Developing a Simple Speech Recognition Algorithm for Open-Source Use: Project Proposal for Udacity Machine Learning Nanodegree Program Capstone

Tyler Sagardoy

Project Domain

Within the past few years, voice command and activation has become a popular trend in user experience and design. Voice command allows for technologies to meet specific consumer demands, such as providing a user interface when hands or vision is occupied – a reason 61 percent of users state why they use voice command – or when one is in the car or on the go – the primary setting for voice usage in 55 percent of instances (Young, 2016). According to Google CEO Sundar Pichai, 20 percent of all queries made in 2016 on Google apps and Android devices were voice searches (Sterling, 2016).

The future looks bright for voice command. Comscore estimates that 50 percent of searches will be conducted by voice by 2020 (Young, 2016). While that figure may be highly optimistic, a study of 39 leading SEO experts identified voice usage as one of the top 3 important trends in search (Alameda Internet Marketing, 2016). While these estimates speak to the Internet search industry exclusively, the enthusiasm for voice activation in this industry portends demand for adoption in other industries.

As independent makers and entrepreneurs develop their products to feature this new and exciting interface method, or learn about it for the first time, many find that their needs for voice commands are minimal – only limited to a handful of basic operational commands. Many robust voice recognition algorithms are extraneous, costly, and unnecessary for many projects. Therefore, an opportunity exists to develop an open-source speech recognition algorithm intended for smaller developers and entry-level enthusiasts.

Problem Statement

The purpose of this project is to provide an algorithm that understands a small selection of simple audio commands. The selection of commands the algorithm should understand should be "Yes," "No", "Up", "Down", "Left", "Right", "On", "Off", "Stop" and "Go". In addition, the algorithm should be able to distinguish silence and unknown commands.

This algorithm is intended to be run on by TensorFlow on a Raspberry Pi 3 by makers, innovators and entrepreneurs in the open-source community to provide some speech recognition abilities to their own projects.

I selected this project, in part, to compete in the TensorFlow Speech Recognition Challenge hosted by Kaggle and sponsored by Google.

Solution Statement

The proposed solution for this problem consists of developing and training a convolutional neural network to accurately predict the library of ten words. The algorithm should be able to take an audio clip, convert it into a tensor that is fed into the trained CNN on a Raspberry Pi 3 computer, and correctly output the command in a usable form.

This algorithm must:

- Be runnable on a Raspberry Pi 3 computer,

- Be runnable as frozen TensorFlow GraphDef files with no additional dependencies beyond TensorFlow 1.4,
- Be under 5MB in size,
- Run in less than 200ms on a stock Raspberry Pi 3 running Raspbian GNU/Linux 8 (Jessie), with no overclocking, and
- Have a standard set of inputs and outputs as defined below.

Some of the principles I will incorporate into the architecture of the CNN include:

- Regularizing and normalizing tensors to reduce overfitting and attain better generalized results
- Minimize the number of layers in the architecture to maintain some simplicity within the model and to minimize runtime
- Use convolutional layers with a wide convolution window and moderate stride to identify speech patterns within inputs
- Use pooling layers to reduce dimensionality between convolutional layers
- Principally, the hyperparameters I expect to toggle as I train and evaluate my models include the convolution window width, the stride of convolution, the number of filters, the type and number of pooling layers between convolutional layers, and the type of activation function that will be present in each layer (excluding the sigmoid activation functions in the output layer)

Dataset and Inputs

The algorithm will be trained and tested using the Speech Commands Dataset released by TensorFlow on August 3, 2017. The data contains 64,727 one-second audio clips of 30 short words. The audio files were crowdsourced by Google with the goal of collecting single-word commands (rather than words as said and used in conversation). A group of 20 core words were recorded with most speakers saying them 5 times. An additional group 10 words were recorded to help distinguish unrecognized words; most speakers recorded these words once. The core words consist of “Yes,” “No”, “Up”, “Down”, “Left”, “Right”, “On”, “Off”, “Stop”, “Go” and the numbers zero through nine. Auxiliary words consist of "Bed", "Bird", "Cat", "Dog", "Happy", "House", "Marvin", "Sheila", "Tree" and "Wow".

This algorithm will take as inputs

- a [16000, 1] float tensor representing the audio PCM-encoded data
- a scalar int32 tensor representing the sample rate, which must be the value 16000

The output will be a [12] float tensor representing the probabilities of each class label as an output from zero to one.

Comparative Benchmark

With 10 command labels and an additional ‘non-command’ label included in the library, a naïve algorithm that outputs only one label would be correct 1 out of 12 times, or 8.3% of the time. For purposes of assessing whether a CNN has any predictive power, I will use this low 8.3% threshold. However, for practical and usability purposes, I hope to train a CNN that has at least 80% accuracy – a threshold that would put me within the top 200 competitors, in the top quartile of the leaderboard, and indicate that my model performs better than 75% of all other models.

Evaluation Metrics

The model will be evaluated using Multiclass Accuracy, defined as the ratio of correct classifications over total classifications. The logic behind this evaluation is rather straight-forward: out of the number of testing cases, how many did the algorithm respond correctly?

Project Outline

The procedural outline is as follows:

1. Gather and load the Speech Commands Dataset from Kaggle
2. Convert dataset files into normalized, standardized float tensor of shape [16000, 1]
3. Divide dataset into training, testing, and validation sets
4. Define model architecture
5. Compile and train the model, then load the model with the best classification accuracy on the validation set
6. Use the test set to calculate Multiclass Accuracy
7. Test the algorithm in the required Raspberry Pi 3 and TensorFlow environments
8. Repeat steps 4 through 7 until a model is found that is superior to both the naïve benchmark and meets the algorithm criteria
9. Once algorithm is selected, prepare the TensorFlow GraphDef file and submit to Kaggle

Works Cited

Alameda Internet Marketing. (2016, 07 06). *39 Experts Share Their Top 3 SEO Trends for 2017 and Beyond*. Retrieved from Alameda Internet Marketing: https://alamedaim.com/seo-trends/

Sterling, G. (2016, 05 18). *Google says 20 percent of mobile queries are voice searches*. Retrieved from Search Engine Land: https://searchengineland.com/google-reveals-20-percent-queries-voice-queries-249917

Young, W. (2016, 06 20). *The voice search explosion and how it will change local search*. Retrieved from Search Engine Land: https://searchengineland.com/voice-search-explosion-will-change-local-search-251776