Urban Sounds Classification

Deep Learning – Final Project Report

Problem statement

The objective of this project is to use deep learning to classify urban sounds from the UrbanSound8k dataset. Whilst there is a large body of research in related audio fields such as speech and music, work on the classification of environmental sounds is comparatively scarce. We have 10 classes of urban sounds and we predict probabilities for a given audio clip belonging to each of the classes. The model performance evaluation metric is classification accuracy.

We predict the following applications of our work:

1. Assistive devices for hearing impaired individuals
2. Smart home security systems
3. Predictive maintenance with airborne sound analysis

Dataset

The UrbanSound8K dataset consists of 8732 labeled sound excerpts (=<4s) of urban sounds. There are 10 classes: air_conditioner, car_horn, children_playing, dog_bark, drilling, enginge_idling, gun_shot, jackhammer, siren, and street_music. Along with the individual files in .wav format corresponding to audio clips, a metadata(.csv) file is provided which tags each file to its class labels

At the end of this section, you can see a visual representation of our data in the time domain.

Aside: Existing Human Accuracy – A human with a decent hearing ability can easily identify the corresponding class of sound. The classes ‘jackhammer’ and ‘engine_idling’ created confusion and were the primary reason for misclassifications by humans. The classes ‘gunshot’, ‘air_conditioner’ and ‘drilling’ were also problematic for humans. The other classes like ‘dog bark’, ‘car horn’ etc. were easier to identify. In general the human accuracy rate can be estimated to be in the range 95% to 98%.

The dataset is of decent size, the total size of the data in a compressed format is 5.7GB. The audio file properties like the sampling rate, the number of channels, and the bit depth weren’t uniform over all the observations and required standardization. Although all the audio clips were clipped at 4s, they still had different durations (the spectrograms extracted had different image resolutions and needed padding).

Audio Augmentation: Since we were dealing with audio files we used the *nlpaug* package to use six augmentation techniques: cropping, loudness augmentation, speed shifting, pitch shifting, time shifting and noise injection, giving us a dataset 6x the original size.

Class Imbalance: Initially we had class imbalance(e.g. siren and gunshot) due to which our CNN models were overfitting, hence we used class weights, so that the model pays equal attention to the minority classes.

Additional Feature Exploration: We also extracted features such as spectral rolloff, centroids, contrast, bandwidth, and zero crossing rate but they didn't add as much value as mel frequency cepstrum coefficients did.The loss and accuracy curves for our final model (Densenet121) shows that there is no significant overfitting

Aside on Mel frequency Cepstrum Coefficients: MFCCs produce a spectrogram of a given audio signal. A spectrogram allows us to view the frequency content of a signal at a particular frequency, time combination. A spectrogram is essentially an image, and thus we can use CNN image classification techniques to classify them.

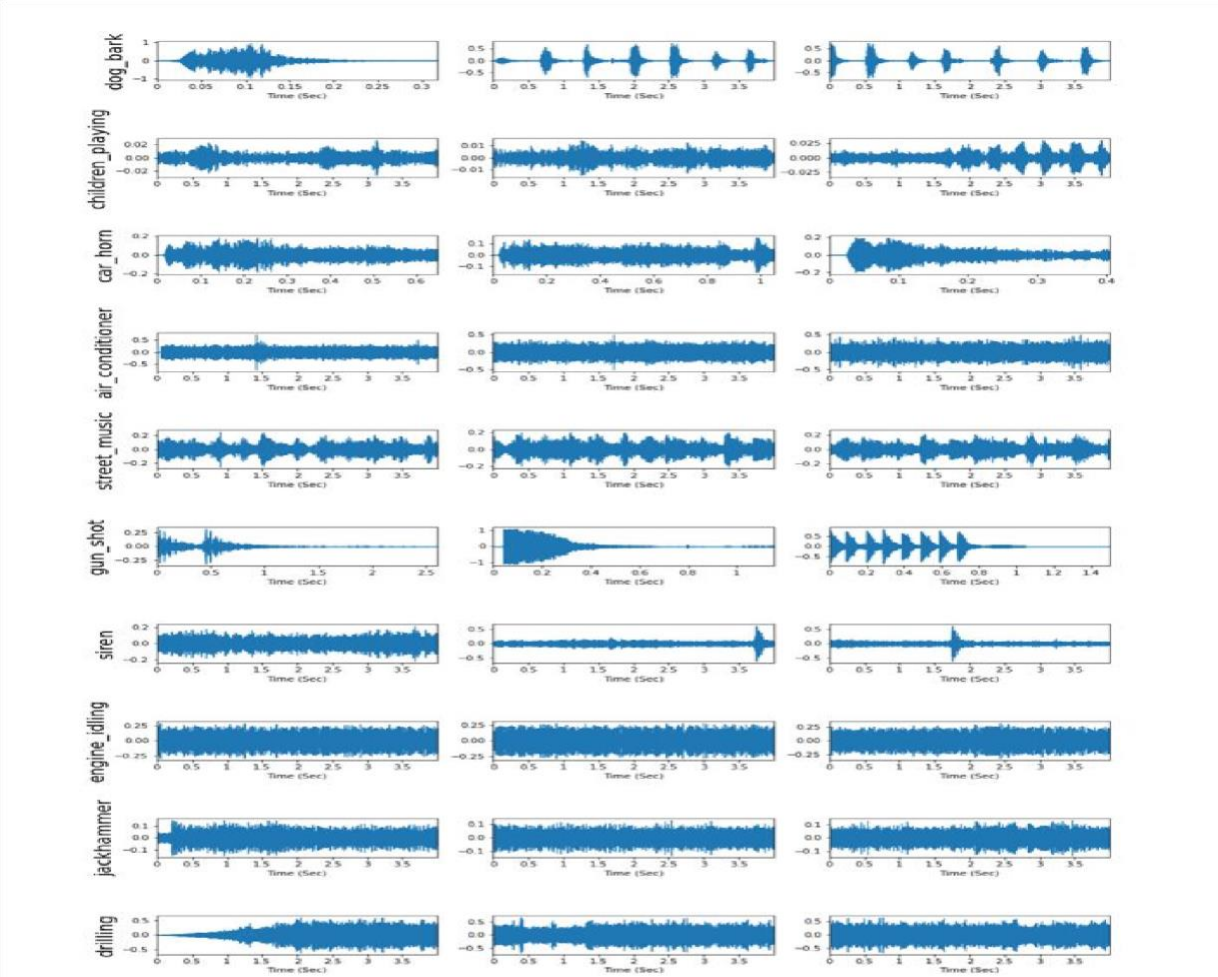
Audio Preprocessing: We iterated through each of the audio sample files and extracted the number of audio channels, sample rate, and bit-depth using Librosa package in Python. The number of audio channels was converted to 1 by merging signals, the sample rate was converted to 22.05 kHz and the bit depth was normalized so that the values were in the range of -1 to 1.

