Capstone Project **Adolfo Moreno**

Machine Learning Engineer Nanodegree

January 31, 2020

Simple words speech recognition system

Definition

Project Overview

In 2017 Google's teams TensorFlow and AIY released to the public the dataset "Speech Commands Dataset", containing 65,000 one-second long utterances of 30 short words [1]. This dataset was aimed to provide a simple but robust base for the development and testing of speech recognition systems. Alongside this, they also released a set o example models [2] and sponsor the Kaggle competition "TensorFlow Speech Recognition Challenge" [3] so developers or enthusiast with low knowledge could start right away working and learning.

Following the bases of the Kaggle competition, the aim of the proposed project is to build a functional speech recognition system able to spot the following set of words: *yes, no, up, down, left, right, on, off, stop,* and *go,* plus two labels for *unknown* words and *silence*, so there are 12 possible labels.

Problem Statement

The approach for the solution for the given problem statement is to build, train and test a TensorFlow model for a classification task, having as input a one-second clip and as output one of the 12 possible labels. The task involved will be:

1. Download and explore the information provided in the dataset
2. Evaluate the different audio features that could be used as input for the model
3. Build an input-pipeline to feed efficiently the data into the model
4. Define some data augmentations techniques
5. Build and test different models, evaluating performance against model hyper parameters and data augmentation

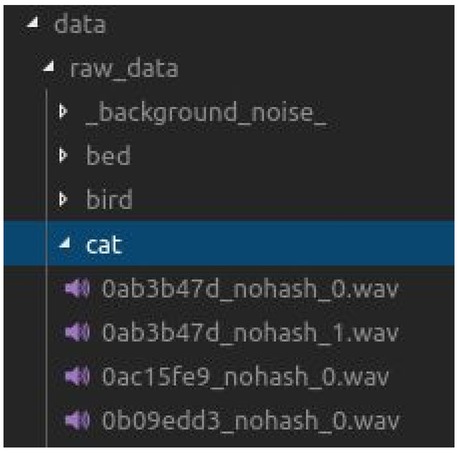
Metrics

Having a balanced dataset, as showed in the data exploration chapter, and following the same metric used to evaluate submissions in the Kaggle's competition, this project will use Multiclass Accuracy, which is simply the average number of observations with the correct label.

Analysis

Data Exploration

The data to be used in the project include the "Speech Commands Dataset, released by google in 2017. As described before, this dataset contains 65,000 one-second long utterances of 30 short words, by thousands of different people, contributed by members of the public through the AIY website.



```
data
  raw_data
    _background_noise_
    bed
    bird
    cat
      0ab3b47d_nohash_0.wav
      0ab3b47d_nohash_1.wav
      0ac15fe9_nohash_0.wav
      0b09edd3_nohash_0.wav
```

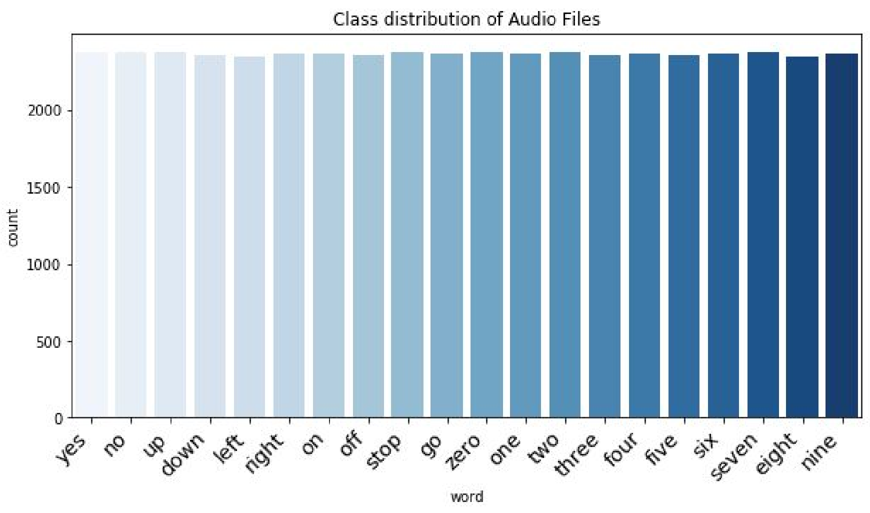
The dataset is organized into 31 folders containing the audio files for each word. There are 20 core words ("Yes", "No", "Up", "Down", "Left", "Right", "On", "Off", "Stop", "Go", "Zero", "One", "Two", "Three", "Four", "Five", "Six", "Seven", "Eight", and "Nine"), and 10 auxiliary words ("Bed", "Bird", "Cat", "Dog", "Happy", "House", "Marvin", "Sheila", "Tree", and "Wow") to help to distinguish between unrecognized words. There is also included a set of files with background noise.

The filename of each file contains information about the user that recorded the audio (first 8 characters), as well as an index that indicate if the word is repeated by the same speaker. This structure is important when performing the train / test / validation split, as we have to ensure homogeneity in the data, for example, keeping the audios from the same speaker in the same set, hopefully Google's team provided a function to perform this slice that we will discus later.
Be aware that the same filename could be found in different folders as the same speaker could have recorded more than one word.

