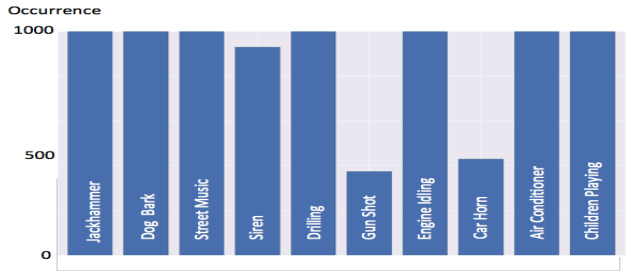


Figure 1 Number of occurrences of sounds in different classes

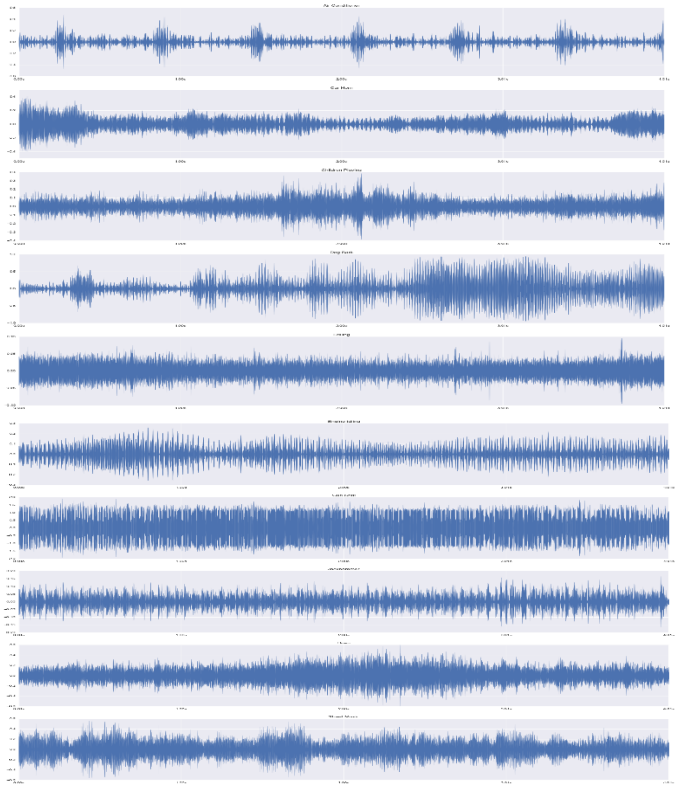


Figure 2 shows the plot of raw data for a sample from each class

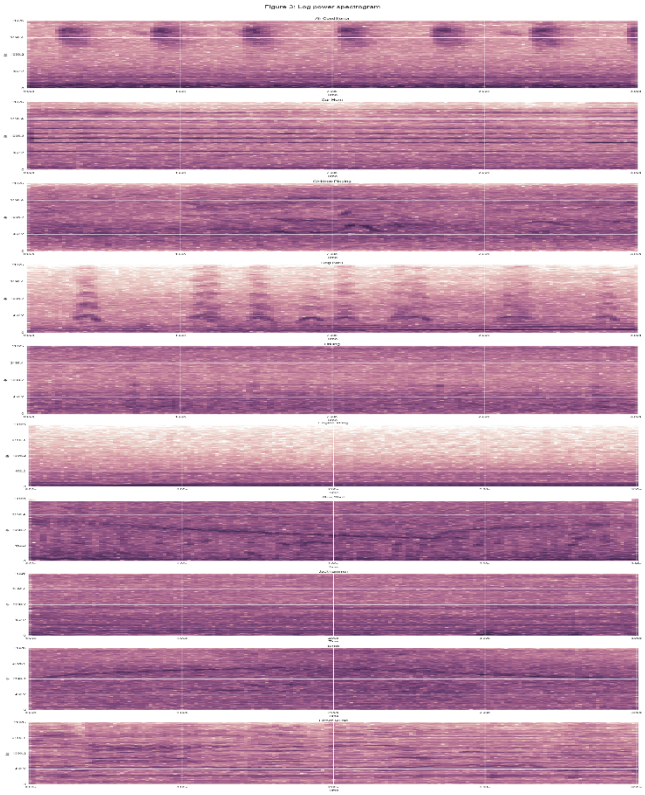


Figure 3 shows the log spectrogram filterbanks

3. FEATURE EXTRACTION

Feature extraction for sound data is less straightforward than other data formats. The parsed raw data of a 4 second sample contains 44,100 values.