Challenges: The dataset is of decent size, the total size of the data in a compressed format is 5.7GB. The audio file properties like the sampling rate, the number of channels, and the bit depth weren't uniform over all the observations and required standardization. Although all the audio clips were clipped at 4s, they still had different durations (the spectrograms extracted had different image resolutions and needed padding). The transfer learning models and CNN models needed to be run on a GPU and at least 16 GB RAM, hence we used Google Colab.

Since we were dealing with audio files we used the *nlpaug* package to use six augmentation techniques: cropping, loudness augmentation, speed shifting, pitch shifting, time shifting and noise injection, giving us a dataset 6x the original size.

Initially we had class imbalance(e.g. siren and gunshot) due to which our CNN models were overfitting, hence we used class weights, so that the model pays equal attention to the minority classes. We also extracted features such as spectral rolloff, centroids, contrast, bandwidth, and zero crossing rate but they didn't add as much value as mel frequency cepstrum coefficients did. The loss and accuracy curves for our final model (Densenet121) shows that there is no significant overfitting

From the waveform below it can be seen that engine_idle, air conditioner and jackhammer can be easily confused. The classes 'jackhammer' and 'engine_idling' created confusion and were the primary reason for misclassifications. The classes 'gunshot', 'air_conditioner' and 'drilling' were also problematic for humans. The other classes like 'dog bark', 'car horn' etc. were easier to identify.



Waveforms: 3 samples of each category

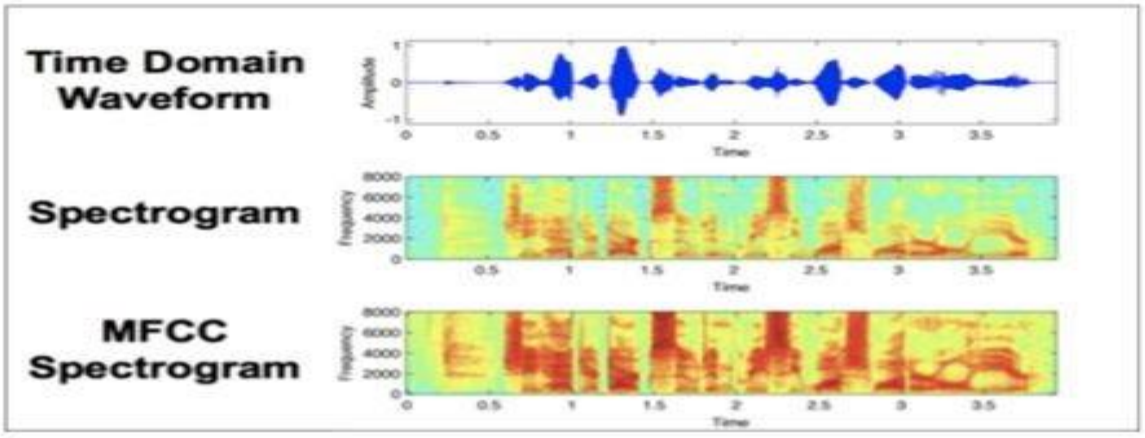
Note: The transfer learning models and CNN models needed to be run on a GPU and at least 16 GB RAM, hence we used Google Colab.

