Classification of Dogs and Cats Audio

MLND  
Jonathan Mak

# Project Overview

Classification of audio sources have long relied on application of statistical methodologies such as SVM and dimension reduction techniques such as PCA. With the advent of neural network and subsequent rise of deep learning, the problem of positional invariance has been given an alternative solution in the form of convolutional neural network (CNN).

Through use of filters and pooling layers, CNN allows for extraction and aggregation of localized features at various resolution levels, greatly enhancing machine’s understanding of contexts. This advantage is most notably shown in its application in image recognition where CNN can extract features both small and big and construct decision boundaries upon them.

Another application of such strengths is that of audio recognition where the audio wave is decomposed into bands of frequencies and corresponding intensity (harmonics) in a time series, filters are then applied to capture the unique audio signatures of say, a cat.

This project is inspired by the audio files of cats and dogs available in Kaggle. The audio records store the cat “meows” and dog “barks” as WAV. Although the two different sources can easily be classified by humans, traditional methodologies such as SVM may not apply well because of activations at different time points. However, it is expected an appropriate CNN architecture will be able to accurate model data.

# Problem Statement

The proposed solution is that of a convolutional neural network which takes the Fourier Transformed WAV data as input and outputs a binary value to denote whether the sample is dog or cat. The CNN model consists of layers of convolutional filters followed by activation and regulation. After the last convolutional block, activations are flattened and a binary value (probability) is calculated through a fully connected layer and sigmoid activation.

# Metrics

The specified validation set (49 dog clips and 49 cat clips) will be used to measure model performance at test time. Since the number of clips are balanced, a simple accuracy score would be appropriate for the binary classification problem.

# Data Exploration

The data used in this project is “Audio Cats and Dogs” dataset from Kaggle: (<https://www.kaggle.com/mmoreaux/audio-cats-and-dogs/data>)

The original dataset is divided into train and test set but for the purpose of this project, the train set is further divided into train and validation set for model validation.

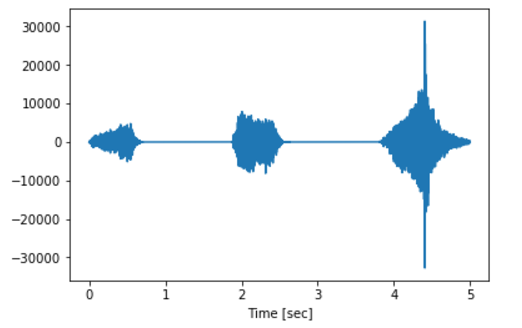
The data specs are as follow:

* Length: 2 to 10+ seconds per clip
* Train/test split: recommended sets of train and test sound clips are specified
* Number of clips:
  + Train: 64 (dog) and 115 (cat)
  + Validation: 14 (dogs) and 30 (cats)
  + Test: 49 (dog) and cat (cat)
* Total clip length:
  + Train: 728s (cat) and 250s (dog)
  + Validation: 213s (cat) and 67s (dog)
  + Test: 382s (cat) and 281s (dog)
* Sampling rate: 16000Hz
* Format: WAV

# Exploratory Visualisation

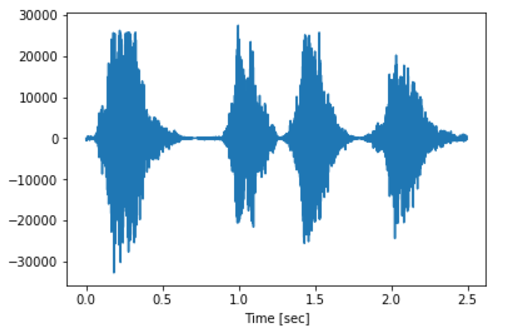
## WAV visualisation

Using *scipy.io.wavfile*, these files are read into arrays of length(x) with x being the sample length of the record. *Maplotlib* can then be used to visualize the values in the array. For example:



In this example, the recording contains three cat “meows”, one at the beginning, one in the middle and one at the end. The length of each can be found by measuring the width of the activated area.