Although each signal from different classes shows its distinct patterns, it will be prohibitively expensive to handle such a long sample. Feature extraction from the raw data is hence critical for building classifiers.

The first category of our feature extraction approaches is to extract characteristics from each sample. So the number of features is fixed regardless of the shape of raw data.

One commonly used feature extraction technique in speech recognition is isolating the Mel Frequencies Cepstral Coefficients (MFCC). It is also widely used in environmental sound analysis and has become competitive baseline for bench marking new techniques. Steps to extract MFCC as detailed by James Lyons in his *Mel Frequency Cepstral Coefficient (MFCC) Tutorial* include:

1. "Frame the signal into short frames of 20 – 40 milliseconds. Each frame overlaps 50% of its neighbor frames.
2. "For each frame calculate the periodogram estimate of the power spectrum. This step records what frequencies are presented in each frame.
3. "Apply the Mel filterbank to the power spectra, sum the energy in each filter. This step takes clumps of periodogram bins and sum them up to further reduce the number of features.
4. "Take the logarithm of all filterbank energies. This step is motivated by the fact that human does not perceive loudness in a linear scale.
5. "Take the discrete cosine transformation (DCT) of the log filterbank energies. This step de-correlate overlapping frames. This is essential for HMM classifier to work but is not as relevant for other classifiers.
6. "Keep DCT coefficients 2-13, discard the rest. Experiment results show that dropping the coefficient above the 13th improve performance in automatic speech recognition." [2]

There are other commonly used feature extraction methods, such as Mel-scaled spectrogram, chromagram, spectral-contrast, and the tonal centroid features. The Librosa library has functions to extract all the above characteristics. We ended up extracting a total of 193 characteristics (features) for each sample using these methods.

The second category of our feature extraction approaches is filterbanks. This approach allows us to keep the time-series attribute of the raw data. (e.g. The extracted series of features are also time series)

Computing filter banks and MFCCs involve somewhat the same procedure, where in both cases filter banks are computed and with a few more extra steps MFCCs can be obtained.)[3] To obtain MFCCs, a Discrete Cosine Transform (DCT) is applied to the filter banks retaining a number of the resulting coefficients.

Because of the way we partition each sample into overlapping frames, each filterbanks we extract is highly correlated with its neighbor filterbanks. This autocorrelation can be problematic for some classifiers such as the hidden Markov Chain. However, as we will be implementing the neural network family classifiers, the autocorrelation will not be an issue. It might be beneficial for the classifiers to learn directly from the signal in the time domain.

The challenge we faced in extracting the filterbanks features are different-sized raw data. Since our samples include sound excerpts of different length, resolution, and number of channels. The shape of filterbank features also varies across samples. Because neither Sklearn nor Tensorflow allows varied data shape, we would need to make the size of extracted features identical across samples. Our first solution is zero padding, making the shorter signals as long as the longest ones. Our second solution is to cut each sample into the same number of windows (frames), so each sample will have the same number of windows regardless of the length of the original sound. Shorter samples will have a frame partition that each frame overlaps with others more.

4. MODELS

4.1 Comparison of Neural Networks across Data Sets

We tested with three different architectures of neural network: Recurrent Neural Networks (RNN), Deep Neural Networks (DeepNN), and a Convolutional Neural Networks (CNN). We kept each model simple initially, using at most 3 hidden layers for our DeepNN and 2 for our RNN and CNN. As we see in our comparison plot above, our const_MFCC and features193 datasets gave us the best accuracy across all models, with features193 beating const_MFCC by an average of about 5 percentage points. The only model we were unable to get working for a data set was the CNN for the flat0pad training set that had 27,600 features per sample. Our laptop (i5 CPU, 8gb ram) persistantly froze and/or gave memory errors when attempting to run that particular model.