Technical Approach

Technical Inspiration: Observing the recent advancements in the field of image classification where convolutional neural networks are used to classify images with high accuracy and at scale, these techniques can be extended to other domains, such as sound classification.

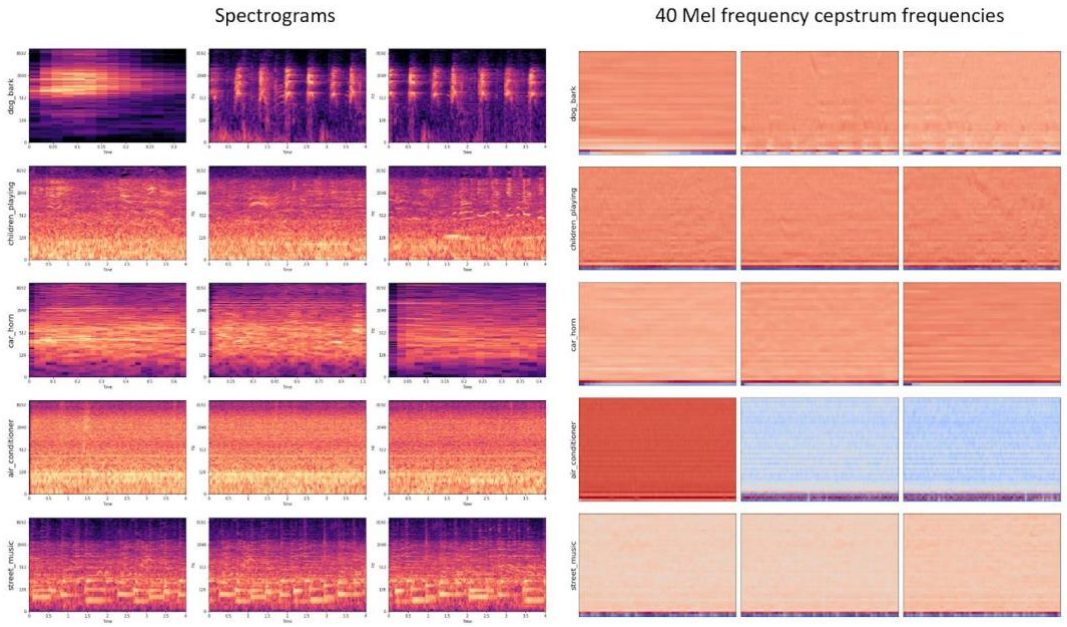
Spectrograms are one of the most popular techniques for visualizing the spectrum of frequencies of a sound and how they vary during a very short period. We used a similar technique known as Mel-Frequency Cepstral Coefficients (MFCC), which has recently become a highly recommended technique in the domain of audio analysis.

The ability of deep convolutional neural networks (CNN) to learn discriminative spectro-temporal patterns makes them well suited to environmental sound classification compared to other approaches. CNNs by design do well in learning patterns inside images which turns out very helpful in this scenario as MFCC spectrograms are very efficient image representations of audio



We trained a deep convolutional neural network on carefully extracted visual representations of each of the audio samples to address the task of environmental sound classification. We then proceeded further by analyzing the audio files and extract attributes like the number of audio channels, sample rate and bit-depth.

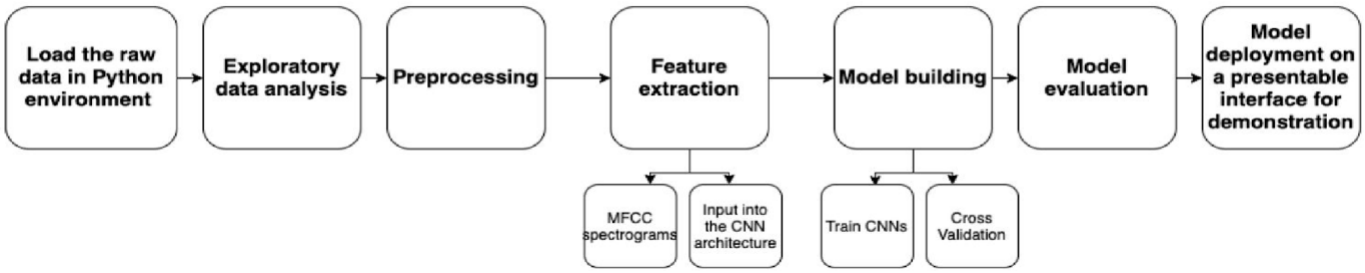
Next, came the key step of preprocessing the data and extracting the MFCC spectrograms. Below you will find the MFCC spectrograms of each of our classes in our data (these are the images we seek to classify):



Finally, we used cross validation to train the CNN and tune its hyperparameters. Our metric of success was: cross validation accuracy.

