Machine Learning Engineer Nanodegree

Capstone Project

Vivek Udacity
May 6th, 2018

I. Definition

Project Overview

The project is closely related to Automatic Speech Recognition except that instead of recognizing the continuous speech constituting of many different words and sentences, we will be recognizing a small set of words and label remaining as unknown or silence. The training and test data contains wave files of nearly one second long each one uttering a word, noise or just silence.

The datasets required to train this project are provided by Google as part of a kaggle speech recognition challenge (kaggle).

In the age all digital products attempting to communicate with their customers in terms of speech rather than typing, ASR and NLP have been an interesting as well as important field of study. You can find brief history of research in the field of ASR in this link.

Problem Statement

The problem is to classify all the sound wave files into twelve broad categories of ['down', 'go', 'left', 'no', 'off', 'on', 'right', 'stop', 'up', 'yes', 'unknown', 'silence']

To achieve this,

1. We will classify whether a wave file has any voice or it is just a silent file.
2. For all the non-silent files, we will build a model that can classify whether the sound in the wave file belongs to one of the words mentioned above or some unknown word or sound.
3. The anticipated solution/ model should predict all the 12 categories as accurately as possible in a large sample test set.

Metrics

Kaggle provides a test set of nearly 150000 samples of wave files for which I will predict the labels and create a submission file to submit. Kaggle will provide the accuracy of my predictions based on the submission file.

Apart from this, we will do test train split of training data and check our models performance on training and test data samples with train and validation accuracies.

Since this is a classification problem with a few set of specific labels possible, accuracy is a good metric to gauge our model as we are only concerned about getting the prediction *exactly* right or wrong and all classes are equally important.

II. Analysis

Data Exploration

Our training set contains .wav files with variants of 30 class labels, where each class label is a word. Number of training .wav files provided for each wave is around 2000 on average which is large enough for training a good model. Out of the approximate 64000 .wav files, nearly 4000 files were detected by VAD (Voice Activation Detector) as silent or noise files. We have excluded these from training set. Few of the word pairs in the dataset like (go, no) etc. have very close pronunciation and predictions compete closely with each other as well. Each wave had some silence/noise padded before and after the word for which we had to use VAD and strip off frames of that sort. One such wave for example is provided in Exploratory Visualization section.

Initially we assumed that for some words in test set that are other than these 30 words (among which we have to recognize 10 words as is and other twenty as unknown), the model will probably not return probabilities more than the threshold for any of the labels and hence we can consider that word as unknown. However, since it is hard to find proper threshold of that kind even if that exists, we introduced a new label called “Unknown unknowns” which are some words apart from these 30.

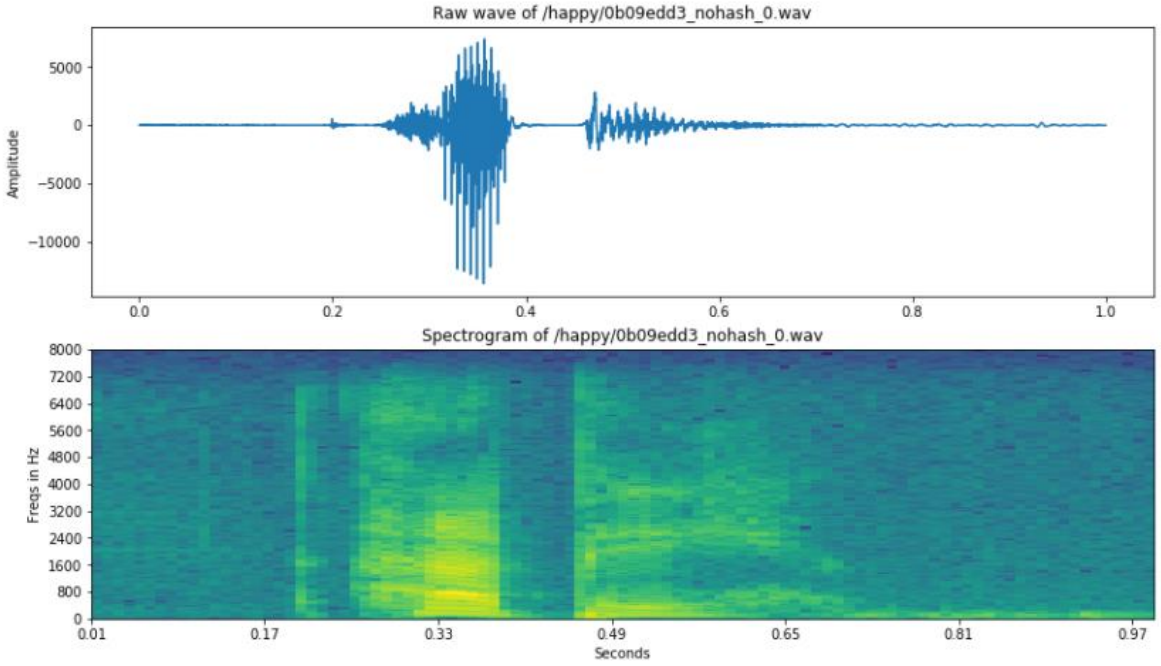
It is probably tough for the model to converge all other words with varied features to aggregate as single class. We also need to prepare some test data with random words to this Unknown unknown's class.