Figure 4. We see that feature193 and const_MFCC sets gave us the best performance. And among those sets our CNN model gave the best accuracy, with 58% on const_MFCC and 64% on features193 (we later optimized it to 72.8% accuracy on features193)

A large part of why the flat0pad did not do well is simply that It took too long to run models on it. Even for our DeepNN, it could take over two hours to train for 5000 epochs. The RNN could take over seven hours. This meant that we simply did not have as much time to really tune the model to the extent we could the others. To illustrate this point more clearly, the fastest time we got was 30 minutes with our DeepNN, and CNN on the features193 training set. The longest was over 25 hours, for our models on the flat0pad training set. This difference means that we had roughly 50 more chances to turn-over and tune our models on features193 training set than the flat0pad set. A difference in time that is understandable considering that the flat0pad training set is over a 140 times larger than the features193 set, however its size did not bring enough new information about the samples to compensate for the slowness of training.

We did not see as much advantage to maintaining the time-series nature of the data as expected. In comparison between const_MFCC (filterbank method) and features193 (signal type isolation method). The features193 training set gave us moderately better performance. One factor is model turn-over rate, computing on 193 features to 820 total features. Another factor may be the shape of the const_MFCC training set. The filterbank split each audio file into smaller segments. Meaning that any models learning on the time series was learning on the smaller patterns contained and not the overall pattern of the audio sequence. In contrast, for the flat0pad data (also a filterbank method) we flattened the data to a single time series, which let our RNN learn on the overall pattern instead of the smaller segments. And as we can see in Fig. 4, the RNN gave better performance in comparison to the DeepNN on the flattened dataset (flat0pad) to the split dataset (const_MFCC).

The feature engineering we attempted with the const_LogMFCC dataset failed because the const_MFCC data, which the const_LogMFCC dataset is produced from, contained negative numbers. Meaning that we were losing half the data when converting the dataset. We tried to correct that by taking the log of the absolute values, yet we were still losing half the scale, which is the result we see above in the above figure. Unfortunately, even correctly scaling the data by shifting it all to positive values then taking the log, $\log(data + (-min + 1))$ did not give us results that were any better due to the function condensing the upper bounds into the the middle. While our manual attempts at feature engineering were failures, we did have success using Sci-Kit Learn's standard scaler, which added about another 3-4 percentage point increase in accuracy for our CNN and DeepNN (though we only had time to test this result on the features193 dataset).

With only 8732 samples in total, overfitting was a major issue with this dataset. In our RNN, and DeepNN models especially, we had to control with high rates of dropout (as much as 80% on each layer) and L2 regularization (lambda of 0.01 on each layer). Even with those controls we were unable to completely prevent the RNN from overfitting, as shown in the plot below. The complexity of the RNN made it the slowest and hardest to tune. Indeed, the RNN "cell" in Tensorflow acted much like a blackbox, making it challenging to add regularization inside. The RNN also needed as much as 80% dropout, and actually required

Figure 5 All models reach their peak very early, before 5,000 epochs. And while DeepNN and CNN plateau, RNN begins to overfit despite high dropout and regularization.

a higher level of regularization than the DeepNN using a lambda of 0.04.

We had the best results with our CNN as can be seen in the fig. 4 and 5 above. While the DeepNN was comparable, the CNN consistently beat it by a few percentage points across all datasets. Part of this success stems from the CNN sharing its weights across multiple features. Therefore it was not as susceptible to overfitting as the other models we tried. Indeed, while both the RNN and DeepNN needed as much as 80% dropout, our CNN worked just fine with 50%. We did also add regularization to the CNN, however it did not give us as much improvement as we found with the other models.

Remarkably, even though our features193 dataset's features are not related together like the image pixels CNNs are known to be good at classifying against, treating the data like an image worked to an extent. We were able to use the patch size in convolution to really control how much detail from the features we wanted. We found a patch size that combined 10 features to a single set of weights and biases to provide the best results.

This analogy of our features as connected like an image goes only so far. In an attempt to artificially increase our data size we tried some of Tensorflow's image distortion functions on our features193 training set to limited effect. Unsurprisingly, these functions lengthened the time it takes to train to a similar accuracy as not using the distortions. However the accuracy curves, as shown below, do not seem to indicate that training longer would improve accuracy.