As a side exploration, we also experimented with transfer learning and other architectures.

The below diagram summarizes our workflow:



What has been done since MVP?

Our MVP was a CNN with the following characteristics:

- 4 Convolutional Layers + 1 fully connected layers
- A num_batch_size of 256
- A Max pool size of 2
- A RELU activation
- A dropout rate of 0.2
- Optimizer: Adam
- This gave us a base accuracy of 83.04%.

We then explored the following directions using our initial MVP as a base:

- Kernel Size: 3,4,5
- Dropout: [0.2, 0.2, 0.5,0.5] (one for each conv layer)
- The final accuracy after these explorations was: 95.63%

Our takeaways from the architecture experimentation process was as follows:

The accuracy increased as kernel size was increased, but peaked at 4. We decided to proceed with 4x4 kernels. As for the dropout, the accuracy decreased by increasing dropout. Thus, we concluded that it was best to leave dropout as 0.2 for all layers (except the last two conv layers, for which we maintained a [0.5, 0.5] dropout.

A more detailed description of results from this first phase of experimentation are described below table

Sl. No.	Model Architecture	Hyperparameters	Model performance	Learnings
1	CNN	4 convolution layers + 1 fully connected layer kernel_size = 2 num_batch_size = 256 maxpool size = 2 activation = relu dropout = 0.2 optimizer = adam [referred as 'original architecture']	CV Accuracy = 83.04%	This is the most basic architecture we tried. Subsequent experimentation was built on this structure
2	CNN (experiment with kernel_size)	Same as above except kernel_size = 3 kernel_size = 4 kernel_size = 5	CV Accuracies, 3 - 90.48% **4 - 92.65%** 5 - 92.27%	The accuracy increased as kernel size was increased, but peaked at 4. We decided to proceed with 4x4 kernels
3	CNN (experiment with dropout)	Same as above with kernel_size = 4 and dropout = [0.2, 0.2, 0.5, 0.5]* (*one for each convolution layer)	CV Accuracy = 90.93%	The accuracy decreased by increasing dropout. Dropout was left at 0.2 for all layers
4	CNN (with augmented MFCC features)	original architecture with kernel_size = 4	Test Accuracy* = 95.63%	This was our best model with basic architecture

We also tried a few completely different architectures as our base model: a VGG16 with transfer learning, a VGG19, a Densenet121, and a Densenet121 with augmented MFCC features. In the end, the densenet121 (with augmented MFCC features) was the winner.

Our results from this second phase of experimentation are summarized in the below table:

Sl. No.	Model Architecture	Hyperparameters	Model performance	Learnings
5	VGG16 (transfer learning)	epochs = 30 batch size = 256	Test Accuracy* = 95.4%	Does not perform better than the basic model (4)
6	VGG19 (transfer learning)	epochs = 30 batch size = 256	Test Accuracy* = 94.5%	Does not perform better than the basic model (4)
7	Densenet121 (transfer learning)	epochs = 30 batch size = 256	Test Accuracy* = 95.28%	Does not perform better than the basic model (4)
8	**Densenet121 (with augmented MFCC features)**	**epochs = 30 batch size = 64**	**Test Accuracy* = 96.69%**	**We selected this network as our best model**

In the end, the final model we choose to submit has the following characteristics:

(1) **Model architecture :** Densenet121 with top layer removed, followed by a global average pooling,dropout layer, dense layer(256 nodes) with relu activation dropout layer and a 10 node dense layer with softmax was used.

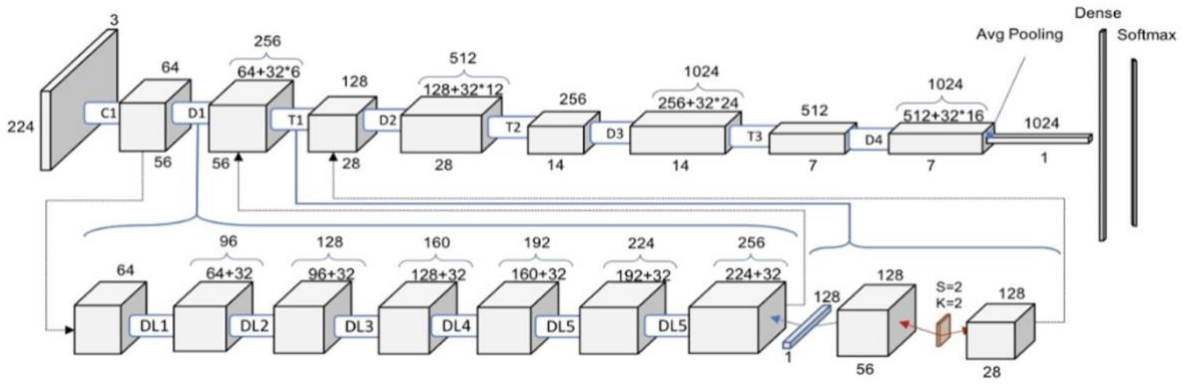
(2) **Model hyperparameters:** Batch size = 64(we tried 128 and 256), Number of epochs = 30(within 5 epochs the model had a validation accuracy of more than 90%)

(3) **Loss -** Categorical cross entropy, Optimizer - Adam, Validation split was 33% on training.

