Practical I

Ben Elliott, Hrólfur Eyólfsson, Can Yeşildere

1 Part A: Feature Engineering, Baseline Models

1.1 Approach

For the baseline models, we trained two simple logistic models using **sklearn**. Since the classification problem concerns 10 classes, the algorithm was using gradient descent to maximize the softmax function. In practice, the algorithm assigns a probability of a data point belonging to each of the 10 classes:

$$\mathbf{P}(y = j|\mathbf{x}) = \frac{e^{x^T w_j}}{\sum_{k=0}^{9} e^{x^T w_k}}$$

and the prediction is chosen as the class which has the highest probability.

We had two distinct representations of the sound data: the amplitudes and the frequencies. Both databases are very high dimensional, with the amplitude data having 44100 dimensions and the Mel spectrogram data having 11136 dimensions. Since we only had 5553 observations in the training set, however, we know that the algorithm can perfectly predict any point in the training set, which is conducive to over-fitting. Even though we might think of the amplitude data as being more expressive due to it having more dimensions, this might also be more conducive to overfitting. The frequency data on the other hand is flattened, which means that some of the data structure might have been lost.

One of the executive decisions we made was to limit the number of iterations for the second baseline model, trained on the Mel spectrogram data to only 1000 iterations such that it could converge in a reasonable time.

1.2 Results

Table 1 displays the results. Baseline 1 refers to the simple logistic model trained on the amplitude data, whereas Baseline 2 refers to the model trained on the Mel spectrogram data. Baseline 1 had an overall accuracy of 17.9 percent, with a per class accuracy of 14.8 percent. Model 1 was unable to predict any observations belonging to class 1 (car horn) accurately, whereas it did best with class 2 (children playing). Baseline 2 had an overall accuracy of 36.8 percent, with a per class accuracy of 40.7 percent. Model 2 was the worst at predicting belonging to class 3 (dog bark) accurately (20% accuracy), whereas it did best with class 6 (gun shot, 73.3% accuracy), .

1.3 Discussion

On both overall accuracy and per class accuracy, the model trained on the Mel spectrogram data did better. We think there were a number of factors influencing this results. First, due to the fact that the amplitude data had almost 4x more dimensions, it was conducive to more overfitting to the noise on the training set. Second, frequencies are more distinctive when it comes to identifying sounds than amplitudes. A gun shot and a siren are both loud in terms of amplitude, but have very

Model	Overall Acc (%)	Per Class Acc (%)	Class with Worst Acc	Acc for Worst Class (%)	Class with Best Acc	Acc for Best Class (%)
Baseline 1	17.89	14.77	1	0.0	2	34.45
Baseline 2	36.82	40.70	3	20.09	6	73.33
kNN - Baseline	26.67	30.41	9	6.67	7	63.14
kNN - Tuned	37.55	31.69	6	0.0	8	65.25
Model 3	7.49	9.82	...			
RFC - Baseline	37.55	33.65	6	6.67	8	63.56
RFC - Best	41.28	36.17	6	6.67	2	64.21
RFC - Balanced	39.6	34.62	6	6.67	8	63.14

Table 1: Results for all models

different pitches. As a result, despite having more dimensions, the amplitude data may contain less information, i.e. meaningful variance.

The fact that the Mel spectrogram data had a better per class accuracy than overall accuracy is surprising, but it shows that distinct frequencies may allow for capturing niche classes. Class 6 was actually the least common class in the dataset, but Model 2 classified it the best.

2 Part B: More Modeling

For part B, we focused on two types of non-linear models: kNN and random forests. In order to simplify the computational complexity of the procedure, we reduced the number of dimensions in the analysis using principal component analysis (PCA). The PCA function creates orthonormal components that are formed from a linear combination of the pre-existing dimensions in the dataset that aim to capture the most variance that exists in the data. Each component is calculated such that it captures the most amount of variance and is independent from the previous components generated. For our implementation, both Mel spectrogram and amplitude data were merged into the same dataframe, which had a combined 55236 dimensions. We used **sklearn**'s **PCA** function and identified that the first 693 principal components captured 99% of the variance in the training set. We applied the PCA weightings trained on the training set to the test set for this section.

