Audio Classification

Dataset:

Details about the dataset can be found on Kaggle under the Title: Freesound General-Purpose Audio Tagging Challenge URL: https://www.kaggle.com/c/freesound-audio-tagging

You can download the data at URL: https://zenodo.org/record/2552860#.YJRPRLUzaUl

1 Introduction

Improvements in technology over the past decades have enabled the development of products to automate and simplify our daily lives. One method that enables us to interact with our environment and the devices we are using is sound. To ‘unlock’ the communication between human and machine and benefit from the capabilities our devices offer we need to train them to understand sound. Audio classification is already a big part of our routine with the use of virtual assistants such as Siri and Google assistant, security systems, automatic speech recognition and text to speech applications. These devices cannot interpret sound, but nevertheless they are able to understand commands and convert audio into text that enables them to serve their purpose. Exploring further the area of sound classification and finding techniques which offer more accurate and fast classification can revolutionize our communication with these devices.

In our project we train a model to classify audio from 41 different classes, making use of a Kaggle dataset which contained 9500 training samples from classes such as musical instruments, gunshots and other sounds that can be come across in our daily life.

In the following sections we outline our approach, analyse how we pre-process the audio files in the best way to achieve better classification and the various Neural Networks explored. The performance of these networks was evaluated using a separate test set from the same source with 1600 samples. By making use of two different processing methods, we trained two Neural Networks with different architectures and compared the accuracy of each of those combinations.

2 Method

2.1 Dataset

All the audio samples in the dataset were collected from Kaggle (link in references). The audio files are delivered as uncompressed Pulse Code Modulation (PCM) with bit depth of 16 as mono audio files, sampled at a frequency of 44.1 kHz. The dataset is split into a train set and a test set. The train set contains approximately 9,500 samples distributed unevenly over 41 different categories while the test set contains around 1,600 samples. The time spam of the audio files varies from 300ms to 30s because of the diversity between the categories. The total audio per class can be seen in *Figure 1*, distribution in duration between each class can be seen in *Figure 2*. Labels for the dataset were provided using crowdsourcing, with all the test data and ~40% of the training data labels manually verified and marked as such. Unverified labels have a quality estimate of at least 65-70%.

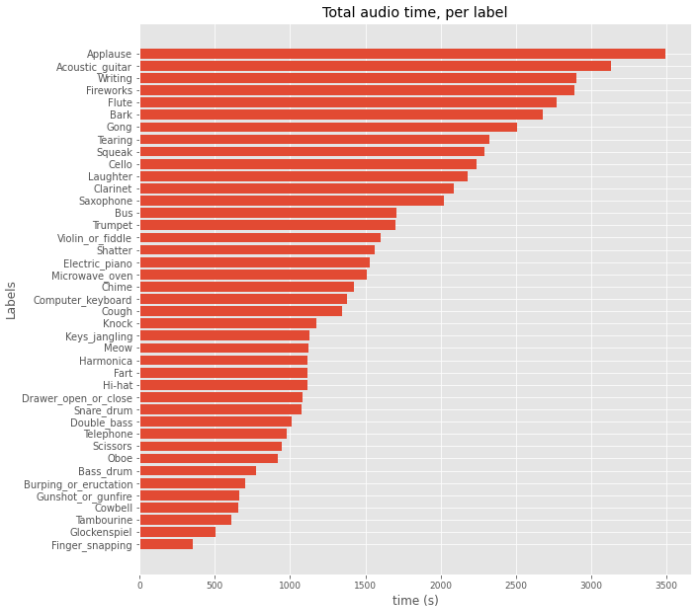

Figure 1: Total audio time length per class.