(4) Model performed better than VGG16, VGG19 and MobilenetV2 that also got close results.

(5) **Model accuracy:** Training accuracy: 97.30 , Testing accuracy: 96.70 (6) **Model loss:** Training loss: 0.0752 , Testing loss: 0.1071

A graphical representation of our final model is presented below:



Tricks learned from doing the MVP

We learned the following from our initial MVP:

1. Correcting for the class imbalance by using class_weights significantly improved model performance (from 67% in our MVP to 85% with the most basic network)
2. 4 x 4 kernels are the best suited for this dataset
3. Data augmentation was very helpful in increasing model performance
4. Additional features did not add much to classification accuracy
5. In parallel one of our team members also explored an RNN approach to solving the problem. Unfortunately, that did not perform as well as CNNs. This gave us further confidence in our current approach. We believe the RNNs didn't perform well because MFCCs are time invariant coefficients
6. One important takeaway was that in this problem simpler model architectures performed almost at par with larger networks. You may recall that our initial 4 layer barebones CNN with only the most basic optimizations produced an accuracy of 85% which is not that bad.

Are there alternate ways that you could have approached the problem, e.g. deeper nets vs wider nets? Different activations? Different software/toolset? How would you expect the results to change?

The alternative ways to have approached this problem could be the following :

- RNN
- Masked convolutional Deep Neural Network

- AudioGmentor : A MATLAB tool

In terms of activations in the current architecture we already tried sigmoid ReLU and tanh. Different configurations in terms of depth and width of the network was tried as well. We also tried RNN, but it did not perform as well as CNN

Results

How much did the final project improve upon the MVP?

Our final project improved tremendously upon the MVP. Quantitatively, we experienced testing accuracy gains of about ~13 percentage points.

However, a discussion of limitations is also in order:

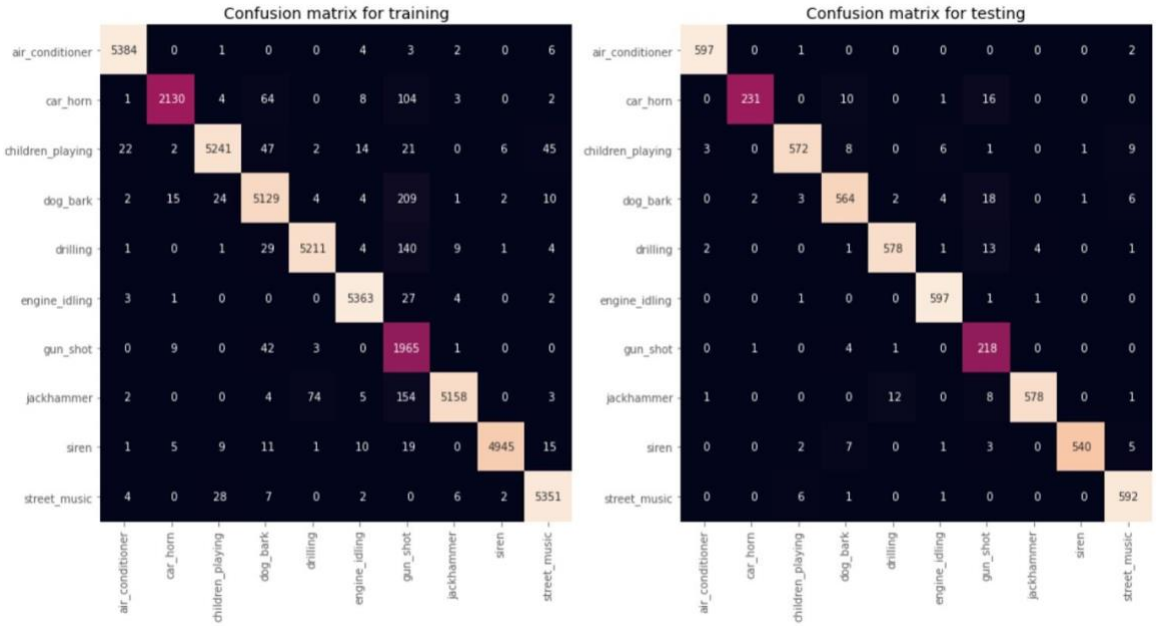
1.Lack of knowledge about what MFCC spectrograms actually mean can result in problems while inspecting the intermediate layer activations of the CNN for diagnosis of signs of poor training.

2.There can be background noise in a few classes of the dataset, which can hamper training and can be difficult to diagnose.