The procedure we followed to create data for Unknown unknown's class is:

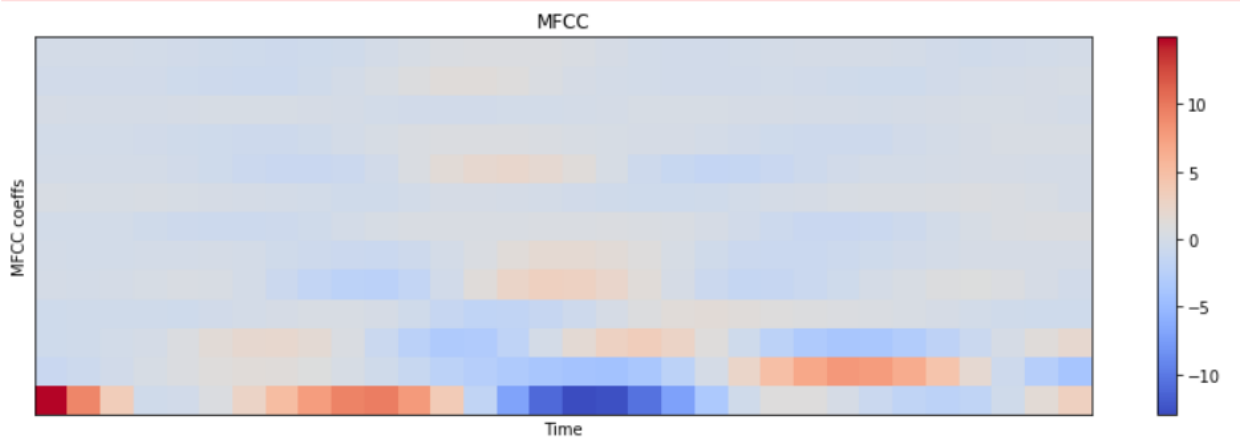
1. Take a wave file from existing training set and find the max amplitude index from that wave's samples.
2. Take the wave from beginning to that max sample value and store it as first half of the final wave required.
3. Take another random wave and pick max sample value to end samples to create second half of the final wave.
4. Merge first half and second half to get a new wave file with some random sound which might not even be an actual word.
5. This data will be a right fit for the unknown unknown's class.

Exploratory Visualization

Plot of raw wave and its spectrogram:



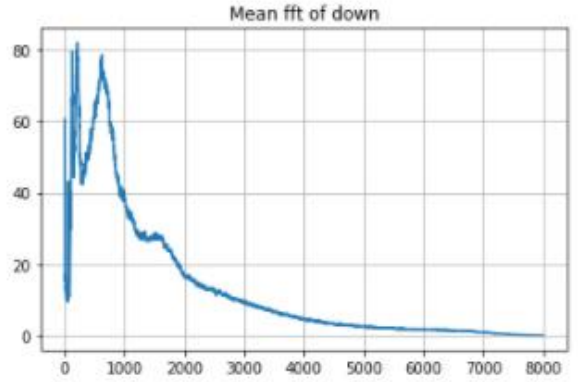
Plot of sample MFCC for a wave file:

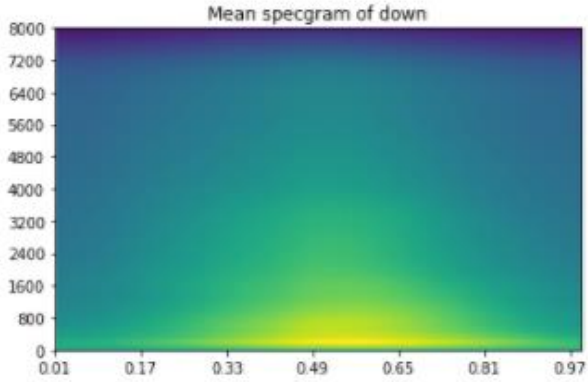


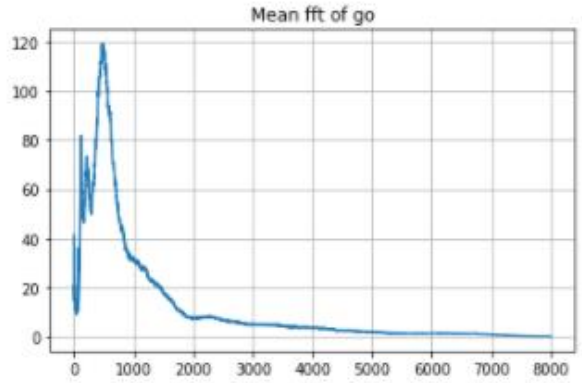
[i]We have plotted mean fft and spectrogram of different words that needs to be

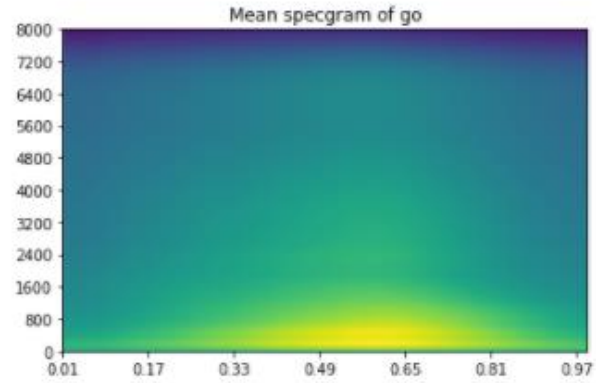
classified, to decide on which features to pick for classification.

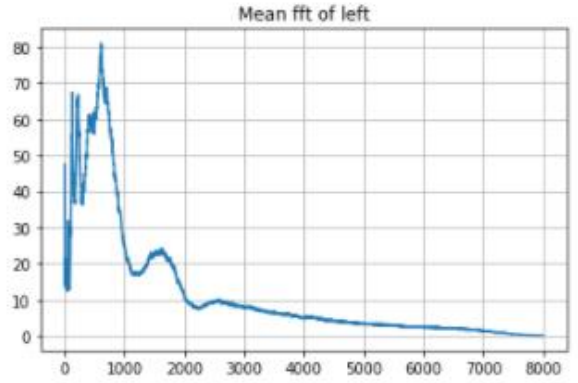
```
['down', 'go', 'left', 'no', 'off', 'on', 'right', 'stop', 'up', 'yes']
```