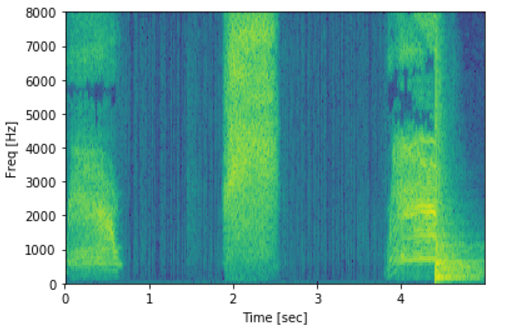
An example of dog bark is shown as follow:



The clip shows four dog barks. It should be noted that although the length of each bark is generally less than that of a “meow”, as one may expect, it may not always be the case. Another discernible difference is that dog barks tend to be louder (intensity).

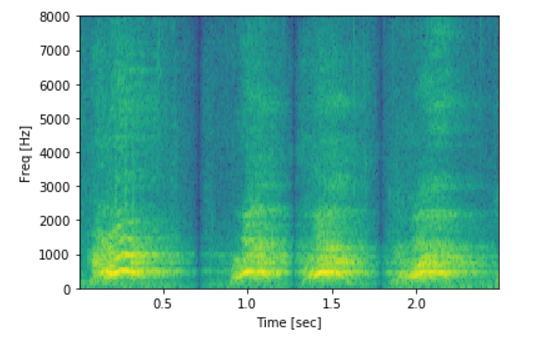
## FFT Visualisation

One common way to process waveforms into meaningful data is to use Fourier Transform. In Python, *scipy.signal* converts the wav data into a spectrogram format. For the cat example used above, this becomes:



Here we can see that the activation areas in the wave data correspond to the highlighted (green) regions in the spectrogram. In each green band, some frequency ranges have higher intensity than others, denoted by the difference in colour.

An example of dog bark is also shown below:



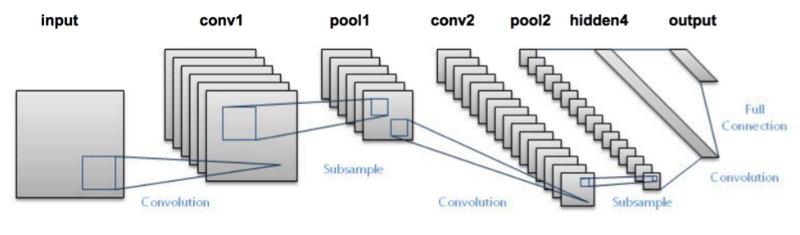
The spectrogram shows 4 green/yellow bands which correspond to the 4 barks recorded. There is an apparent difference in distribution of intensity across frequency ranges compared to that of cat “meow”, where dog barks seem to have intensity concentrated in lower frequency zones. This aligns with intuitions as dog barks often sound “deeper” than cats.

# Algorithm and Techniques

Convolutional Neural Network (CNN) is applied to the problem of classifying cat and dog audio source. CNN is built based on the concept of ordinary multilayer neural network but with the addition of several features:

1. Filters  
   Filters with trainable weights are applied to input arrays using a sliding window method to capture localised features by calculating and summing dot products of filter elements and input elements. These summed values are usually then fed into an activation function and become another array for pooling or more filter operations.
2. Pooling  
   Pooling layers are applied to input arrays by extracting the max/min/average values in a sliding window, outputting one value over all elements covered in each step. This operation is usually done with a step size larger than 1 to downsample the input array, which effectively reduce the resolution of the “image”

These two operations are applied over multiple layer to form a deep learning network, for example:



In the context of this project, the input into the CNN architecture would be the spectrogram data generated from applying Fourier Transform on the original one-dimensional wav data. The spectrogram data has dimension (number of sampled frequency, time) and is cut into arrays of fixed dimension (number of sample frequency, window\_size) where window\_size is a tuneable hyperparameter that specifies the length of each extracted sound clip.

Filters with trainable weights are applied to the input array to calculate the dot products of elements covered by the window in each step, followed up RELU activation to introduce non-linearity. A max pooling layer is applied to downsample (reduce dimension) of the array by calculating the maximum value of each window.