3. The case when the audio sample has more than one class like “engine idling” and “car horn” together, our model will not be able to point out both simultaneously

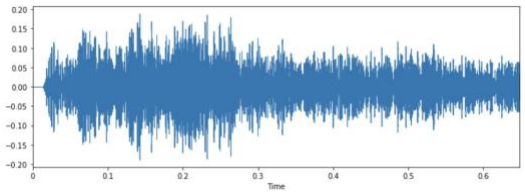
Is there a modified version of your problem that can be more readily solved? E.g. Rather than classifying a particular breed of dog out of a hundred breeds, just classify dog vs cat.

One thing we observed is that there was some confusion between car horn, gunshot, and dog bark. (as can be observed in the below confusion matrix). By removing these classes, we believe that we could get an even higher accuracy (in the order of 98%).

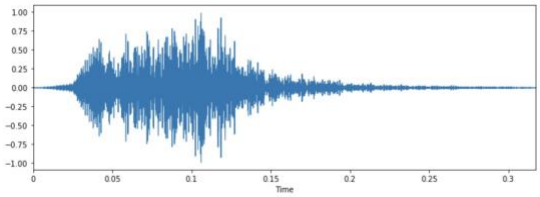


As noted earlier, dog barking, gunshots, and car horn were particularly confused. As can be seen from the below visualizations, each of these categories had similar waveforms with a sudden loud sound

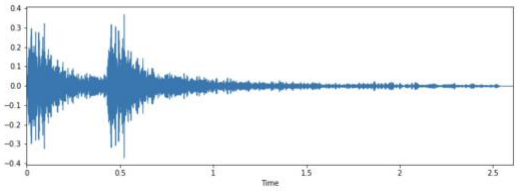
Car Horn



Dog Bark



Gun Shot



If you felt that you didn't have enough data, what were some of the tricks you used to address this issue? If you felt you had enough data, please justify why.

Firstly, we had about 5.9GB of audio data. Given our computational resources, this was more than adequate (in fact slightly smaller would have made things more manageable). However, perhaps by throwing more data into the model (especially from the confused dog barking, car horn, and dog barking classes), we could improve accuracy in those areas.

Were there certain cases where the MVP did better than your final model?

No, there were no cases where the MVP performed better than the final model.

How do you know you've found the "best" model?

The attempt was to match or surpass human accuracy of discerning sound. While the human accuracy was 98% the accuracy our best model could achieve on the test set was 96.7%. This indicates that a superior model is needed in order to make this application worth human use. A more sophisticated model might perform better and yield a higher accuracy like the ones answered in response to a previous question. However since our model did not depict any major signs of overfitting in spite of depicting high accuracy, we were satisfied with our solution.

References

- **List any papers, books or documents that you found helpful**

•This dataset is available on https://urbansounddataset.weebly.com/urbansound8k.html

•Hand crafted features extracted and CNN-based feature extractor gives a testing accuracy of 73.1 +/- 6.2 (R ecognition of urban sound events using deep context-aware feature extractors and handcrafted features)

•Using CNNs on handcrafted features gave a testing accuracy of 87.8% and using Boosting it gave a testing accuracy of 91.9%(What's That Sounds? Machine Learning for Urban Sound Classification).

•VGG with CNN feature vectors of length ≈ 3500 gave a testing accuracy of around 72.86%

(https://arxiv.org/pdf/1805.00237.pd f) **Papers:**

•http://www.justinsalamon.com/uploads/4/3/9/4/4394963/salamon_urbansound_acmmm14.pdf

•https://arxiv.org/abs/1608.04363

•https://www.ijrte.org/wpcontent/uploads/papers/v7i5s3/E11900275S19.pdf

References of code:

•Music Feature Extraction in Python

•Audio Classification Using CNN — An Experiment - AI Graduate

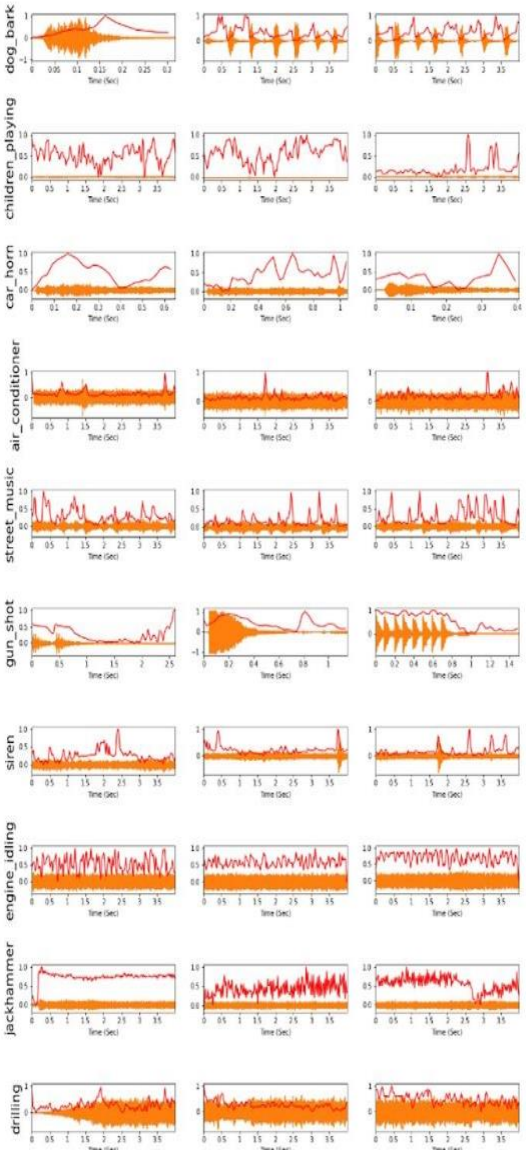
· **Also list things that you were unable to find references on – we'll try to collect these together and improve for the next class**