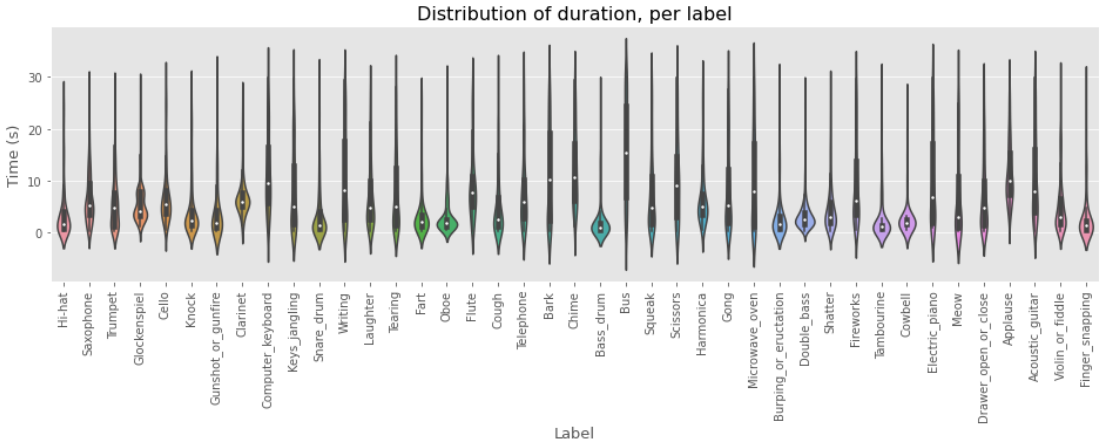

Figure 2: Distribution of Duration per class

2.2 General Pre-processing and feature extraction

1) Filter Bank Coefficients

Given our audio signals are changing constantly, we take short-time Fourier transforms of our data, comparing different sizes of windows and step sizes, to estimate the spectral density of our signal at each time step (McKinney & Breebaart, 2003). Taking the logarithm of these energies (as human hearing also works on a logarithmic scale) we can create a 2d array of spectral density estimates. Human hearing also perceives relative changes in frequency as being much larger at lower frequencies than at higher frequencies. The Mel scale provides a transformation for the linear frequency range to one relative to human hearing. We use several triangular filter-banks spaced equally on this scale to re-bin our data to filter bank coefficient, shown in the figure 3 (right), below.

2) Mel Frequency Cepstrum Coefficients

Our final pre-processing step is to extract MFCCs by taking discrete cosine transformations of our filter bank coefficients , decorrelating the data. We keep the sum of the lowest frequency coefficients while discarding the rest, as this has been shown to improve performance in speech recognition tasks. (Fayek,H. 2016)

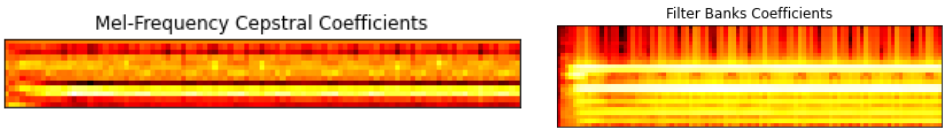

Figure 3 – Example MFCC and Filter Bark Coefficient Features

Overall, we used two different pre-processing and feature extractions methods and compared their results.

Method 1:

The first method extracts a random 2 second window from every audio file. If the total duration of any given audio file happens to be less than 2 seconds, we pad the audio to compensate. We then extract 40 MFCC features using the librosa package, at the original sampling rate of 44100 Hz and a hop length of 512. The audio is transformed into numpy arrays with dimensions of number of mfcc features x the number of total samples divided by the hop length (40,173). The data are then standardized, and the dimensions are expanded from two to three to create a visual input for our networks.

Method 2:

The second method involves cleaning the audio files. As can be seen in *Figure 4*, the raw audio files contain sections of low to little volume. We consider these regions of the data to contain minimal relevant information and so remove them using a filter. To achieve this, we trim the leading and trailing silence from the audio files, where the threshold in

decibels is considered to be zero. The filtered signal can be seen in the *Figure 5*. After cleaning the data, the second method extracts the first 4 seconds of every audio file as we assume that the start of the audio conveys the most relevant information. The same technique of padding was performed to samples whose total duration dropped below 4s. Instead of using the librosa library, we extract the features using a custom preprocessing layer in our model from the kapre library. 128 MFFC features are extracted, again at the original sampling rate of 44100 Hz but with a higher hop length of 128. The audio is automatically tranformed into numpy arrays with dimensions 200x128x1. We assume this second method will produce better results as we extract more information for every sample.