The above layer would then be repeated to downsample the feature arrays. It should be noted that after each application of the filter-activation-pooling layer, output arrays are used to calculate a 16-dimension array which hold intermediate prediction results. These arrays would eventually be concatenated and connected to a dense layer to calculate a probability value which denotes whether it is a cat or dog.

# Benchmark Model

The benchmark model is trained by applying Support Vector Machine (SVM) on the processed spectrogram data. Input wav data is first converted into spectrogram data by using *scipy.signal*. The maximum intensity across time is then extracted for each sampled frequency in the spectrogram array for each recording. The resulting features are then used as input for an SVM with RBF kernel to construct a model for the audio classification task.

The performance of the benchmark model is as follow:

* Accuracy: 83%
* Precision: 87%
* Recall: 83%

# Data Preprocessing

Source data is processed to suitable format by:

1. Since the source data is in .wav format, each recording will have to be read into memory as a 1D array using *sci\_wav.read()* in *scipy*, available in Python.
2. The output of step 1 is then converted into spectrogram data using Fourier Transform using *signal.spectrogram*, also available in the *scipy* package in Python. The module by default sample the input array by 129 different frequencies and output an array of dimension (129, length of clip).
3. The resulting data is then standardized, per recording, using *StandardScaler* (mean-subtraction and standard-deviation-averaging), available in the *sklearn.* From here, preprocessing splits into two streams, one for training CNN and the other for SVM.

For training CNN:

1. The normalized data from each clip is concatenated to form train and test clips for cats and dogs separately (total 4 long recordings/arrays).
2. A Python generator object is created to randomly 20 samples an array of dimension (129, window\_size) from train or test data and output as a batch. The generator object provides the advantage of (almost) unlimited training examples, each begins at any random point in time in the given recording. The generator also output the labels for each batch.

For training benchmark SVM:

1. For each (uncontatenated) recording, the max intensity for each sampled frequency in step 3 is calculated, reducing the dimension of each array to be (129, ) for each clip.
2. Train and test data are separated and assigned a label [1, 0] for each clip.

# Implementation

The CNN model is implemented using *keras*, available in Python, with the following specifications:

* Input: arrays of dimension (batch\_size, number of sampled frequencies = 129, window\_size)
* CNN Blocks: 4 blocks, each consists of a 2D filter of dimension (129, 8) with stride 4, followed by MaxPooling of window size 3 and stride 2, followed by a relu activation.
* Output layer: the output from the last CNN block will be regulated with, a dense layer is used to calculate a single value with sigmoid activation to output a probability.

The hyperparameters in this model are:

* Batch\_size: number of sample per batch
* Window\_size: length of time points included in each array
* Dropout rate: proportion of parameters to be dropped out for regulation
* Learning rate: how much does each iteration update parameters

Window\_size has a default value of 512 to include enough time points to cover an entire bark or “meow”.  
Batch\_size has a default value of 20 to provide sufficient training example each batch.  
Step\_per\_epoch has a default value of 50 to provide sufficient training example each batch

Epoch, Dropout and Learning Rate are tuneable parameters in grid search.

The CNN architecture can be represented graphically as follow:

# C:\Users\Jonathan Mak\AppData\Local\Microsoft\Windows\INetCache\Content.Word\model.png

# Refinement

The training and calibration of the model was done using a grid search over dropout and learning rate:

* Learning rate: [0.001, 0.0001, 0.00005]
* Dropout: [0.2, 0.35, 0.5]