2.1 kNN

The k nearest neighbour classifier is a non-liner model that makes predictions according to what classes the k observations that are most similar to the new observation belong to. Mathematically, it uses a distance function (Euclidean distance in this case) and finds the k observations in the training dataset that are the least distant to it. It then assigns a probability to each class based on how many out of the k nearest neighbours belong to each class. The class with the highest probability is the algorithm's prediction.

2.1.1 Baseline

For the baseline model, we used 700 principal components (99% variance) and chose 5 as the number of neighbours. Row 3 in Table 1 displays the results of the model. It had an overall accuracy of

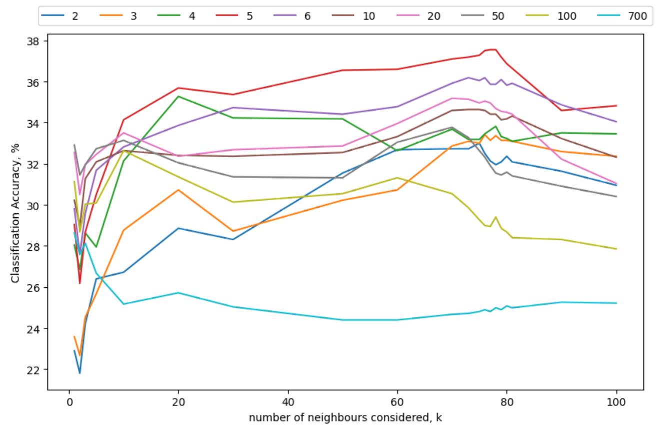


Figure 1: Results for kNN tuning; number of neighbours (x-axis) and number of principal components (color)

26.7 percent, with a per class accuracy of 30.4 percent. It had the lowest accuracy for class 9 (street music), with only 6.7 percent accuracy and the highest accuracy for class 7 (jackhammer).

2.1.2 Tuning

We identified two hyperparameters for tuning kNN: the number of neighbours (k) and the number of principal components (d). The number of principal component is an important hyperparameter because not every component contains the same amount of information, or relevant information. While 693 components capture 99% of the variance in the data, not all of that is relevant for the purpose of classification. The model that performed the best used only 5 principal components, but considered 77 neighbours. This model had an overall accuracy rate of 37.55 percent and a per class accuracy rate of 31.7 percent. While the overall accuracy rate is improved significantly, wee see that this model sacrificed accuracy for some classes over others. For example, we see that the the tuned model couldn't guess any instance of class 6 accurately.

2.2 Random Forest

Random forest is a non-linear model that ensembles the best predictors from decision trees. It is in essence a modified decision tree. Decision trees work by segmenting the space into areas for each class. They do this by making a sequence of decisions that split their relative part of the space into two. Each decision creates new nodes, each node is followed by a decision. The tree maximizes the score of its decision by choosing a dimension and value (i.e. a hyperplane) where the purity (read: homogeneity) in each new created area is maximal.

Decision trees tend to overfit the data, so random forests aggregate the decisions of several decision trees, giving each tree a vote, essentially.

2.2.1 Baseline

The baseline random forest classifier has 100 estimators and no maximum depth, which are the only two parameters we'll be tampering with. Row 4 in Table 1 displays the results of the model

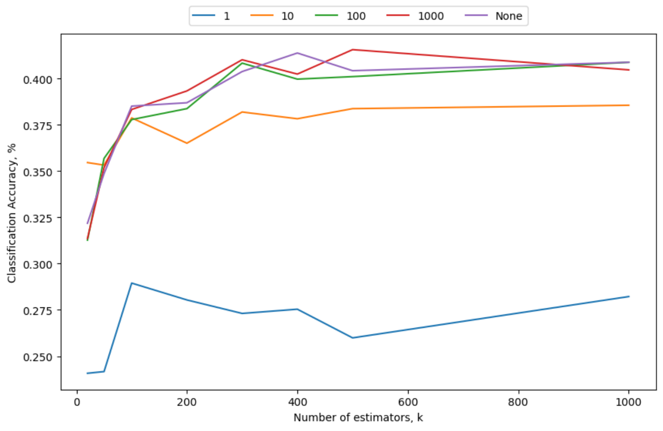


Figure 2: Results for RFC tuning; number of trees (x-axis) and maximal depth (color)

with an overall accuracy of 37.55%.

