Open Set Audio Classification Problem

Yeh Swee Khim, Joey Yeo, Guo Zi Wei, and Leong EnYi

Information Systems Technology and Design, Singapore University of Technology and Design

13th December 2020

1 Introduction

Following the success of the application of machine learning on image classification, a natural extension is the problem of audio classification. This field is well studied and borrows ideas from the image classification problem, with current state-of-the-art models mainly being convolutional neural networks (CNNs) using the Log-Melspectrogram representation of audio data [10]. However, much of current research focuses on the closed set recognition problem (CSR) where the test dataset includes the same set of classes as the training dataset. Given an audio sample, the model will always classify it according to one of the classes it has seen before, regardless of the actual class of the sound. This is particularly detrimental when implementing models in real-life applications where the class of the input data is not controlled, and the availability of training data is limited. A better solution would be to have the model be able to identify data from unseen classes and not produce a classification. This is known as the open set recognition (OSR) problem [2]. In an OSR task, a balance must be found between correctly classifying data belonging to the known classes but also rejecting data from the unknown class. The main difficulty is that the model will not be exposed to any of the unknown classes prior to training but will need to be able to identify these data points during testing. In particular, for our paper we assume that only the positive data is available during training time, and that each class in the dataset is equally important to be accurately classified. In this paper, we propose a range of approaches to attempt to solve the OSR problem in the context of audio classification. We first describe our chosen dataset and data split as well as the data representation that we will be using to train our models. We then choose a baseline model that solves the audio classification CSR problem for us to extend upon for the OSR problem. Next, we describe the feature extraction methods we used to augment our models and discuss the evaluation metrics that will be used to assess model performance. Lastly, we describe the various approaches taken to tackle the OSR problem and discuss the results obtained, with a final summary of our learnings.

All code for this paper can be found at: `https://github.com/YehSweeKhim/Open-Set-Audio-Classification`

2 Dataset and Collection

The dataset we are using is the UrbanSound8K dataset [12] which has seen wide use in existing literature for evaluating models that aim to solve the CSR problem for audio classification. However, we are not aware of any existing work that uses this dataset in the context of OSR.

UrbanSound8K contains audio clips of less than 4 seconds belonging to 10 classes of environmental sounds. The class names together with their labels and distributions are shown in Figure 1. Most of the classes are equally represented, with the exception of classes 1 and 6 which have less samples.

classID	Name	Distribution
0	air_conditioner	0.114521
1	car_horn	0.049130
2	children_playing	0.114521
3	dog_bark	0.114521
4	drilling	0.114521
5	engine_idling	0.114521
6	gun_shot	0.042831
7	jackhammer	0.114521
8	siren	0.106390
9	street_music	0.114521

Figure 1: UrbanSound8K Class Distribution

2.1 Data Split

To prepare the dataset for use in the OSR problem, we select 5 classes from above to be our known classes that we will use to train our models. The remaining 5 classes will be our unknown classes. We will be selecting classes 0 - 4 to be the known classes and classes 5 - 9 to be the unknown classes from the UrbanSound8K dataset. Splitting the dataset in this way ensures that both underrepresented classes are separated and the final distribution for both known and unknown classes would be approximately equal such that the data is not skewed. The distribution of the classes after the split are shown in Figure 2.

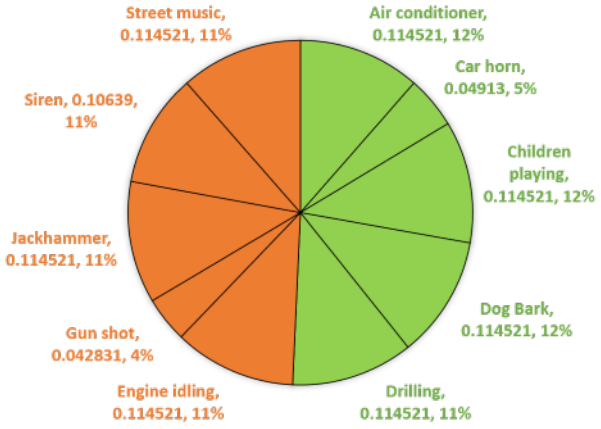

Figure 2: UrbanSound8K Class Distribution after split

The complexity of an OSR task can be quantified by using the openness factor, O^* that measures the relationship between the number of classes seen during training compared to the testing phase. The openness, O^*, can be calculated using Equation 1 [7] where T_{TR} corresponds to the number of classes used during training and T_{TE} corresponds to the number of classes used in the testing phase. Openness values are bounded to the range $0 \leq O^* < 1$. From this equation the openness of our dataset can be calculated to be $O^* = 0.1835$ due to it having 5 known and 5 unknown classes.