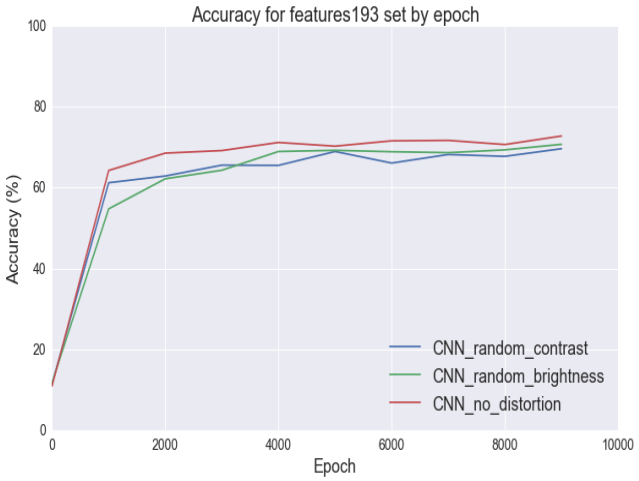


Figure 6. CNN with no distortion on training samples maintains an accuracy consistently higher than the models using randomized brightness and contrast distortions.

While the CNN does have better performance and is more robust, we should also note that it can be as much as three times slower than the DeepNN. The CNN is simply not as scalable as the DeepNN. If needing to work with a larger sound dataset where a small lowering of accuracy is acceptable the DeepNN serves well as the more easily modified and faster model.

5. FINAL RESULTS

5.1 Training Curve Results

Although images distortions did not help much with "increasing" our amount of data, an actual increase in data samples would. We plotted a training curve using the features193 dataset in increments of 1,000 samples recording the training and test accuracies for each batch size after 15,000 epochs with our DeepNN model. While is not difinitve, the plot does to trend toward collision of the training and test accuracies. A trend that appears to continue beyond the sample sizes we are able to capture.

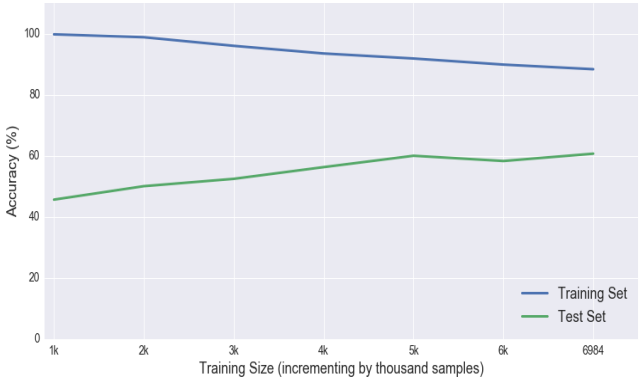


Figure 7. Training sets top line, Test sets bottom line. We see a trend toward convergence between the two lines off to the right indicating that more data samples would increase accuracy.