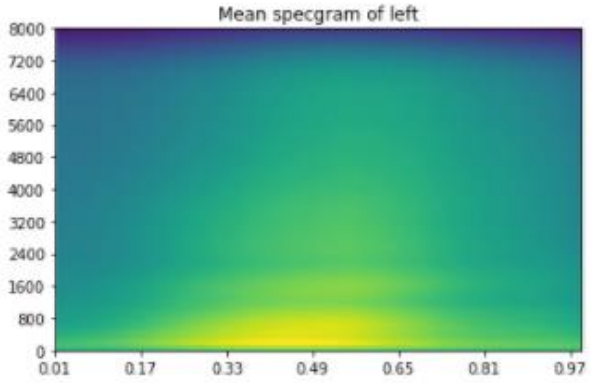


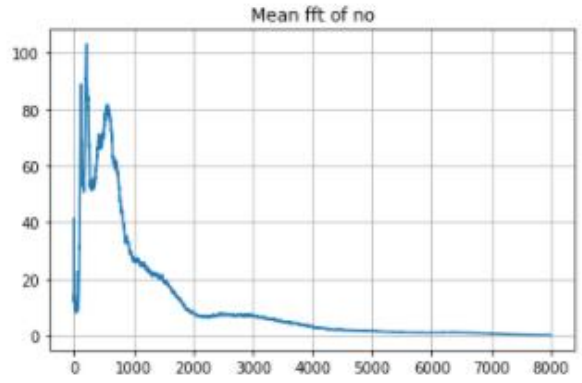


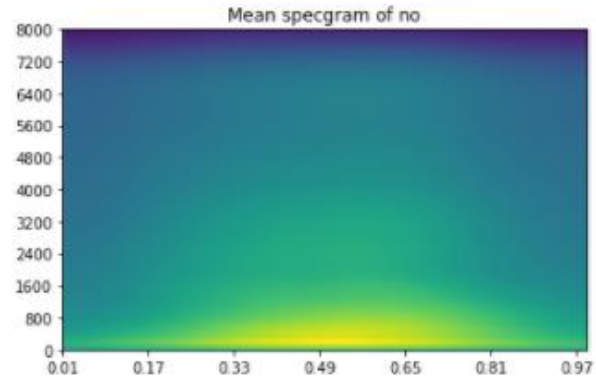


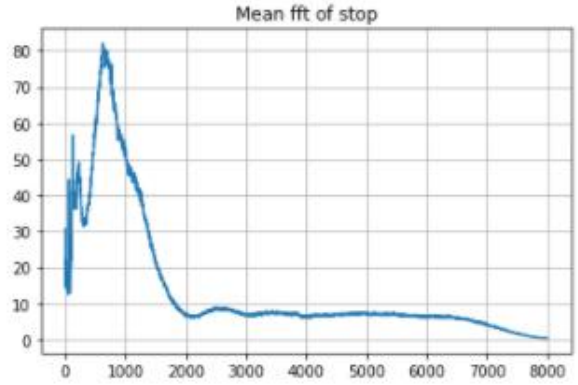


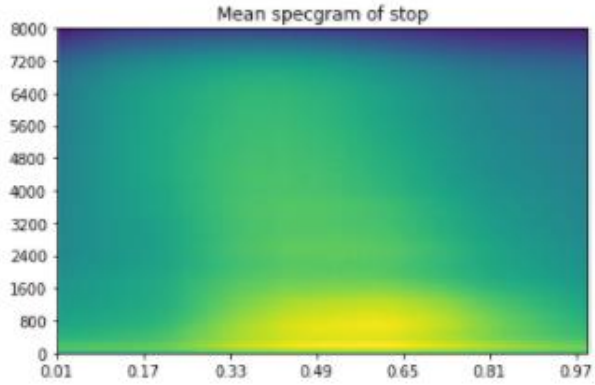


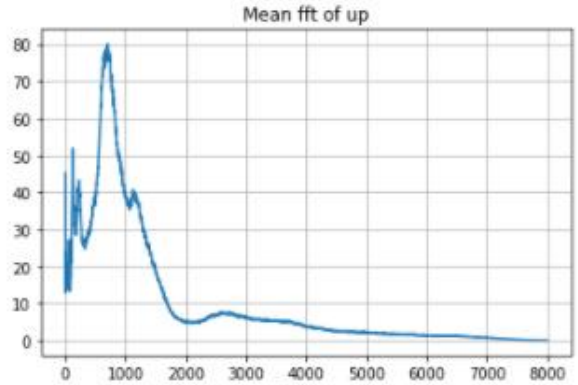


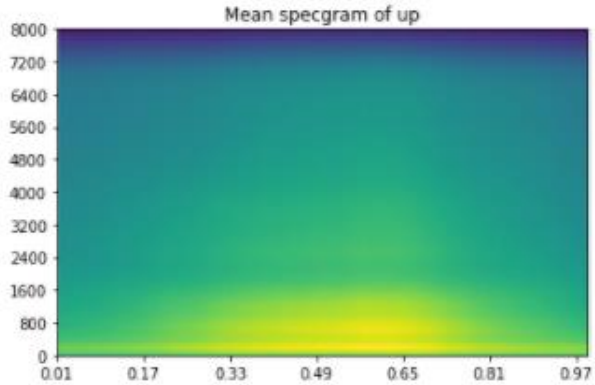


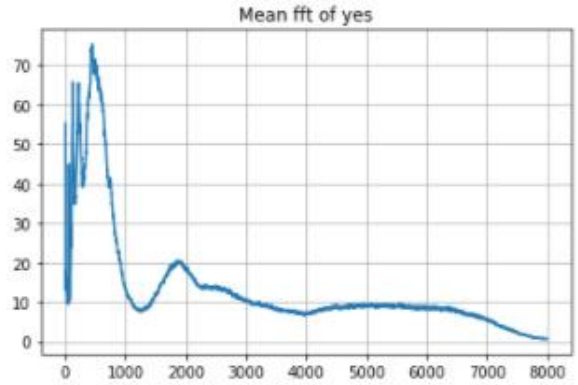


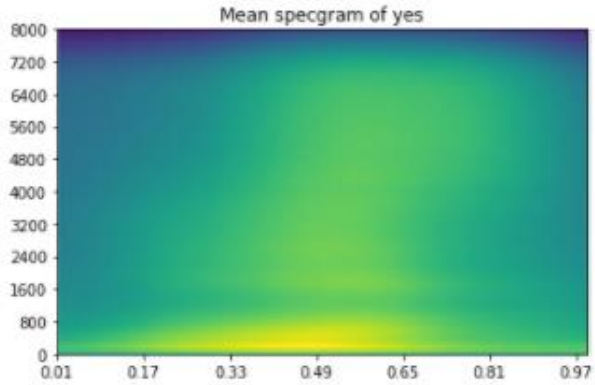




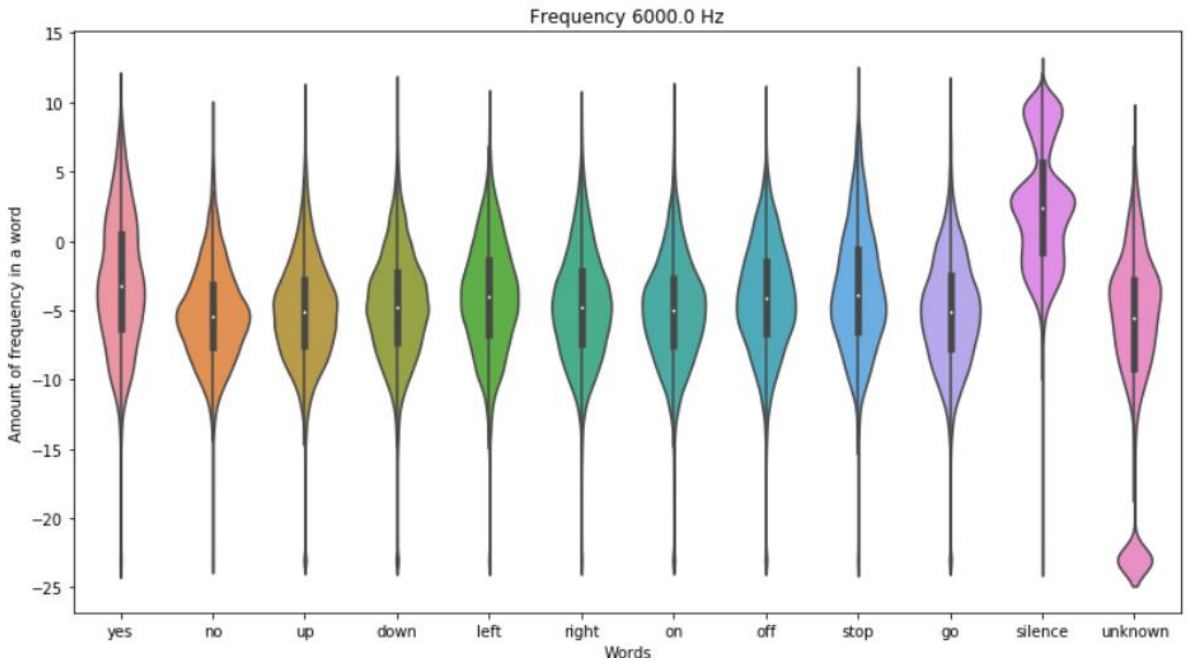








Then we did violin plot to identify amount of different frequencies in each word label of training set.



Algorithms and Techniques

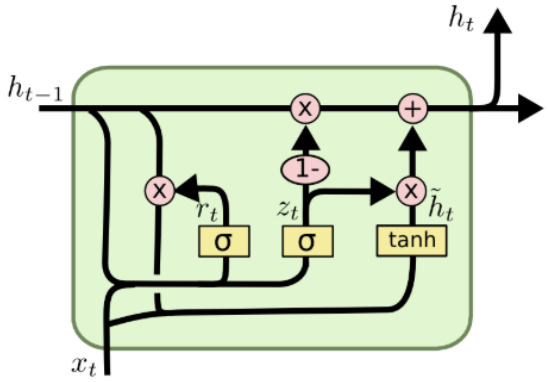
Since speech is a time series where sentences are sequences of words and words are sequences of phonemes, we chose long short term memory networks which can hold the memory of recent phonemes and predict the next phoneme considering the memory and subsequently word.