2.2.2 Tuning

The parameters we tampered with were the number of estimators (k) and the maximum depth of the tree (d). Tampering with the maximum depth further insures us against overfitting, beyond the value that random forest classifiers have over normal decision trees. Changing the number of estimators represents the number of trees in the forest, the more trees, the better (and slower) the model should be at fitting to its data. It is about balancing the two, as well as dealing with the bias-variance tradeoff.

We found, as expected that increasing the number of trees generally improved the accuracy, but after a certain point (for the deeper forests) the number of trees hurt the test accuracy, presumably introducing too much bias into our model. The deeper forests performed better, but adding little, marginally, after a certain point. We found that the best forest had 500 trees and no maximum depth. This had an overall accuracy of 41.28% Almost equally good, though, the forest with 300 trees and 100 depth, with an overall accuracy of 39.6%. That tree took a much shorter time to run.

2.3 Discussion

We see that non-linear classifiers have a better time capturing the complex decision boundaries in this problem, but require tuning. The simple kNN model doesn't do better than baseline model II, but tuning can improve performance. Even then, it achieves this without improving the per class accuracy, suggesting that kNN is particularly susceptible to class imbalances.

The random forest classifiers outperform the baseline logistic models, and greatly outperform the kNN models. While we did try optimizing for training time, it must be said that the RFC model takes very little time to run, in the grand scheme of things (e.g. compared to the CNN models in our appendix). The RFC, like the kNN is very susceptible to class imbalances, however, just like the kNN model.

3 Final Write-up and Reflections

In this project, we started with two distinct types of representation: amplitude and frequencies. We saw in part A, that frequency was a more useful metric for our predictions. This fits with our priors based on our knowledge of the domain. Namely, these sounds come from loud noises in a city, which are expected to all have high amplitudes, but different frequencies. Yet, to fully exploit the richness of the data, we combined all our data together for the rest of the project and reduced the dimensions of the data using PCA, so as to simplify the computational complexity of our models. As a result, we were able to create kNN and random forest models that performed better than the logistic regression, while not being too spatially complex.

What was surprising is that the logistic regression did a decent job compared to the non-linear models we used. kNN, in particular, struggled with the class imbalance. This makes sense, since there are were so few instances of certain classes that it was unlikely for them to constitute a plurality in most of the neighbours. The random forest models, however, performed better. While decision trees tend to overfit, the ensembling that is built into the random forests allows for a reduction in this bias.

Our evaluation metric had a big impact on the types of models we produced, especially given the class imbalance in the data. Most models, especially kNN, prioritized classifying classes that were larger. We believe this is the biggest point of criticism for our model, and we would have liked to prioritise other metrics. One potential idea is to design a loss function that maximizes the accuracy for the class with the lowest accuracy or minimizes the variance between the accuracy rates for different classes.

Our models make sense for the way in which the problem was framed, that is, choosing the label that fits the observation the best out of the 10 options. In real life, however, there is limited applicability for this model, given that there are a lot more types of noises that don't fit these 10 categories. We should include a separate category for other or incoroprate a metric of uncertainty or confidence for our classifications.

There are a number of ethical considerations for the use of this model. In terms of sourcing, we are unsure how exactly the sound of children playing were acquired. In terms of labeling, it is unclear who was subjected to the sound of sirens and gun shots, both of which might be highly triggering. Furthermore, since the motivating question behind this dataset was sound complaints for New York City, we are unsure how this would be used. We can't take on legal responsibility with a model that is just accurate 40 percent of the time!

4 Optional Exploration, Part C: CNN

4.1 Approach

We chose to implement a CNN, since this seemed both challenging and interesting. CNNs work very similarly to standard neural networks, but include convolution layers in order to capture local features of the input image. These layers have the effect of making the CNN location invariant, which is to say that, if the network has learned that a certain collection of pixels corresponds to *spaceship*, this collection of pixels can be located in any part of the image. Mathematically, the convolution layers accomplish this by looking over the input image with a kernel whose size is a hyperparameter. This kernel examines local features of the image by multiplying its elements with the input values and then summing the resulting matrix. This is done for the whole image. Because CNNs are specialized to work in computer vision settings, this is an important property. Then, the spectrogram data provided for this model can be treated as images and passed through a CNN.