$$O^* = 1 - \sqrt{\frac{2 \times T_{TR}}{T_{TR} + T_{TE}}} \tag{1}$$

We further split our data into training, validation, and testing sets. From the 5 known classes, 80% of the data will be used for training and 20% of the data will be used for testing. We further split the training

data 80-20 into training and validation sets. To represent the OSR problem, the test set will be augmented with 20% of the data from the unknown classes. A visual representation of the dataset split can be found in Figure 3.

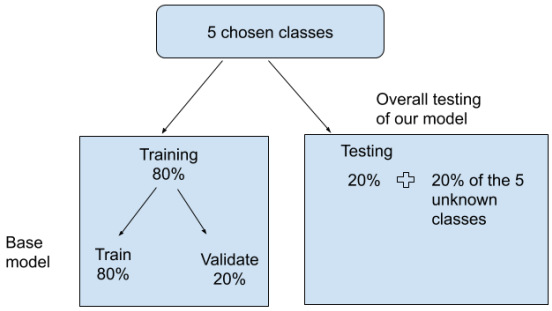

Figure 3: Train-Test Split

3 Data Preprocessing

For our data representation, we have chosen the Log Mel Spectrogram. The Log Mel Spectrogram is one of the most commonly used data representations in the field of audio classification [10]. The Mel scale is constructed in a way which scales frequencies in the range that is audible and distinguishable by human ears [1], while taking the log of the audio amplitude puts it in terms of decibels. Since audio classes are mainly determined by how they sound to human ears, this representation accurately captures the key features of audio data.

To generate the spectrograms, we made use of a script from [10] which can be found at [8]. The script makes use of Librosa [9]. The spectrograms generated have 3 channels, with each channel having a different hop length and window size [10]. This is to better fit the input space of classic image classification models which take in 3-channel RGB images. This will make it easier to apply transfer learning. An example of a generated Log Mel Spectrogram is given in Figure 4. Each 3-channel spectrogram has shape (250, 128, 3).

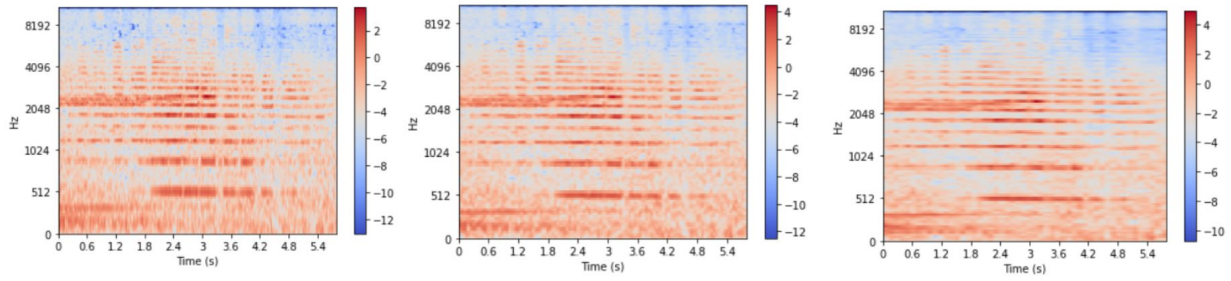

Figure 4: Mel spectrograms of car horn in 3 channels

4 Base Model

We chose a model with state-of-the-art performance on the UrbanSound8K dataset for the CSR problem to be our base model. We would then expand on this model to solve the OSR problem. We chose the DenseNet-201 Convolutional Neural Network (CNN) [6] pre-trained on ImageNet based on its performance in [10].