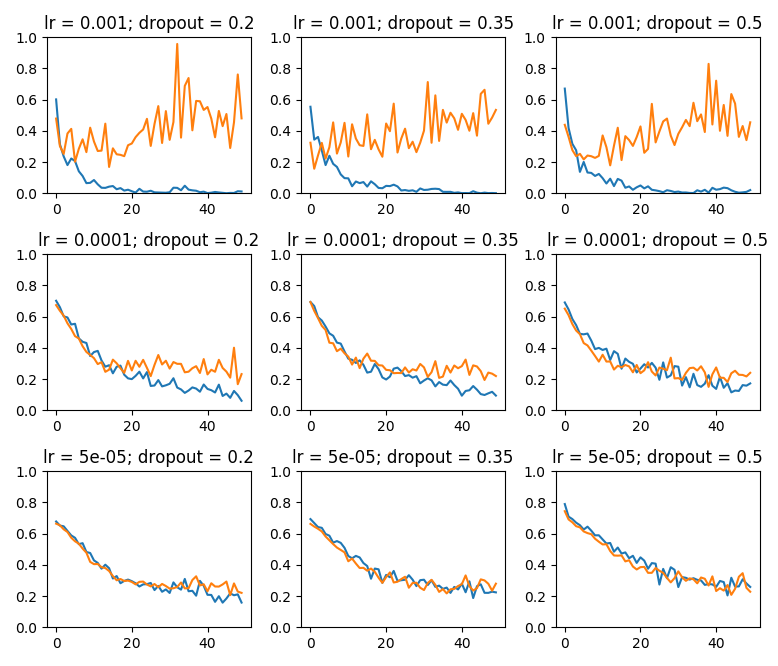
The grid search method iteratively trains the CNN model proposed above with different combinations of learning rate and dropout rate and records the train accuracy, evaluated at the end of each epoch (after 20 \* 100 = 2000 examples randomly generated from train batch) and test accuracy (also 20 \* 50 = 1000 examples randomly generated from test batch).

This methodology deviates from conventional set up of validation sets. This is due to the generator approach adopted (described in Data Preprocessing) which gave the advantage of virtually unlimited training examples. The validation approach is to mimic that of training.

# Model Evaluation and Validation

## Grid Search

The grid search results are as follow:



Grid search shows that the CNN model test performance diverges with high learning rate (0.001), inititally converges but fails to continue at medium learning rate (0.0001). With low learning rate (0.00005) the learning successfully converges and seems to perform better with high dropout.

The final hyper parameters are chosen to be

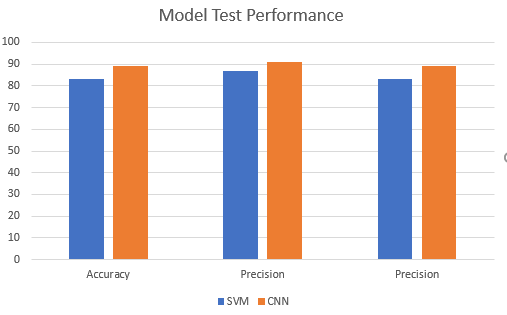
* learning rate = 0.00005 (from grid search)
* dropout = 0.5 (from grid search)
* epoch = 10 (to allow for further convergence)

The testing of the model is done by dividing each original clip into segments with window size being the window\_size used in the CNN model to fulfil the input requirements. Predictions are made for all segments and the final classification result is calculated by averaging the resulting probability [0, 1]. If the averaged probability over segment(s) is less than 0.5, the clip is classified as cat, otherwise dog.

The test metrics are reported as follow:

* accuracy: 0.89
* precision: 0.91
* recall: 0.89

The following chart compares the test performance of the CNN and benchmark model:

* 

# Justification

Given a balanced test set, CNN yields better performance in terms of accuracy than the bench SVM model (0.89 vs 0.83). While this is just a mere 6% difference, it is a substantial gap. This result aligns well with expectation as the averaging over time of intensity level in the preprocessing stage of the SVM model loses a lot of useful audio features and patterns which are otherwise captured in a CNN framework, where application of filters can recognise and capture local features.

# Robustness

The model is tested also on external dataset to evaluate its robustness. The external dataset chosen is that from <http://www.wavsource.com/animals/animals.htm>. 6 clips of dog sounds and 6 clips of cat sounds are used. The wav files are processed the same way as the train and test set albeit with a different sampling rate because of the difference in sources. The FFT data is then fed to the trained CNN model to output predictions.