N/A

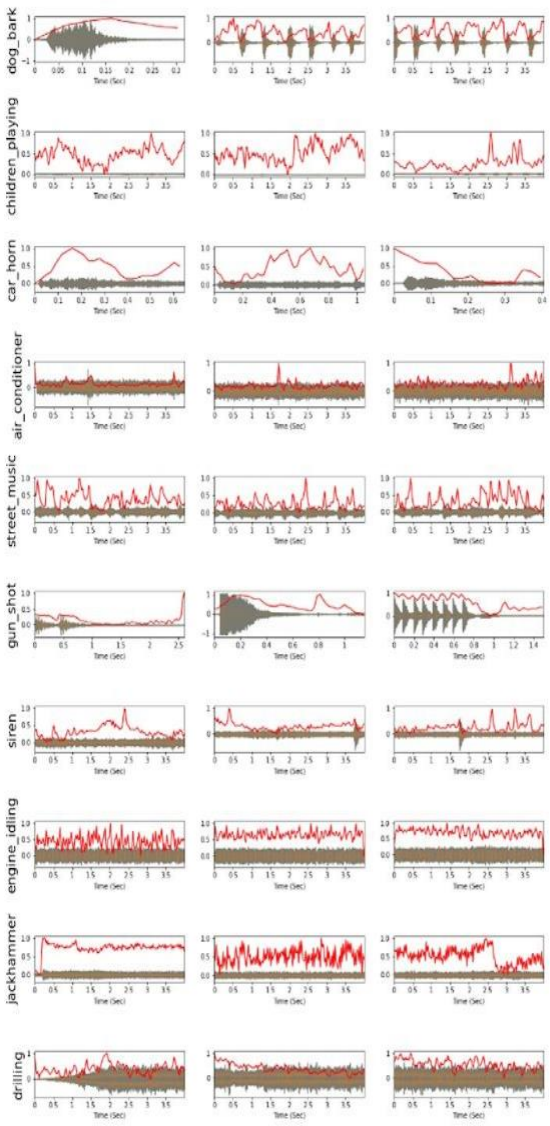
Appendix

Additional Features Explored

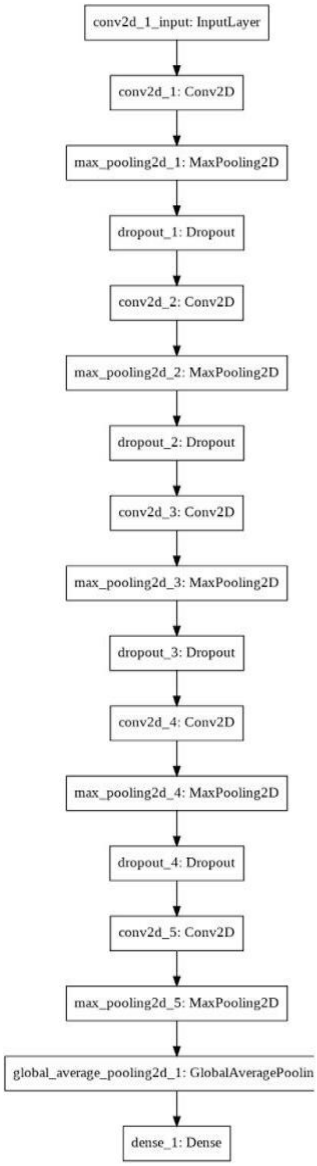
Spectral centroids

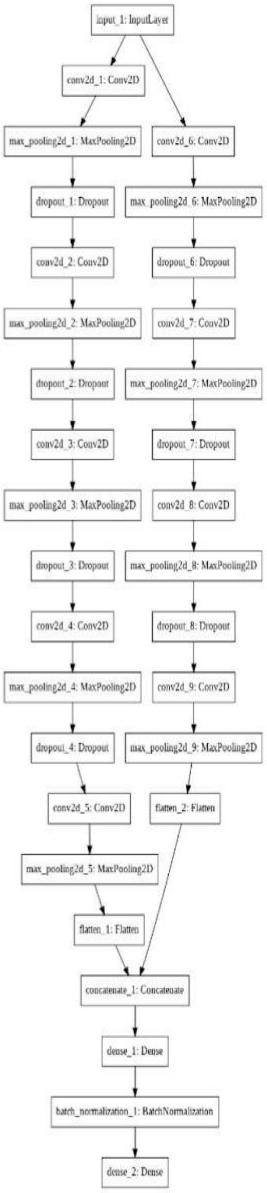


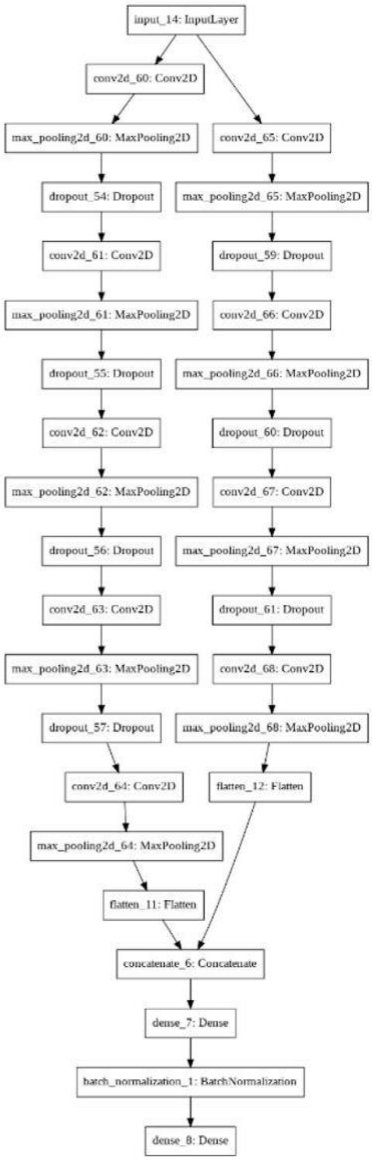
Spectral rolloff



Other Model architectures Explored


conv2d_1_input: InputLayer
conv2d_1: Conv2D
max_pooling2d_1: MaxPooling2D
dropout_1: Dropout
conv2d_2: Conv2D
max_pooling2d_2: MaxPooling2D
dropout_2: Dropout
conv2d_3: Conv2D
max_pooling2d_3: MaxPooling2D
dropout_3: Dropout
conv2d_4: Conv2D
max_pooling2d_4: MaxPooling2D
dropout_4: Dropout
conv2d_5: Conv2D
max_pooling2d_5: MaxPooling2D
global_average_pooling2d_1: GlobalAveragePoolin
dense_1: Dense


input_1: InputLayer
conv2d_1: Conv2D
max_pooling2d_1: MaxPooling2D
conv2d_6: Conv2D
dropout_1: Dropout
max_pooling2d_6: MaxPooling2D
conv2d_2: Conv2D
dropout_6: Dropout
max_pooling2d_2: MaxPooling2D
conv2d_7: Conv2D
dropout_2: Dropout
max_pooling2d_7: MaxPooling2D
conv2d_3: Conv2D
dropout_7: Dropout
max_pooling2d_3: MaxPooling2D
conv2d_8: Conv2D
dropout_3: Dropout