We fine-tuned the model on our training set for 70 epochs with early stopping and a patience of 5 based on the validation loss. The Adam optimizer was used and the loss function was sparse categorical cross entropy. The weights that gave the highest validation accuracy across all epochs were chosen as the final weights. The model accuracy on the known portion of our test set is shown in Table 1.

Metric	Value
Overall Accuracy	0.9458
Class 0 Accuracy	0.9856
Class 1 Accuracy	0.9730
Class 2 Accuracy	0.9360
Class 3 Accuracy	0.8947
Class 4 Accuracy	0.9651

Table 1: Base Model Accuracy

5 Feature Extraction

Feature extraction is an important step to reduce the dimensionality of input data by extracting useful information and producing a compact set of features. The Log Mel Spectrograms generated have a large number of dimensions (96000). As such in order to reduce complexity and training time for some models we chose 2 feature extraction methods to create additional data representations to augment working directly with the spectrograms.

5.1 Base model as a feature extractor

By freezing the weights in our base model and removing the classification layer, we are able to use it as a feature extractor for our initial spectrograms. The feature vectors are extracted after the final global average pooling layer in our fine-tuned DenseNet-201 [6] and hence have a dimensionality of 1920.

5.2 VGGish as a feature extractor

In contrast to using our base model which was fit on our training set itself as a feature extractor, we chose a more generalized model as our second feature extraction method. In 2017, Google released VGGish [4], a CNN for audio classification pre-trained on their large AudioSet dataset [3]. VGGish generates 128-dimensional embeddings for each 1s of audio data represented as 96x64 Log Mel Spectrograms from the input audio .wav file. Segments less than 1s are truncated and excluded. In order to generate a single embedding for each of

the files in our dataset, we generated a Log Mel Spectrogram for the entire file and resized it to 96x64 to be the input for VGGish.

6 Evaluation Metrics

The choice of statistics used to evaluate our OSR model include the overall accuracy, per-class accuracy and weighted F1-score. The accuracy is computed using the fraction of correct predictions made by our models as shown in Equation 2.

$$Accuracy = \frac{Number\ of\ correct\ predictions}{Total\ number\ of\ predictions} \tag{2}$$

Although our test dataset has a roughly equal number of known and unknown data points, since the unknown data only comprises of a single class (class label -1) the negative class will have much more data compared to each of the positive classes (labesl 0 - 4). To account for this imbalance, we calculate the weighted F1 score of each test, which weighs the F1 scores for each individual class according to its rate of appearance in the test set. To calculate the weighted F1 score, the F1 score is first tabulated for each class independently as in Equation 3. When added together as shown in Equation 4, it uses a weight that depends on the number of true labels of each class, favouring the majority class.

$$F1 = \frac{2 * (precision * recall)}{(precision + recall)} \tag{3}$$

$$Weighted\ F1\ score = F1_{class1} * W_1 + F1_{class2} * W_2 + \cdot + F1_{classN} * W_N \tag{4}$$

7 Approaches

7.1 Threshold Check

For this method, we will do an additional threshold check on top of the prediction made by the base classification model. If the test data falls outside the boundary defined by the threshold, it will be classified under the unknown class. An overview of the method is shown in Figure 5

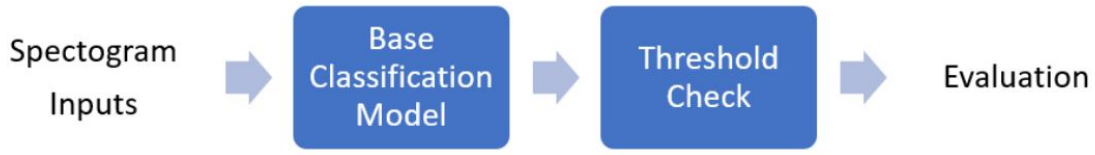

Figure 5: Threshold Architecture

7.1.1 Softmax Threshold

Model Description Given that the softmax activation function is applied to the classification layer of the base model, we will be able to get the relative predicted class probabilities for each data point. We then determine a minimum threshold probability value for each of the known classes using the lowest predicted probability that resulted in a correct classification on the training and validation datasets. With a minimum acceptable probability for each known class now determined, every prediction that the model makes on the test set will be cross checked against the threshold probability for the predicted class. If it falls below the threshold, the original prediction will be rejected and the data will be classified under the unknown class instead.