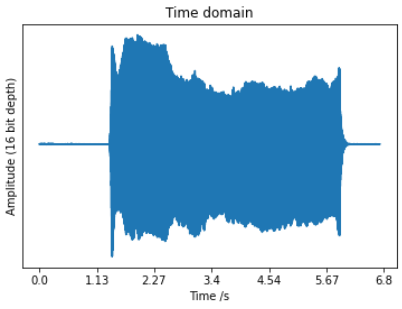

Figure 4: Before filtering

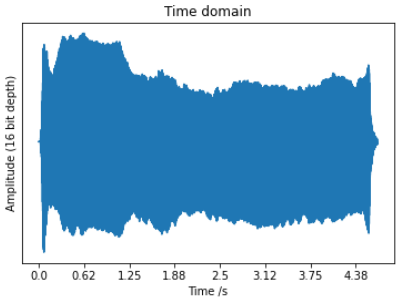

Figure 5 : After filtering

2.3 Model architecture

Both the methods discussed above were tested with two different model architectures.

The CNN network expects an input of 40x173x1 or 200x128x1, depending on the method of pre-processing. The first four convolutional layers are almost identical using 32 4x10 filters with the same padding, batch normalization and a Rectified Linear Unit (relu) as the activation function. The only difference between is them is that a dropout layer of 0.5 follows the first three convolutional layers to prevent overfitting, while a Dropout of 0.1 is followed after the 4th convolution layer. The output of the final convolutional layer is then flattened and followed by a fully connected layer with 64 rectified units. Lastly the final layer is again fully connected with a Softmax activation function and an output of 41, one for each class.

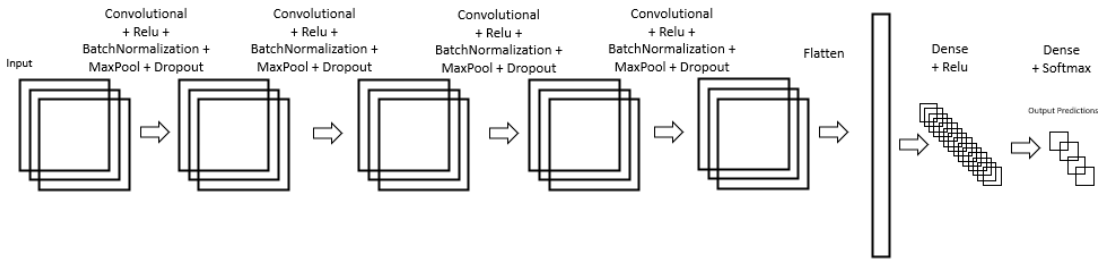

Figure 6. CNN model architecture used

The alternative model used is the much complex model of ResNet50, a variant of the popular ResNet model which has 48 Convolutional layers, along with 1 MaxPool and 1 Average Pool layer. After the ResNet50, there is a GlobalAveragePooling2D layer to apply average pooling on the spatial dimensions until there is one dimension by averaging the values, followed by a fully layer 1024 rectified units. The output layer is fully connected with a Softmax activation function and an output of 41, one for each class.

Layer name	Output size	ResNet50 Attributes
conv1	112x112	*7x7, 64, stride 2*
		3x3 max pool, stride 2
conv2.x	56x56	$\begin{bmatrix} 1x1 & 64 \\ 3x3 & 64 \\ 1x1 & 256 \end{bmatrix} x\ 3$
conv3.x	28x28	$\begin{bmatrix} 1x1 & 128 \\ 3x3 & 128 \\ 1x1 & 512 \end{bmatrix} x\ 4$
conv4.x	14x14	$\begin{bmatrix} 1x1 & 256 \\ 3x3 & 256 \\ 1x1 & 512 \end{bmatrix} x\ 6$
conv5.x	7x7	$\begin{bmatrix} 1x1 & 512 \\ 3x3 & 512 \\ 1x1 & 2048 \end{bmatrix} x3$
	1x1	*Average pool, 1000-d fe, softmax*
FLOPs		$3.8\ x10^9$