max_pooling2d_8: MaxPooling2D
conv2d_4: Conv2D
dropout_8: Dropout
max_pooling2d_4: MaxPooling2D
conv2d_9: Conv2D
dropout_4: Dropout
max_pooling2d_9: MaxPooling2D
conv2d_5: Conv2D
flatten_2: Flatten
max_pooling2d_5: MaxPooling2D
flatten_1: Flatten
concatenate_1: Concatenate
dense_1: Dense
batch_normalization_1: BatchNormalization
dense_2: Dense


input_14: InputLayer
conv2d_60: Conv2D
max_pooling2d_60: MaxPooling2D
conv2d_65: Conv2D
dropout_54: Dropout
max_pooling2d_65: MaxPooling2D
conv2d_61: Conv2D
dropout_59: Dropout
max_pooling2d_61: MaxPooling2D
conv2d_66: Conv2D
dropout_55: Dropout
max_pooling2d_66: MaxPooling2D
conv2d_62: Conv2D
dropout_60: Dropout
max_pooling2d_62: MaxPooling2D
conv2d_67: Conv2D
dropout_56: Dropout
max_pooling2d_67: MaxPooling2D
conv2d_63: Conv2D
dropout_61: Dropout
max_pooling2d_63: MaxPooling2D
conv2d_68: Conv2D
dropout_57: Dropout
max_pooling2d_68: MaxPooling2D
conv2d_64: Conv2D
flatten_12: Flatten
max_pooling2d_64: MaxPooling2D
flatten_11: Flatten
concatenate_6: Concatenate
dense_7: Dense
batch_normalization_1: BatchNormalization
dense_8: Dense