Intuition We assume that the confidence of the prediction from the base model would be lower for data from classes outside one of the known classes that the model was trained on. In this way, even though the model can only classify input data into one of the known classes, we can filter out data from the unknown class by setting an appropriate minimum prediction confidence.

Result

Softmax Threshold	
Overall Accuracy	0.505437
Class 0 Accuracy	0.980861
Class 1 Accuracy	0.972972
Class 2 Accuracy	0.931034
Class 3 Accuracy	0.894736
Class 4 Accuracy	0.965116
Class -1 Accuracy	0.054587
Weighted F1 Score	0.627390

Table 2: Test results using Softmax Threshold

Evaluation From the results obtained, we can see a high accuracy for the known classes and low accuracy for the unknown class. This performance is similar to what would have been obtained if base model was used on the test set without the implementation of the prediction confidence threshold. This indicates that unknown data is rarely being filtered by the threshold and that when the model makes a wrong prediction, it does so with high confidence. The minimum threshold value we defined for each class is also most likely not a good estimate in setting the boundary for each of the known classes as it was naively determined as described above, making it sensitive to outliers. If there was a data point in the training or validation data set with an unusually low prediction confidence that ended up being correctly predicted, the threshold would be set to this value.

7.1.2 Similarity Comparison

Model Description Using the extracted features from the base model and VGGish, we use 2 metrics to make a comparison between the test data point and the centroid of its predicted class. The 2 metrics are namely the cosine similarity as calculated in Equation 5 to measure how alike 2 data points are and the euclidean distance as calculated in Equation 6 to measure how close 2 data points are. To generate the centroids for each of the known classes, we find the mean value for each metric from the extracted features of the training and validation data sets per class. We then determine the threshold value for each class (minimum cosine similarity or maximum euclidean distance) according to the data point in the training or validation dataset that are the most dissimilar to or furthest away from its class centroid. If the test data point falls below the either of the thresholds of the predicted class, it will be classified under the unknown class.

$$similarity(A, B) = \frac{A \cdot B}{\|A\|\|B\|} \tag{5}$$

$$d(p, q) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2} \tag{6}$$

Intuition The inspiration for this approach came from the K-means clustering algorithm, where data points are clustered into classes using the the 2 metrics mentioned above. We assume that data from the unknown class would be dissimilar and far away from the centroids of the known classes, and hence a suitable boundary can be determined to filter out data from the unknown class.

Result

Euclidean Distance		
	Base Model	VGGish
Overall Accuracy	0.478534	0.483113
Class 0 Accuracy	0.985645	0.985645
Class 1 Accuracy	0.959459	0.972972
Class 2 Accuracy	0.935960	0.935960
Class 3 Accuracy	0.890350	0.894736
Class 4 Accuracy	0.965116	0.965116
Class -1 Accuracy	0.0	0.006968
Weighted F1 Score	0.631200	0.632351

Table 3: Test results using Euclidean Distance

Cosine Similarity		
	Base Model	VGGish
Overall Accuracy	0.480824	0.485975
Class 0 Accuracy	0.985645	0.985645
Class 1 Accuracy	0.972972	0.972972
Class 2 Accuracy	0.935960	0.935960
Class 3 Accuracy	0.894736	0.894736
Class 4 Accuracy	0.965116	0.965116
Class -1 Accuracy	0.002322	0.012775
Weighted F1 Score	0.632785	0.631873

Table 4: Test results using Cosine Similarity

Evaluation From the result obtained, we can see a high accuracy for the known classes and low accuracy for the unknown class for both sets of features, similar to the softmax threshold method. As with the softmax threshold, cluster boundaries created might be too large since extreme values from the training data are used to defined the threshold boundary. Features extracted from VGGish perform slighly better at rejecting data from the unknown class, possibly owing to the fact that the features are more generalizable, but the difference is not significant enough to be of note.