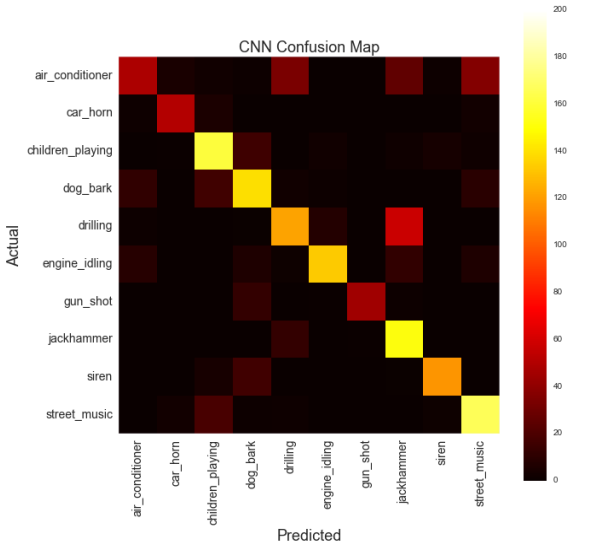
5.2 Optimization Results

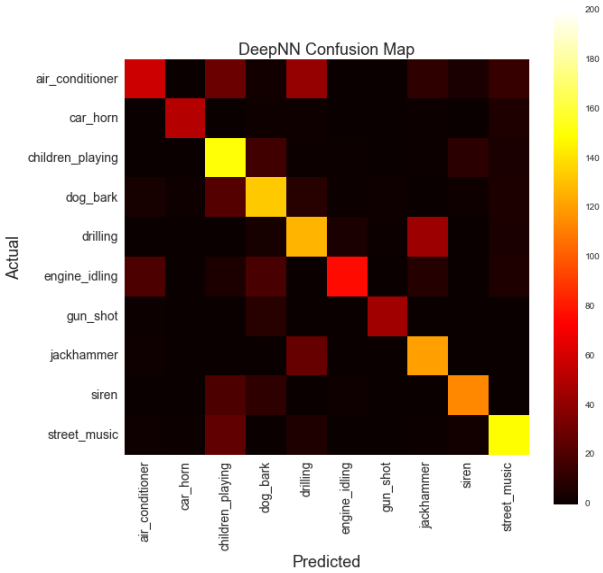
As we have written earlier, in the context of this dataset prone to overfitting, simpler models that gave the most control to add restrictions won out. As such our CNN model in Tensorflow gave the both the highest accuracy as well as AUC score. Our final best predictions are:

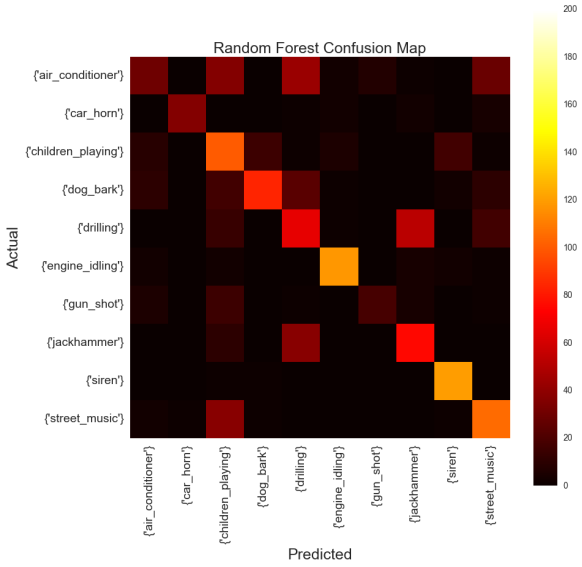
1st CNN	73% acc.	93% AUC
2nd DeepNN	68% acc.	91% AUC
3rd Random Forest	61% acc.	-
4th SVM	59% acc.	-
5th RNN	56% acc.	87% AUC
6th Naïve Bayes	23% acc.	-

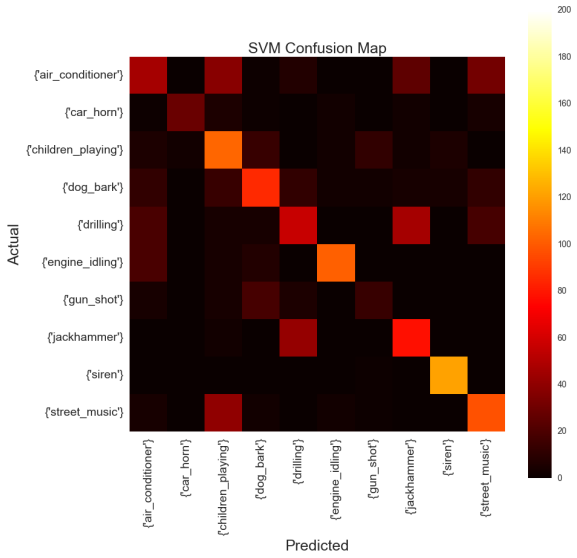
Table 1. Final Accuracy results for models on the features193 dataset.

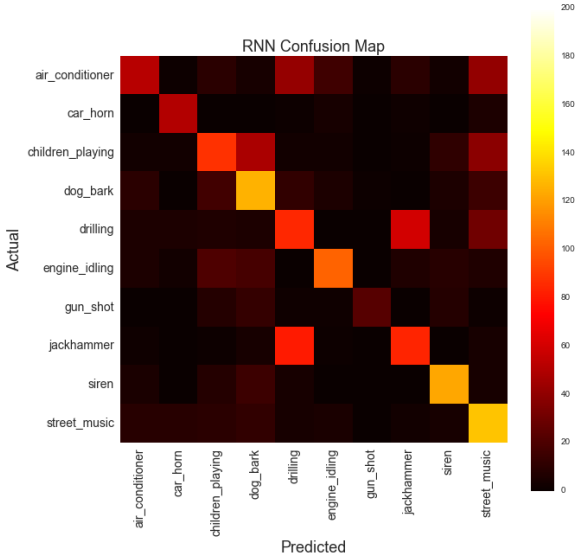
Looking at the confusion maps of our models on the features193 test set, we can see that our models is making the same types of mistakes that a human might: air conditioners for drilling and drilling for jack hammering. This similarity indicates that our feature extraction techniques are validly representing the data, and that we are predicting on the data in a way useful to human interpretation albeit with less accuracy than is hoped for.