During the process of building our model, we tried several different architectures to determine which was best. In order to better understand what the standard models for spectrogram data are, we read a few papers and journal articles which pointed us in the direction of incorporating two convolution layers followed by two fully connected layers, with some additional pooling and dropout layers between. While modeling, we took this basic architecture and played around with its depth and width, along with key hyperparameters like learning rate to find the best model for our data. In all, we tried three architectures.

We began with a fairly shallow but extremely wide model given by

```
(conv1): Conv2d(1, 32, kernel_size=(10, 10), stride=(1, 1))
(conv2): Conv2d(32, 64, kernel_size=(5, 5), stride=(1, 1))
(dropout25): Dropout(p=0.25, inplace=False)
(dropout5): Dropout(p=0.5, inplace=False)
(pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
(fc1): Linear(in_features=134976, out_features=128, bias=True)
(fc2): Linear(in_features=128, out_features=10, bias=True)
```

for 20 epochs and with a learning rate of 0.001. We perform `dropout25` on the MaxPooled output of `conv2`, and `dropout5` on the ouput of `fc2`. This architecture is very similar to a CNN given in course tutorials, but we found it in a Medium article claiming to achieve 97% test accuract on audio spectrograms. However, we've increased our kernel size from 3×3 to 10×10 and then 5×5 since our images have significantly more pixels in them. We'll discuss this model's performance in the Results section, but we note here that the training process was extremely slow. Both to try to tune our model and to be able to do so quickly, we chose to test a deeper but skinnier model, which follows:

```
(conv1): Conv2d(1, 2, kernel_size=(10, 10), stride=(1, 1))
(conv2): Conv2d(2, 6, kernel_size=(5, 5), stride=(1, 1))
(conv3): Conv2d(6, 10, kernel_size=(5, 5), stride=(1, 1))
(pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
(fc1): Linear(in_features=660, out_features=512, bias=True)
(fc2): Linear(in_features=512, out_features=256, bias=True)
(fc3): Linear(in_features=256, out_features=128, bias=True)
```

```
  (fc4): Linear(in_features=128, out_features=64, bias=True)
  (fc5): Linear(in_features=64, out_features=10, bias=True)
```

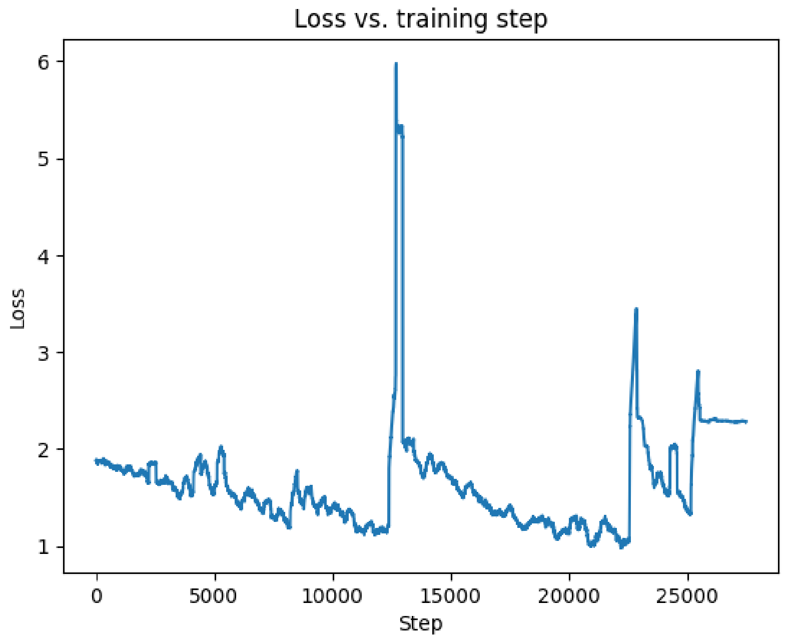
We perform MaxPool on the final convolution layer. This network ran much faster, but was unsatisfactory; the model settled on the strategy of predicting `street_music` every time, indicated that we needed to add Dropout layers. Dropout layers randomly and independently zero weight elements in the layer they're applied to according to a given probability. We hoped that this would force the model out of any local optimums which predict only one category. This model's architecture follows:

```
  (conv1): Conv2d(1, 2, kernel_size=(10, 10), stride=(1, 1))
  (conv2): Conv2d(2, 6, kernel_size=(5, 5), stride=(1, 1))
  (conv3): Conv2d(6, 10, kernel_size=(5, 5), stride=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (dropout1): Dropout(p=0.1, inplace=False)
  (dropout5): Dropout(p=0.5, inplace=False)
  (fc1): Linear(in_features=660, out_features=512, bias=True)
  (fc2): Linear(in_features=512, out_features=256, bias=True)
  (fc3): Linear(in_features=256, out_features=128, bias=True)
  (fc4): Linear(in_features=128, out_features=64, bias=True)
  (fc5): Linear(in_features=64, out_features=10, bias=True)
```