7.2 Generation of synthetic data

Model Description One method we tried was to synthetically generate random data to simulate data from our negative class. For each real datapoint in our training and validation dataset, we appended a datapoint with randomized values for each feature and labelled it as the negative class. This created the distribution of our training data as shown in Table 5.
We chose to use the features extracted by VGGish as our input data as the number of features is relatively small at 128 and each feature is fixed to have a value between 0 and 255, simplifying the randomization process. For each feature, an integer between 0 and 255 is chosen at random.

	Positive Classes (labels 0 - 4)	Negative Class (label -1)
Proportion of Train/Validation Sets	50%	50%

Table 5: Training data distribution for randomized data generation method

The chosen model was a single-layer perceptron (SLP) with 6 output nodes and a softmax activation function, as increasing the number of layers in the classifier network did not seem to have an appreciable effect on the results.

The training parameters were the same as those used to train our base model, with the exception that the batch size was increased to 32.

Intuition Since we assume that we have no knowledge of data in the negative class, we believe that randomly generated data would be the next best alternative. By training our model on randomized negative data points, we hope that the model will learn to identify the positive data and exclude everything else as negative.

Result

Metric	Value
Overall Accuracy	0.4516
Class 0 Accuracy	0.7751
Class 1 Accuracy	0.7838
Class 2 Accuracy	0.8424
Class 3 Accuracy	0.6447
Class 4 Accuracy	0.8547
Class -1 Accuracy	0.1208
Weighted F1 Score	0.3898

Table 6: Test results for randomized data generation method

Evaluation Even though the model achieved a validation accuracy of 0.88 and class -1 validation accuracy of 0.98, Table 6 shows that the results on the actual test data is significantly worse. This indicates that randomized data is not an accurate representation of our negative class. Despite being different types of environmental sounds, our positive and negative class data might share some similarities that are not represented by the randomized data that this model is unable to differentiate between.

7.3 One Class SVC

Model Description We prepend a one-class classification model to our base model to first decide if the input data belongs to one of the positive classes or the unknown class. Data that is classified as negative will not be put through the base model, while data identified as positive will then be passed on to the base model to perform further classification. The architecture is illustrated in Figure 6.
The one class classification model we used was One Class Support Vector Machine (One Class SVM or SVC) with four different kernels (linear, polynomial, radial basis function, sigmoid). We used the model implementation by scikit-learn [11] with the default parameters. The model was trained using all 3 of our data representations, namely the Log Mel Spectrograms, features extracted using our base model, and features extracted using VGGish.

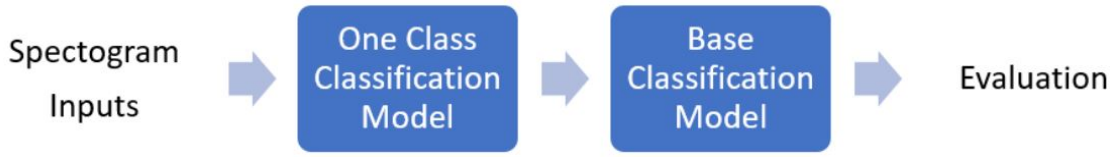

Figure 6: One Class SVM Architecture

When training the one class SVM, all 5 of our known classes were grouped together into a single positive class with class label 1 for the model to be trained on.

Intuition One class SVMs are trained to identify outliers, which are data points that fall outside the target positive class. By grouping all of our known classes into a single positive class, we hope that the one class SVM can identify data from the unknown class as outliers. After knowing that the data is one of our known classes, we can then proceed to classify into one of the specific known classes using the base model.