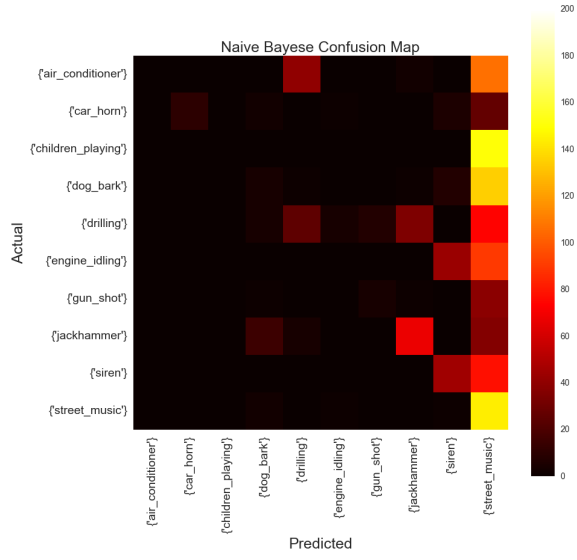

CNN Confusion Map
air_conditioner
car_horn
children_playing
dog_bark
drilling
engine_idling
gun_shot
jackhammer
siren
street_music
Actual
Predicted
200
180
160
140
120
100
80
60
40
20
0


DeepNN Confusion Map
air_conditioner
car_horn
children_playing
dog_bark
drilling
engine_idling
gun_shot
jackhammer
siren
street_music
Actual
Predicted
200
180
160
140
120
100
80
60
40
20
0


Random Forest Confusion Map
{'air_conditioner'}
{'car_horn'}
{'children_playing'}
{'dog_bark'}
{'drilling'}
{'engine_idling'}
{'gun_shot'}
{'jackhammer'}
{'siren'}
{'street_music'}
Actual
Predicted
200
180
160
140
120
100
80
60
40
20
0


SVM Confusion Map
{'air_conditioner'}
{'car_horn'}
{'children_playing'}
{'dog_bark'}
{'drilling'}
{'engine_idling'}
{'gun_shot'}
{'jackhammer'}
{'siren'}
{'street_music'}
Actual
Predicted
200
180
160
140
120
100
80
60
40
20
0


RNN Confusion Map
air_conditioner
car_horn
children_playing
dog_bark
drilling
engine_idling
gun_shot
jackhammer
siren
street_music
Actual
Predicted
200
180
160
140
120
100
80
60
40
20
0


Naive Bayese Confusion Map
{'air_conditioner'}
{'car_horn'}
{'children_playing'}
{'dog_bark'}
{'drilling'}
{'engine_idling'}
{'gun_shot'}
{'jackhammer'}
{'siren'}
{'street_music'}
Actual
Predicted
200
180
160
140
120
100
80
60
40
20
0

6. ACKNOWLEDGMENTS

Our thanks to Justin Salamon, Christopher Jacoby, and Juan Pablo Bello for creating the UrbanSound8K dataset

7. REFERENCES

[1] Justin Salamon, Christopher Jacoby, and Juan Pablo Bello. 2014. A Dataset and Taxonomy for Urban Sound Research. In Proceedings of the 22nd ACM international conference on Multimedia (MM '14). ACM, New York, NY, USA, 1041-1044. DOI=http://doi.acm.org/10.1145/2647868.2655045

[2] James Lyons. 2013. Mel Frequency Cepstral Coefficient (MFCC) tutorial. In Practical Cryptography Online. URL=http://www.practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/

[3] Haytham Fayek. 2016. Speech Processing for Machine Learning: Filter banks, Mel-Frequency Cepstral Coefficients (MFCCs) and What's In-Between. In http://haythamfayek.com/blog/ Online. URL=http://haythamfayek.com/2016/04/21/speech-processing-for-machine-learning.html

[4] Aaqid Sayeed. 2016 Urban Sound Classification, Part I, Part II. https://aqibsaeed.github.io/2016-09-03-urban-sound-classification-part-1/