Where we perform `dropout1` on every convolution layer and `dropout5` on every fully-connected layer.

4.2 Results

Our main metrics for evaluation are Cross Entropy Loss (which the model optimizes on) and test accuracy. The first model we ran produced the following trace plot:

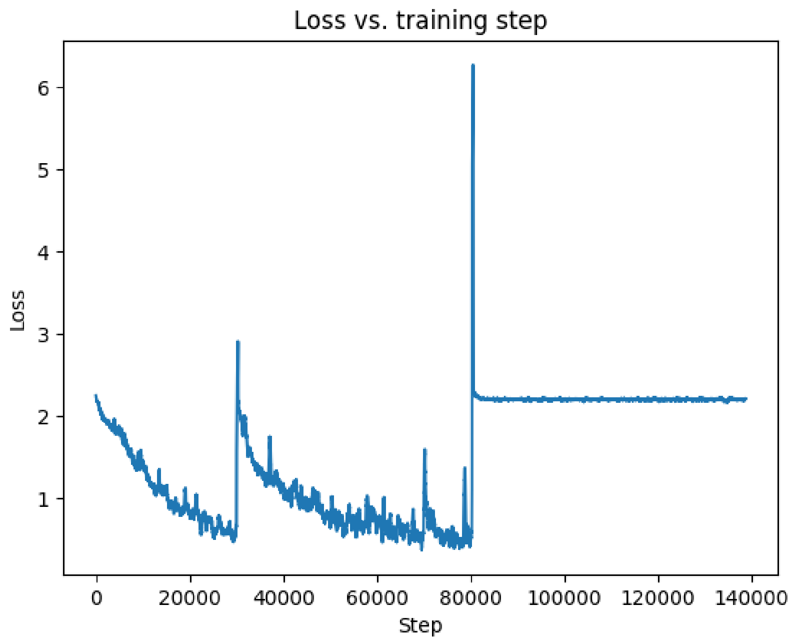


It looks like our learning rate of 0.001 is too large, as the loss varies greatly. We take this into account for our next model. The first model's accuracies are listed below.

```
Accuracy of the network on the 10000 test images: 14 %
Accuracy for class: air_conditioner is 76.0 %
Accuracy for class: car_horn is 0.0 %
Accuracy for class: children_playing is 24.7 %
Accuracy for class: dog_bark is 0.0 %
Accuracy for class: drilling is 3.0 %
Accuracy for class: engine_idling is 6.1 %
Accuracy for class: gun_shot is 0.0 %
Accuracy for class: jackhammer is 1.3 %
Accuracy for class: siren is 0.0 %
Accuracy for class: street_music is 0.0 %
```

14% is a very weak accuracy. Moreover, we see that this model never correctly predicts certain classes like `siren`, `dog_bark`, and `gun_shot`. We try to improve this model by making it deeper and skinnier.