Result

	Original Spectrograms				Features extracted from Base Model			
Kernel	Linear	Poly	RBF	Sigmoid	Linear	Poly	RBF	Sigmoid
Overall Accuracy	0.5123	0.4876	0.4493	0.5151	0.6783	0.6846	0.3113	0.6742
Class 0 Accuracy	0.3827	0.3971	0.6746	0.4258	0.3923	0.4354	0.4928	0.3636
Class 1 Accuracy	0.4729	0.3783	0.4054	0.4594	0.5270	0.5270	0.1891	0.5270
Class 2 Accuracy	0.6108	0.5665	0.7290	0.7142	0.4384	0.4778	0.5123	0.4088
Class 3 Accuracy	0.6885	0.7061	0.2192	0.5701	0.3991	0.3552	0.5087	0.4166
Class 4 Accuracy	0.2151	0.25	0.3081	0.2325	0.6337	0.5639	0.5465	0.6569
Class -1 Accuracy	0.5365	0.4901	0.4216	0.5365	0.9001	0.9186	0.1312	0.8966
Weighted F1 Score	0.5047	0.4825	0.4430	0.5087	0.6598	0.6653	0.2885	0.6546

	Features extracted from VGGish			
Kernel	Linear	Poly	RBF	Sigmoid
Overall Accuracy	0.4882	0.4917	0.5014	0.4819
Class 0 Accuracy	0.4736	0.4736	0.6172	0.4497
Class 1 Accuracy	0.5810	0.5810	0.2702	0.5675
Class 2 Accuracy	0.5123	0.5073	0.4876	0.4876
Class 3 Accuracy	0.4342	0.4166	0.4473	0.4649
Class 4 Accuracy	0.4767	0.4767	0.4825	0.4883
Class -1 Accuracy	0.4947	0.5075	0.5145	0.4843
Weighted F1 Score	0.4895	0.4927	0.5004	0.4827

Table 7: Test results for One Class SVM

Evaluation The results suggest that the best data representation to use to train the one class SVM are the features extracted from the base model. It is possible that the number of features extracted from VGGish are too few and hence insufficient to accurately describe the data. On the other extreme, the poor performance of the model trained on the full Log Mel Spectrogram representation could be attributed to the high number of features per audio sample as compared to the number of samples, which is not suited for a relatively simple model such as an SVM.

It is also observed that for the model using features from base model, its -1 class accuracy is significantly higher than positive classes. This indicates a possible overfit of the positive class whereby the boundary that encloses the positive class is set to be too tight, leading to a high number of test data being classified as being part of the unknown class. The poor performance of the RBF kernel across all classes could be due

to overfitting of the model due to the sensitivity of the RBF kernel to model hyperparameters, although it is surprising that this behaviour is not reproduced in the models trained on the full spectrograms and VGGish features. The answer might lie in the way scikit-learn sets the default parameters and warrants further investigation.

7.4 One VS Rest

Model Another method that we tried was to adapt the One-vs-Rest (OVR) Multi-Class classification strategy to allow for negative class predictions. In the ordinary OVR method, a binary classifier is trained for each class, with the other classes being grouped as the negative class. During the final classification between classes, the datapoint is put through all the binary classifiers and the predicted class is the one that has the highest confidence [5]. To allow for negative classification, if none of the classifiers produce a satisfactory confidence level, we classify the datapoint as belonging to the negative class.

When training each binary classifier, we relabel the chosen class in our training set as 1 and the other classes all as -1. For example, when training the classifier for class 4, the class label 4 is replaced with 1, and the labels 0, 1, 2, and 3 are replaced with -1.

For this method, we tested multiple different models across different data representations. The definition of each combination tested is as follows:

1. **Pre-trained DenseNet:** DenseNet-201 models with ImageNet weights fine-tuned on Log-Melspectrograms for each binary classification problem, similar to our base model. All training parameters are the same as when training our base model.
2. **SLP with features extracted from base model:** SLPs each with 2 nodes and a softmax activation function trained on the features extracted from our base model. All training parameters are the same as when training our base model, with the exception of batch size which was increased to 32.
3. **SLP with features extracted from VGGish:** Same model 2, but using features extracted from VGGish instead.
4. **Support Vector Machine (SVM) with features extracted from the base model:** 4 different kernels were tested, namely Linear, Polynomial, Radial Basis Function (RBF), and Sigmoid. Grid search was used to fine-tune the hyperparameters (C and Gamma) of each SVM.
5. **SVM with features extracted from VGGish:** Same as model 3, but using features extracted from VGGish instead. Also, the Linear kernel was excluded as it threw errors during training that we were not able to fix.