Though we could have tackled this problem using convolutional neural network, since the problem is restricted to words, eventually when we need to scale it to sentences, it makes sense to approach with LSTM. Though RNNs are good for sequential data, when you need to predict long term dependencies, it is better to use LSTMs over vanilla RNN. LSTMs remember data for longer periods of time comparatively.

A standard LSTM cell consists of:

1. A forget layer which makes the decision what needs to be remembered and passed from previous cell state and what needs to be forgotten.
2. An input gate layer which makes the decision of what part of input needs to be added to the cell state.
3. An output gate layer which decides which part of cell state should be given as output.

A slightly modified version of the same called Gated Recurrent unit.



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

We chose binary cross-entropy and categorical accuracy to train our model as it is a Multi-classification problem to predict our labels. Our labels are one-hot encoded and passed on to model for training.

We have probabilities for each label coming out of the model and picked the label with maximum probability after passing through a threshold value. If none of the labels is more than the threshold, we guessed the word as unknown.

For silence labels, we have passed the wave through the VAD even before passing it model for prediction and labelled accordingly.

The model will be given an input numpy array of size (16, 26) which will be taken by an input LSTM layer where 16 represents sequence of 16 frames and 26 represents the mfcc and delta_mfcc features of each frame.

Benchmark

I have picked the results of my first trained model as the benchmark which contained 12 labels with a dense layer at the output, one LSTM input layer and one LSTM hidden layer. The accuracy of that model against the test set was 0.62.

Test accuracy: 0.62

```
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 16, 39)            8268
_________________________________________________________________
lstm_2 (LSTM)                (None, 26)                6864
_________________________________________________________________
dense_1 (Dense)              (None, 12)                810
=================================================================
Total params: 15,942
Trainable params: 15,942
Non-trainable params: 0
```

III. Methodology

Data Preprocessing

After considering, spectrogram, fft and mfcc for Features to train a model, We chose to build features based on mel scale, which is inspired by how humans process speech in ears. Hence built mfcc features for each wave which is stripped off with silence using Voice Activation Detection.

We have used Librosa library to build mfcc features from a raw sound wave. It includes

1. Converting wave file into smaller frames.

2. Find the power spectrum of each frame
3. Apply mel filter bank to the spectra and sum power inside each filter.
4. Take logarithm of few filterbank energies.
5. Convert them to DCT and pick few important coefficients of the same.

These coefficients are called Mel-frequency cepstral coefficients and state of the art in Automatic Speech Recognition systems.

Later on, considering Speech information could be present in dynamics of spectral frames, rather than just the spectral envelope of frames, we added delta mfcc features as well.

Stacking both mfcc and delta_mfcc features for each frame and doing it for all 16 frames of the wave, we have ended up with features of size (16, 26).

Implementation

Machine Learning Pipeline:

1. Pass wave file through VAD (Voice Activity Detector) to filter out and label all silence files.
2. Pass remaining wave files through chosen feature extractor, MFCCs in this case.
3. Pass the features through stacked LSTM model which can detect patterns across long range of phonemes/frames in the wave file and has the output of 32 labels.
4. Train the model with different parameters with epochs enough for the model to converge.

It took around 30 minutes to train each epoch in my CPU and the model used to take around 15 to 20 epochs to converge. It is very time consuming to experiment each time to train the model.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_9 (LSTM)                (None, 16, 52)            16432
_________________________________________________________________
```

```
dropout_9 (Dropout)          (None, 16, 52)            0
_________________________________________________________________
lstm_10 (LSTM)               (None, 16, 45)            17640
_________________________________________________________________
dropout_10 (Dropout)         (None, 16, 45)            0
_________________________________________________________________
lstm_11 (LSTM)               (None, 45)                16380
_________________________________________________________________
dense_5 (Dense)              (None, 45)                2070
_________________________________________________________________
dropout_11 (Dropout)         (None, 45)                0
_________________________________________________________________
dense_6 (Dense)              (None, 30)                1380
=================================================================
Total params: 53,902
Trainable params: 53,902
Non-trainable params: 0
_________________________________________________________________
```

Refinement

The initial solution had our model with 11 labels, with two LSTM layers and one dense output layer. It had only mfcc features and delta_mfcc were added later. All the words other than the ten words that needs to be predicted are grouped and one label.

Accuracy on test set : 0.63