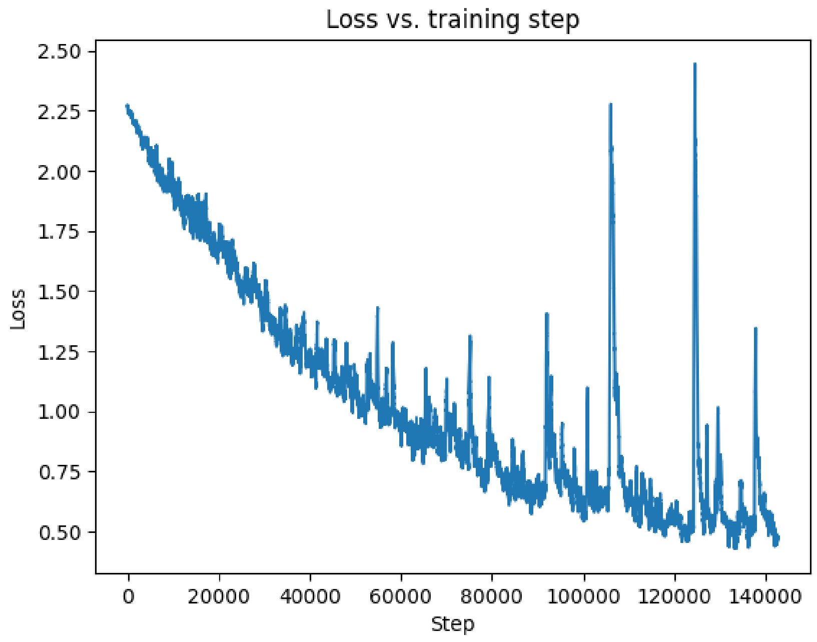
Our second model runs much faster since we pruned the dimensionality of our layers down; this is a welcome improvement if only for our sanity. As noted above, we attempted to decrease the learning rate from 0.001 to 0.0001, but it seems that reducing the width of our architecture made learning somewhat slower, so we kept the learning rate at 0.001. The model, however, performs rather poorly. It's trace plot follows:



When we ran this model for 20 epcohs, we realized that it had not converged during training, so we increased to 100 epochs. We see that the model was training pretty well in the early epochs, but the large spikes in loss signal that perhaps the learning parameter is too high. Moreover, the second model found a local optimum which it could not break through at around step 80000. At this stage, the CNN only predicts `street_music`, shown in the accuracy readout below. This is clearly not a smart prediction, and we introduce dropout layers in our next iteration of the model in order to force the model out of loss-valleys that lead to one-category predictions.

```
Predicted:  street_music street_music street_music street_music
Accuracy of the network on the 10000 test images: 13 %
Accuracy for class: air_conditioner is 0.0 %
Accuracy for class: car_horn is 0.0 %
Accuracy for class: children_playing is 0.0 %
Accuracy for class: dog_bark is 0.0 %
Accuracy for class: drilling is 0.0 %
Accuracy for class: engine_idling is 0.0 %
Accuracy for class: gun_shot is 0.0 %
Accuracy for class: jackhammer is 0.0 %
Accuracy for class: siren is 0.0 %
Accuracy for class: street_music is 100.0 %
```

Our final model uses the same architecture as the second, but adds Dropout layers throughout, as discussed above. The trace plot for this model's training follows, and afterwards the accuracy across classes.



This model performs quite well, though the trace plot still looks jagged. Then, we might be able to improve the model if we decrease the learning rate, but this seems unlikely since we ran just over 100 epochs to get the model to this point; if we decrease the learning rate much more, we'll likely have to train the model much longer, which is cumbersome and inefficient. Rather, the jaggedness of this trace plot suggests that the loss landscape is complicated. The accuracies across classes for this model follows:

```
Accuracy of the network on the 10000 test images: 50 %
Accuracy for class: air_conditioner is 24.0 %
Accuracy for class: car_horn is 20.5 %
Accuracy for class: children_playing is 60.9 %
Accuracy for class: dog_bark is 49.3 %
Accuracy for class: drilling is 41.3 %
Accuracy for class: engine_idling is 60.6 %
Accuracy for class: gun_shot is 60.0 %
Accuracy for class: jackhammer is 50.8 %
Accuracy for class: siren is 51.7 %
Accuracy for class: street_music is 67.7 %
```

4.3 Discussion

Our initial approach was based on an article we read. However, this architecture was too computationally expensive and did not fit our data well. To improve this model and our ability to train it, we added layers but decreased the width of each layer. This resulted in a much faster training process, but also in a bad model which predicted one class every time. To move our model away from this stasis, we introduced random dropout layers, which kept our model out of suboptimal loss valleys and resulted in 50% test accuracy, which we're quite happy with.

Overall, our CNN performed much better than the other models we tried. Previously, the best performing model was a random forest that achieved 41% test accuracy (see Figure 1). Moreover, the CNN's test accuracies by class varied much less than the random forest, whose worst class accuracy was 6%, much like the second CNN model. We want our model to be able to discriminate more finely between classes, and even though the final CNN's worst class (`car_horn`) is only correctly predicted 20% of the time, this is a marked improvement.