We did not train SVMs on the full representation of the original spectrograms due to the long training time. We also felt that the large number of features (96000) compared to the number of training samples would not be suitable to train SVMs.

Intuition Training individual OVR binary classifiers for each class solves the problem of not having data from the negative set. We hope that each binary classifier would be able to identify the specified target class from other data, and in combining them we would be able to identify all of the positive classes. Anything that was not predicted to be positive by any of the binary classifiers would then be safe to be predicted as being from the negative class.

Result

	Model 1	Model 2	Model 3
Overall Accuracy	0.6119	0.5924	0.5541
Class 0 Accuracy	0.8517	0.9665	0.7416
Class 1 Accuracy	0.9730	0.9865	0.7703
Class 2 Accuracy	0.8670	0.9606	0.7488
Class 3 Accuracy	0.7807	0.8904	0.6009
Class 4 Accuracy	0.9419	0.9767	0.7326
Class -1 Accuracy	0.3519	0.2253	0.3961
Weighted F1 Score	0.5987	0.5373	0.5509

Table 8: Test results for OVR NNs

	Model 4				Model 5		
Kernel	Linear	Poly	RBF	Sigmoid	Poly	RBF	Sigmoid
Overall Accuracy	0.4922	0.4934	0.4945	0.4504	0.4425	0.4539	0.4808
Class 0 Accuracy	0.9952	0.9952	0.9952	0.9904	0.9904	0.9952	0.5407
Class 1 Accuracy	1.0	0.9864	0.9864	0.7972	0.8648	0.8919	0.2163
Class 2 Accuracy	0.9704	0.9803	0.9802	0.9212	0.8768	0.9015	0.5025
Class 3 Accuracy	0.9342	0.9298	0.9341	0.7368	0.7237	0.7851	0.0088
Class 4 Accuracy	0.9767	0.9884	0.9942	0.9852	0.9244	0.9128	0.0465
Class -1 Accuracy	0.0	0.0	0.0	0.0	0.0	0.0	0.6957
Weighted F1 Score	0.3403	0.3399	0.3413	0.4505	0.3162	0.3190	0.4808

Table 9: Test results for OVR SVCs

Evaluation For the OVR NNs tested, we can observe from Table 8 that Models 1 and 2 outperformed Model 3 in the classification of the positive classes. This can be attributed to the fact that both Models 1 and 2 have a significant part of their models that were trained or fine-tuned on our training set. For Model 3, the VGGish feature extractor was not fine-tuned and as such can be expected to produce more generalized features. This can also explain why Model 3 performs the best in identifying the negative class. That being said none of the models are particularly effective at that. For datasets that have a 50/50 distribution of positive and negative data such as our test set, Model 1 is the best performer based on its weighted F1 score.

For the OVR SVCs, we have observed that most SVCs with different kernels are unable to correctly predict the negative classes but able to classify positive classes with relatively high accuracy. However, Model 5 Sigmoid kernel performs relatively well on the negative class, but it performs significantly poorer as compared to the other kernels when classifying the positive classes. This could be due to the reason that Model 5 Sigmoid uses features extracted from VGGish which produces more generalized features and does not overfit our model while Sigmoid kernel was able to map the non-linear features into a higher dimensional space that is separable.

Between the NNs and SVCs, the performance when identifying the positive class is similar with the exception of Model 3 and the Sigmoid kernel for Model 5. This suggests that when solving the closed set problem using an SVC might be just as effective while having a lower training cost especially when compared to the model size and complexity of DenseNet-201. However, the ability of the NNs to identify the negative class in general is far superior to that of the SVCs, and hence are the stronger models for solving the open set problem.

8 Conclusion

From our study, we have a couple of key points of learning. Firstly, there is no strict best model. What is considered to be the best model heavily depends on the intended application, and more specifically the relative importance of false negative and false positive data. For example, in the case of an environmental sound audio classifier to help the deaf navigate false negatives might be more significant as compared to false positives, especially if the system is designed to warn the user about potential dangers. In such a case, a model such as the multiple OVR pre-trained DenseNet classifier (model 1 under the OVR approach) might be chosen for its high positive class accuracy and limited but existent ability to reject the negative class. However, we also believe that none of the models tested are suitable to be deployed in an actual real-life application. Despite being better than a random classifier, none of the models achieved a weighted F1 score of greater than 0.7, and even those that achieved a weighted F1 score of greater than 0.6 were extremely biased towards either the positive or negative class, and only achieved that score because of the 50/50 split of the test set between unknown and known classes.

