An

Experiment Report

On

Tensorflow Speech Recognition Challenge

By

Jayant Verma

INTRODUCTION

In this report, I will introduce my work for my Deep Learning task. My task is to finish the Kaggle Tensorflow Speech Recognition Challenge organized by Kaggle and Google Brain from 2017 Nov to 2018 January, where I need to build an algorithm that understands simple spoken commands. By improving the recognition accuracy of open-sourced voice interface tools, we can improve product effectiveness and their accessibility.

The dataset consists of about 64000 audio files which have already been split into training/validation/testing sets. Then they asked to make predictions on about 150000 audio files for which the labels are unknown.

My code is available at https://github.com/jayant766/MIDAS-IIITD

Objective

Inquire and explore steps in the data scientist pipeline including: data wrangling, data cleaning, and predictive modeling with AI. I will use the Speech Recognition data set, found on Kaggle.com, in order to build an algorithm that recognizes simple speech commands.

Requirements

- Tensorflow 1.4
- Librosa
- Scikit-Learn
- Python 3

Approach

My approach is broken up into three phases: **Research**, **Modeling**, and **Evaluation**. In the research phase, I will gather and clean the data. I must also determine which type of machine learning models are the best practice for this data set. Search for Kaggle Kernels relating to our topic to avoid mistakes others have made. Then, implement a handful of machine learning models to obtain an accuracy score. Finally, examine our results to determine the precision/accuracy of our models.

Models

Like all other competitions, the idea is to ensemble a few different models that differ in their architecture, data augmentation, feature extraction, etc. Here I list a few of the base model variations I used, all of which are deep neural networks. I considered using Kaldi but since I thought optimising language model and silence probabilities could be tricky I rode the deep NN wave like everyone else.

- A variant of Convolutional LSTM (https://arxiv.org/pdf/1610.00277.pdf)
- LSTM-L (https://arxiv.org/pdf/1711.07128.pdf

- C-RNN (https://arxiv.org/pdf/1711.07128.pdf)
- GRU-L (https://arxiv.org/pdf/1711.07128.pdf)
- Resnet

The features use is MFCC and Audio spectrogram. After initial training on the actual dataset each model was retrained on a combination of the train set and psuedo labelled test set (only predictions with 95%+ confidence were used).

What Worked

- **1D convolutions:** A well designed neural network should be able to learn features from the raw wav files by itself and potentially get more informative features than one could get from a spectrogram. This first model is an attempt to do that. But it is not very effective yet. Some of the mistakes it makes are easy to understand. It often confuses 'no' with 'go' and 'off' with 'up'. Theses words have very similar vowel sounds, so confusion can be expected. Some mistakes are less obvious: does it really think 'right' is 'left' ? Maybe it's the 't' at the end?

Another way of balancing the training set better is by creating batches with equal amounts of samples from each class. I assume one would use the leftover 'unknown' samples in subsequent epochs, but I haven't tried this myself. I don't know which balancing trick is best. After some tweaking of the Conv1D model I got the result.

- **CNN-RNN combination:** The idea is to extract features using a convnet, flatten non-time dimensions, and then use an RNN to exploit the sequential nature of data. I also found about the CuDNNLSTM implementation around then which is an optimized LSTM implementation on Cuda; it really makes a big difference and the API is almost the same as the default LSTM implementation.

It's notable that the 2d conv layers are not that deep and probably more diversity is to be gained by trying other alternatives. Using 1d convs didn't yield as much gain.

- **ResNet**: A residual neural network, or Resnet is basically a deep convolutional NN with shortcuts. Every couple of layers there is an identity connection back to a layer a few levels up. These shortcut connections are thought to help the network generalize better. And this definitely works: I don't need any dropout layers anymore. What happens is that the network depth defaults to using as many shortcut connections as possible. This keeps the network small and therefore helps generalization. This model performed really well.

But it has to do with the test set containing a lot of words that are not given to me in the training set. So called unknown unknowns.

- **Speaker Embeddings:** The dataset for this competition came with speaker labels so I thought it could be helpful to train a standalone speaker identification model and use its bottleck features (embedding) as an additional input when training main keyword detection models.

- **Autoencoder:** I implemented this with a simple not-too-deep 2d cnn encoder-decoder architecture with varying bottlenck shapes, e.g. 128. It gave like 0.01 boost to my early 1d conv ensemble and I used it in my final lightgbm stacker. This part was also extremely slow to train and I resorted to the simplest possible approach I could think of; for example, replacing deconvolution layers with simple upsampling layers.

- **Pseudo Labelling:** The idea is to predict the label of test samples using your best model so far and then retrain/finetune your model using test data and your predicted labels. In the 'hard' version, you take the argmax of softmax predictions and use that as the lable. In the 'soft' version, you use predicted softmax probabilities directly as labels, so a certain test sample could be 50% "on", 30% "off", and 20% a combination of other labels. I think the soft version steps into the realm of knowledge distillation by encoding relationship between labels into the assigned probabilities for your model to learn. I've seen improvements with this method in previous competitions and thought it could be also helpful here.

- **Unknown-Unknwons:** The dev set contains 42k samples from 21 different labels that should be treated as unknown. In practice, however, literally anything can be said instead of main keywords and they should all be treated as unknown. So it's important to find ways to make our models robust against unseen-unknowns in test set or for a better rhym eunknown-unknowns.

- **Ensembling**: What we had at the end was a simple weighted average of different models/ensembles. The idea is to basically flatten all the class predictions into a one-dimensional array and then treat them as a single fingerprint for the model. I then use these fingerprints to calculate model correlations. The hierarchical clustering view helps with identifying diversity between models and choosing the right ones/right weights to pick.

I also tried ensembling my best ensemble with a lightgbm stacker which worked on all experimental predictions.

To make more advanced speech systems

- I must limit the overall computation.
    - I limit the number of multiplications.
        - Limit the number of multiplies is to have one convolutional layer rather than two conv layer
        - Have the time filter span all of the time.
    - I limit the number of parameters.
- Must have a small memory footprint and low computational power.

What Didn't Work

- **Predicting unknown unknowns**: I didn't find a good way to consistently predict these words. Often, similar words were wrongly classified (e.g. one as on).

- **Data Augention**: Something I tried augmenting training data by reversing them, so when someone says "stop" it's actually treated as "stop" but its reverse spoken "pots" is an "unknonw" sample.The idea was to have more unknowns to prevent the network from wrongly mapping unknowns to the known words. For example the word follow was mostly predicted as off. However, neither my validation score not my leaderboard score improved.

- **CLR Schedules:** I used cyclic learning rate but I didn't get the best result.

    - I've tried more extensive augmentation, add different pitch and generated noise, however this gives no improvement. Also not used test set during training, however this might boost the metric.

Result

Public Score	Private Score
0.80208	0.80664

Conclusion

Like all competitions so far, a lot of learning points and challenges along the way, so thank you kaggle and google brain for the organization. It was quite interesting that a lot of techniques that people had learned in other domains like image processing could be applied here too. Although helpful but you didn't need to be a speech scientist to have a chance to compete here.

Also, I heartily thankful to **MIDAS@IIITD** for giving me the opportunity to solve this challenge.

References

- Google Commands Dataset
- Tensorflow Speech Recognition Challenge
- Speech Processing for Machine Learning

- Time Series Classification with CNNs
- EESEN: End-to-End Speech Recognition using Deep RNN Models and WFST-based Decoding
- https://github.com/ARM-software/ML-KWS-for-MCU
- https://arxiv.org/pdf/1610.00277.pdf