Table 1 ResNet50 CNN attributes

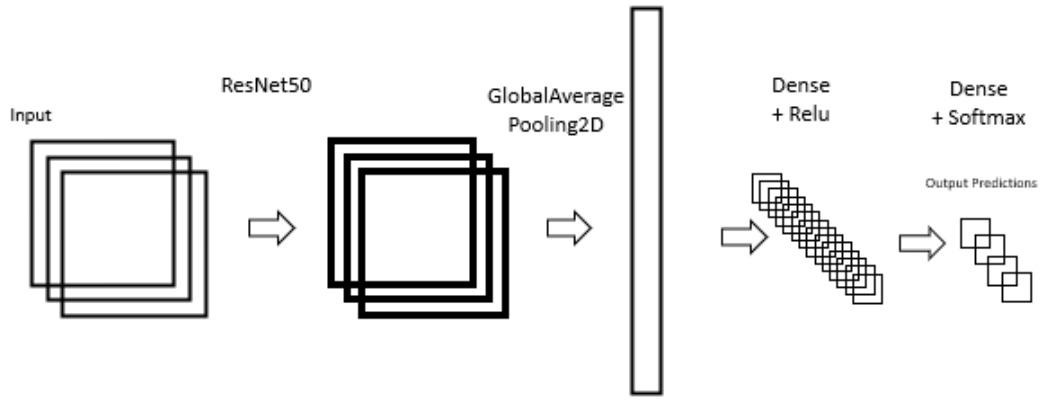

Figure 7. Architecture of the ResNet model used

Since the classes of our dataset are imbalanced, we provide the class weights during model training to compensate for this. We additionally assign higher weights to samples that have been manually verified since some non-verified labels are assumed to be incorrectly labelled. We optimise a categorical cross-entropy loss using an Adam optimiser with learning rate of 0.001 and batch size of 64. The models are trained until the validation loss stops decreasing, at which point we reduce the learning rate to 0.0001 and repeat the procedure until the model converges again. We also use a checkpoint call back to save the best performing models and a tensor board call back to visualise our results.

3 Results and Discussion

Both methods were evaluated using both models. *Figure 8* shows the results of the training.