```
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 16, 39)            8268
_________________________________________________________________
lstm_2 (LSTM)                (None, 26)                6864
_________________________________________________________________
dense_1 (Dense)              (None, 11)                810
=================================================================
Total params: 15,942
Trainable params: 15,942
Non-trainable params: 0
```

Then, we have added delta features, drop out layers for regularization to avoid overfitting. We have changed the number of labels in the output dense layer to 31. This increased the number of trainable parameters to approximately 53k and significantly increased the model training time per epoch.

Initially we were working with threshold value of 0.5, which skipped many words that are recognized around 0.25 and 0.3 max probabilities. After reducing threshold to 0.1, score has crossed 0.7 on the leaderboard test set.

Later on, I have split background noise wave files into multiple 1 sec long waves and used them as well as training data for another model has one more silence label included which makes it to a 32 labelled model.

Using the batch size parameter while fitting the model to training data helped regarding the training time.

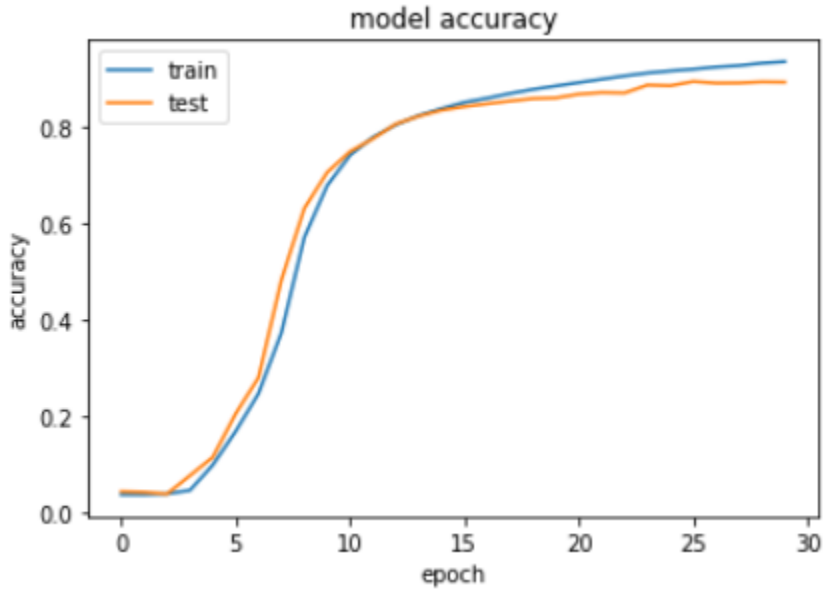
Final model:

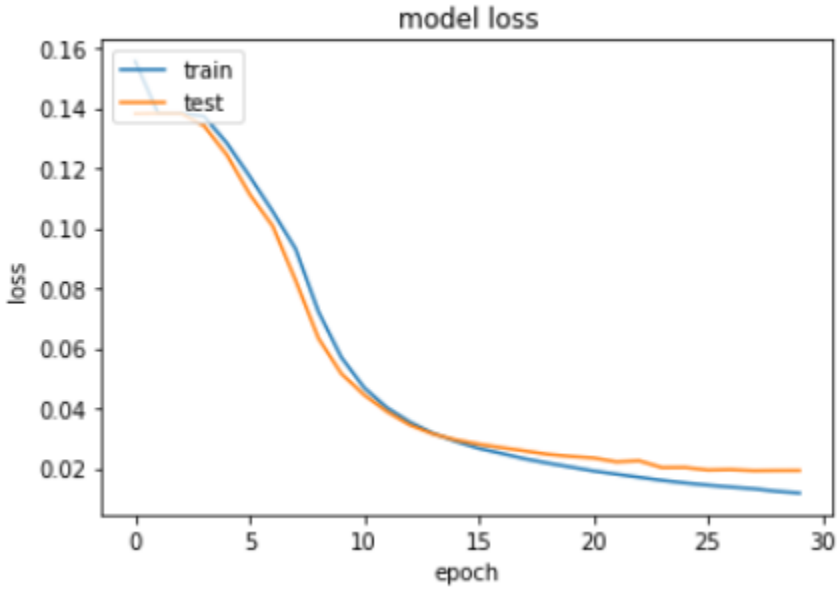
```
Layer (type)                 Output Shape              Param #
=================================================================
lstm_32 (LSTM)               (None, 16, 96)            41856
_________________________________________________________________
lstm_33 (LSTM)               (None, 16, 96)            74112
_________________________________________________________________
lstm_34 (LSTM)               (None, 96)                74112
_________________________________________________________________
dense_18 (Dense)             (None, 32)                3104
=================================================================
Total params: 193,184
Trainable params: 193,184
Non-trainable params: 0
```

IV. Results

Free Form Visualization:

As I've recorded the accuracy and loss of the models per epoch, here's the accuracy/loss graph of the model with batch normalization.





As we can see the training accuracy is near 100% in the diagram and the loss is near 0. Similarly the validation accuracy is also near 95% while the validation loss is around 0.2% near the end of the 30 epochs. We also see the trend where the validation loss reached the lower bound before the training loss. Clearly this model is overfitting on the training data.

Model Evaluation and Validation

The final model has been tested against the test set of 1.5 lakh samples with varied sounds including normal words, meaningless sounds, different kinds of noises, silence samples etc. Hence the score on the test set validates the model to be of reasonably good quality.

Our model is robust enough as we take the sound wave through Voice Activity Detection and strip off the sound wave from unwanted silent frames and pass on only the valid sound frames to the model for prediction. I have created some wave files with words spoken in the different background noises and model was robust enough to predict the words.

I have also hand-picked few predictions from the test set and verified manually to validate my model's predictions.

The final model has reported an accuracy of 0.75 in test set of 1.5 lakh samples and close to 0.95 on training set.

Justification

The accuracy of 0.75 on test set produced by the final model is lot better than the 0.63 score of initial benchmark model.