That leads to our next finding, which is that it is observed that there is an inverse relationship between the accuracy in predicting known classes and unknown classes. A higher accuracy for known classes will correspond to a lower accuracy for unknown classes. This suggests that none of the models are able to clearly define the difference between the positive and negative classes, and have a tendency to behave like naive classifiers choosing either positive or negative with a heavy bias.

We also observed that the models trained on the features extracted by our base model had the tendency to overfit to the training data and produce poor results on the test set. This is likely due to the fact that the base model itself was trained on the same training dataset. As such, our suggestion is to find a more generalized feature extractor such as VGGish if less complex models such as SVMs are to be used for the OSR audio classification problem. Alternatively, if the complexity of the model is no issue then using a CNN with the full Log Mel Spectrograms as inputs might be the best performing method.

Finally, it is possible that we were too strict in our problem definition by requiring all positive classes to be accurately classified. If this requirement is relaxed, it becomes possible to construct a validation set consisting of known-unknown classes, which are classes that are unknown to the model during training but is used as part of the validation set to tune hyperparameters [13]. This could potentially allow us to improve the performance of our approaches, in particular the threshold and random data generation approaches. Known-unknown classes would allow us to not set the threshold values naively based on the extreme datapoints in the training set, but possibly tune it to a percentage of that value, based on the validation results. For random data generation, it would allow us to better determine the key differences between the positive and negative data, and only randomize those key features rather than the full feature vector. This would allow the random data to better represent the unknown class.

References

[1] S. Thornton B. Zhang J. Leitner. *Audio Recognition using Mel Spectrograms and Convolution Neural Networks*. 2019. URL: http://noiselab.ucsd.edu/ECE228_2019/Reports/Report38.pdf.

[2] Terrance Boult et al. *Learning and the Unknown: Surveying Steps Toward Open World Recognition*. 2019.

[3] J. F. Gemmeke et al. *Audio Set: An ontology and human-labeled dataset for audio events*. New Orleans, LA, 2017.

[4] S. Hershey et al. *CNN Architectures for Large-Scale Audio Classification*. 2017. URL: https://arxiv.org/abs/1609.09430.

[5] J.H. Hong and S.B. Cho. *A probabilistic multi-class strategy of one-vs.-rest support vector machines for cancer classification*. Advances in Neural Information Processing (ICONIP 2006) / Brazilian Symposium on Neural Networks (SBRN 2006). 2008. DOI: https://doi.org/10.1016/j.neucom.2008.04.033. URL: http://www.sciencedirect.com/science/article/pii/S0925231208003007.

[6] G. Huang et al. *Densely Connected Convolutional Networks*. 2017. DOI: 10.1109/CVPR.2017.243.

[7] H. Jleed and M. Bouchard. *Open Set Audio Recognition for Multi-Class Classification With Rejection*. 2020. DOI: 10.1109/ACCESS.2020.3015227.

[8] kamalesh0406. *kamalesh0406/Audio-Classification*. 2020. URL: https://github.com/kamalesh0406/Audio-Classification.

[9] Brian McFee et al. *librosa/librosa: 0.8.0*. July 2020. URL: https://zenodo.org/record/3955228.

[10] Kamalesh Palanisamy, Dipika Singhania, and Angela Yao. *Rethinking CNN Models for Audio Classification*. 2020. arXiv: 2007.11154 [cs.CV].

[11] F. Pedregosa et al. *Scikit-learn: Machine Learning in Python*. 2011.

[12] J. Salamon, C. Jacoby, and J. P. Bello. *A Dataset and Taxonomy for Urban Sound Research*. 2014. DOI: 10.1145/2647868.2655045.

[13] Walter J. Scheirer, Lalit P. Jain, and Terrance E. Boult. *Probability Models for Open Set Recognition*. Nov. 2014.