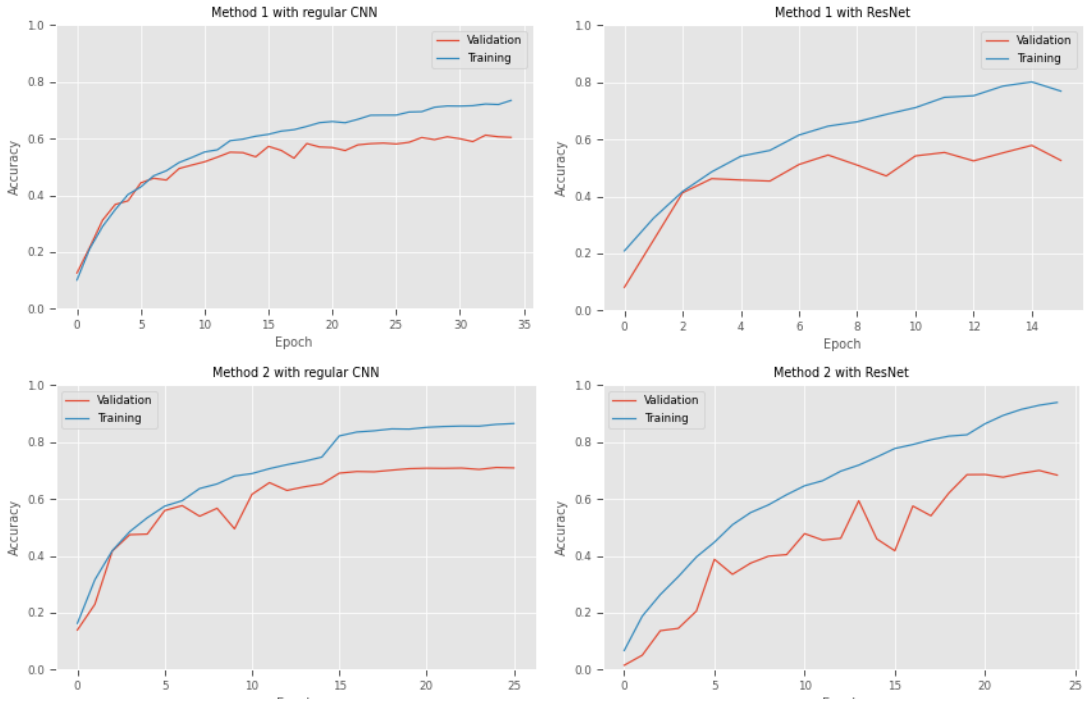

Figure 8. Performance of the models

The graphs demonstrate that method 2, which involved cleaning the data and extracting higher dimensional features from the raw data achieved both a better training score and a validation score. ResNet50 methods achieved much higher training scores than the CNN suggest overfitting due to the significant complexity of the model given our task. On the other hand, the validation scores for the different architectures on a given method were similar. In the table below,

To improve accuracy on the test set, we also use an ensemble of the two networks, which combine the results of the models. The following test scores were recorded:

Method	Model	Training%	Validation%	Test Accuracy%
1	CNN	72	61	64
	ResNet	80	58	65
	Ensemble	--	--	69
2	CNN	86	71	76
	ResNet	94	70	73
	Ensemble	--	--	78

Table 2 Training - Validation - Test scores for both methods discussed.

Both models scored better using method 2. In fact, the test accuracies achieved were higher than the validation scores which could be a result of all the samples being manually verified in the test set. Using an ensembled prediction, method 2 was able to correctly predict around 1250 labels out of the 1600 of the test set from 41 different labels.

Even though the ResNet50 architecture is much more complex, the improvement in accuracy was only 1% higher than the CNN architecture for the first method, and worse for the second, suggesting that the features in our data are captured sufficiently with the smaller CNN and the additional complexity of the ResNet did not provide any benefit.

We also observe that the additional MFCC coefficients and longer sample length extracted using the second method made a significant improvement to both models' ability to classify the audio sources.

As stated before, the data were sampled at 44.1 kHz. By down sampling the data to lower frequencies of 32 kHz or 16 kHz, it should lead to an easier analysis of longer temporal range since the data size will decrease. Even though low sampling rate might lose useful information present in higher frequencies, it will allow for more complex techniques to be explored since the computational power needed would be much lower.

To further improve the accuracy of our model, we would look into enhancing our training dataset using augmentation methods such as mixup augmentation, where samples from classes are combined, or scale augmentation to randomly scale the data. We predict this would increase classification accuracy by providing additional samples for our models to train on and has been shown to be an effective approach (Jeong, I.Y et al. 2018).

Despite the top performance of convolutional neural networks, they might fail modelling sequences, such as audio classification (Phan et al. 2017). Recurrent neural networks are extremely good at dealing with sequential data. In fact, Long Short-Term Memory (LSTM) approaches can be optimized to understand long-range patterns, such as audio files. They are often used in combinations with the CNNs to merge the pattern recognition capabilities along with the feature learning capabilities.

REFERENCES

Fayek, H., 2016. Speech processing for machine learning: Filter banks, mel-frequency cepstral coefficients (mfccs) and what's in-between. *URL: https://haythamfayek. com/2016/04/21/speech-processingfor-machine-learning*

Jeong, I.Y. and Lim, H., 2018. Audio tagging system for dcase 2018: focusing on label noise, data augmentation and its efficient learning. *Tech. Rep., DCASE Challenge*.

Kao, C.C., Wang, W., Sun, M. and Wang, C., 2018. R-CRNN: Region-based convolutional recurrent neural network for audio event detection. *arXiv preprint arXiv:1808.06627*.

McKinney, M. and Breebaart, J., 2003. Features for audio and music classification.

Phan, H., Koch, P., Katzberg, F., Maass, M., Mazur, R. and Mertins, A., 2017. Audio scene classification with deep recurrent neural networks. *arXiv preprint arXiv:1703.04770*.