The final solution had stacked LSTM of three layers, first two returning full sequences while the last layer returns a vector. These layers are followed by dense layers of 32 nodes equivalent to number of labels to predict.

Though it has lot of scope for improvement and experimentation which is described in below section, this score and model is robust and significant enough to do speech recognition for specified labels.

V. Conclusion

Reflection

After trying out with different models and feature data, the best model obtained has the unknown unknowns (words not in training set considered) and have been trained on 32 labels with MFCC features. One more binary classification for detecting Silence in the wave file is employed. We have stripped off silent frames in the audio file before training. Model with LSTM layers considering sequences and memory states with dropout layers for generalization have been fruitful.

One difficult and yet interesting part of this problem is classifying words that pronounce closely and classifying words in dominating background noise which can possibly have multiple interpretations. For examples, go and no pronounce closely and can be easily mistaken in noisy background. However, even humans might not be accurate in this situations but the sentence context can have some information which hints towards a particular word in real word situations.

Improvement

1. The dimensionality of features is huge and try using PCA to remove some dimensions of negligible variance before passing it to the model for training.
2. Though the MFCC features are state of the art in speech recognition based on the literature, we should try other features like raw wave, fft, **log mel** features etc and combinations of different features as well.
3. We should try using few other neural network layers for mfcc features and see how the model performs. Some of them might include:
    a. Convolution LSTM.
    b. VGGbn19.
    c. InceptionResnetV2.
    d. Densenet201 etc.
4. I should try other models like XGBoost (LGBM implementation, as it is computationally effective), random forest kind of ensemble models and see how they perform on this Multi-classification problem.
5. I should either setup or rent GPU to run these models as these will take days if I were to run on normal CPU. I need to make use floydhub, paperspace or vectordash in future.
6. I believe, experimenting all these options with proper computational power will definitely enhance my final benchmark model.
7. Listen to mislabeled samples in validation set.
8. Data augmentation: Add heavy noise augmentation while keeping noise vs signal ratio below 2.

---

[i]

References :

You can find data required for this project in below link https://www.kaggle.com/c/tensorflow-speech-recognition-challenge/data

Some of the Blog posts followed:

1. https://ideasforeversite.wordpress.com/2018/03/18/automatic-speech-recognition/
2. http://practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/
3. Voice Activity Detection
   : https://pypi.python.org/pypi/webrtcvad , https://github.com/wiseman/py-webrtcvad/blob/master/example.py

https://github.com/marsbroshok/VAD-python

https://github.com/marsbroshok/VAD-python/blob/master/detectVoiceInWave.py

Libraries used:

1. For extracting MFCC features
   : https://librosa.github.io/librosa/generated/librosa.feature.mfcc.html
2. Keras : https://keras.io/layers/recurrent/#lstm

Boiler plate: https://www.kaggle.com/davids1992/speech-representation-and-data-exploration

https://dsp.stackexchange.com/questions/13817/how-to-reduce-speech-feature-dimensions