The results are as follow:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Item | True Label | Probability | Prediction | Correct? |
| Dog | 1 | 1.86E-01 | 0 | N |
| Dog | 1 | 3.23E-01 | 0 | N |
| Dog | 1 | 6.61E-01 | 1 | Y |
| Dog | 1 | 2.90E-01 | 0 | N |
| Dog | 1 | 5.58E-01 | 1 | Y |
| Dog | 1 | 1.76E-02 | 0 | N |
| Cat | 0 | 3.81E-01 | 0 | Y |
| Cat | 0 | 4.36E-01 | 0 | Y |
| Cat | 0 | 5.25E-01 | 1 | N |
| Cat | 0 | 6.33E-01 | 1 | N |
| Cat | 0 | 4.35E-01 | 0 | Y |
| Cat | 0 | 4.43E-01 | 0 | Y |

The performance is disappointing as the model returns a 50% accuracy in the external dataset.

Analysis of the cause of this reveals a major shortcoming of the model. Comparing the sampling rates of train data and external test data:

* Train: 16000Hz
* External: 8000, 22050, 11025 Hz

The difference in sampling rate results in a major discrepancy in length of the FFT (spectrogram data):

* Train (dog): 353
* Train (cat): 584
* External (dog): 103
* External (cat): 116

It is clear that the while the audio features of the train and external data may be very similar (audibly the same), the features are more compressed in shorter arrays in the external dataset. This this causes the CNN to perform poorly on external data as the sampling rates are different. The implication of this is that data preprocessing has to account for potential re-sampling of wav files.

# Reflection and Improvement

While the test results are promising, more effort should be devoted to further enhance the performance of the CNN model. Down below are some suggestions on further work:

* Data preprocessing:  
  As mentioned in the last section, although the CNN model works well on the test set, it fails to account for different sampling rates because the CNN model was trained only with samples with sampling rate 16000Hz. This can be either solved by including different training examples with varying sampling rates or incorporating re-sampling/decompression of wav files in the data preprocessing step.
* Augmentation of training and testing data:  
  By adding noise to the background or even overlapping cats and dogs, the model can be trained to be even more robust to background audio. The sheer increase in training example is also expected to help model test performance.
* Alternative model design:  
  While the proposed model used certain types of Conv layers, more architectures should be explored to find out which suits the audio data the most. Adding/reducing number of layers, including/excluding skip layers and use of other regulation methods such as global averaging are some good ideas to start with.
* Variety of data input:

The proposed model uses a window\_size of 512 based on rough investigation of model performance. However, in light of the successes of ensemble models in competitions, it is worth investigating whether the making an ensemble of CNNs with inputs of different sizes would increase accuracy.

# Conclusion

This project took the Audio Cat And Dog data from Kaggle and trains a Convolutional Neural Network for classification of cat “meows” and dog “barks”. Data from Kaggle was processed in Python by first reading into 1D arrays and then Fast Fourier Transform, available in the Scipy module to generate spectrogram data. The resulting data is then fed into a 2-layer convolutional neural network with dropout as regulation and outputs prediction in the form of a binary value

The model delivers satisfactory performance with the test set, yielding an accuracy of almost 90% and beating that of the benchmark SVM model by a fair margin. However, the robustness of the model questionable as it only performs well with wav files of a certain sampling rate.

Further work on the CNN model is recommended to increase test accuracy. The viable pathways include re-sampling wav files in data preprocessing, adding more convolutional layers, augmenting existing train data as well as tuning hyperparameters that are not part of grid search.

## Personal Reflection

The greatest challenge I faced in the project was that of understanding the processing of audio data and common ways to process them. In particular, a lot of research was done on figuring how to best measure the quality of FFT data. This task extends to the design of data preprocessing.

The core part of the project, the CNN model, comparatively, was more straightforward as I had a good grasp of the methodologies beforehand, having applied it in several occasions. The more difficult part of it was in fact the visualisation and interpretation of results as matplotlib is quite clunky.

Overall, I have found new appreciation in data preprocessing and model evaluation (things other than building model) in the project and I thoroughly enjoyed it.