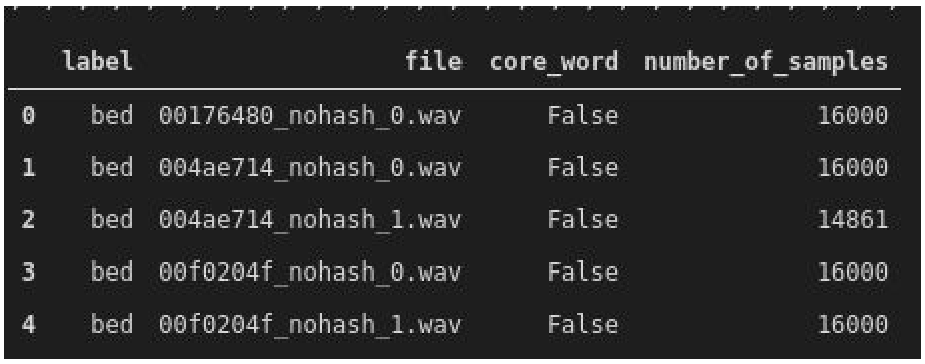
A detailed description of this dataset can be found at
https://www.kaggle.com/c/tensorflow-speech-recognition-challenge/data

Exploratory Visualization

As show in the graph below, the distribution of the target classes in the given dataset is balanced, since all core words, included those we want to classify, have a similar number of samples, around 2370.

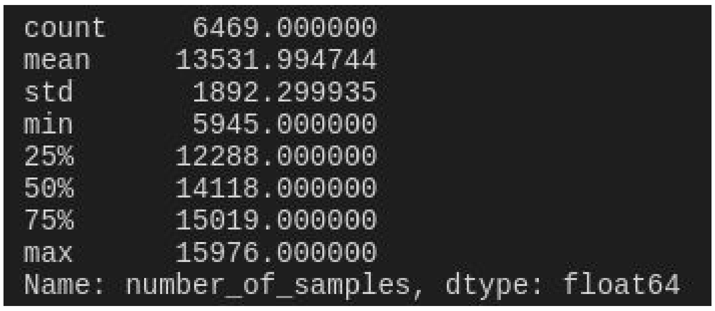


The next table shows the head of a dataset built with the filenames and the number of samples of every file in the dataset. As we can see in the file with index 2, the number of samples is lower than 16000 (sample rate used to record all files), this means that the length of the file is lower than 1 second. Indeed, searching the number of files that do not follow this rule we found that more than 6400 files lay below the 1s. We have to treat specially this files, as we need all the input to have the same shape, one option could be padding the array with zeros.



	label	file	core_word	number_of_samples
0	bed	00176480_nohash_0.wav	False	16000
1	bed	004ae714_nohash_0.wav	False	16000
2	bed	004ae714_nohash_1.wav	False	14861
3	bed	00f0204f_nohash_0.wav	False	16000
4	bed	00f0204f_nohash_1.wav	False	16000

This chart shows the “describe() “function applied over the files lower than 1 second

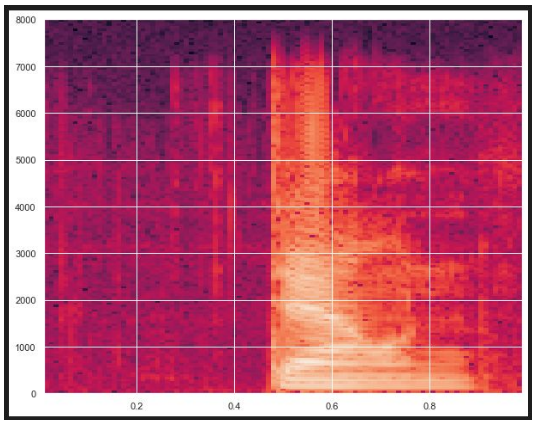


```
count      6469.000000
mean      13531.994744
std        1892.299935
min        5945.000000
25%       12288.000000
50%       14118.000000
75%       15019.000000
max       15976.000000
Name: number_of_samples, dtype: float64
```

Algorithms and Techniques

The first approach for the solution of this project will be to follow the framework proposed by Google's TensorFlow tutorial [2] which implements several neural network architectures including a low latency CNN based on the paper Convolutional Neural Networks for Small-footprint Keyword Spotting [5].

In this tutorial, audio clips are transform into spectrograms, which is a visual way to represent the spectrum o frequencies of the file over time, from this point on the input could be treated as a single-channel image, making the process more familiar since the problem can be handled as an image classification task.



Having this model as a framework the next step will customize it, evaluating the performance of variations for the possible hyperparameters in the training phase and trying different architectures, as light-weight CNN or even RNN.
Finally, having a proper understanding of the solutions it would be interesting to prepare better the input of the model using algorithms to preprocess the audio, trying to handle background noise or the different volume levels. Using data augmentation could also provide a way to make the model stronger.

Benchmark model

As a reference, the accuracy of the top ranked 200 models on the public leaderboard of the Kaggle competition range from 0.87 to 0.91, with the first place having a score of 0.91060. It should be said that the testing dataset for the kaggle's competition is different from the test dataset provided for Google and the performance of the model once predictions are summited tends to be lower.
On the other hand, models implemented in the Google's TensorFlow tutorial [2], which include a low latency CNN, could reach an accuracy of between 85% and 90%, this could be the best reference for my custom model.

Methodology

Data Preprocessing

The input data and preprocessing process implemented follows the next pipeline, mostly inspired in the implementation done in the Google's solution, simplifying the scripts and making several editions to migrate the code to work with TensorFlow v2:

1. Single audio files are read and split into train, test and validation sets using the hashing function provide by Google. Unknown class files are selected from the no core words to have a similar percentage than the other classes. Silence class is populated with files with zero values
2. Data augmentation is implemented, basically these transformations are applied:
    - Background noise is taken from the background files and added in a random percentage of the audios, volume of the noise could be also randomly adjusted.
    - Audio signal is time shifted, this could help the model to handle the fact that words information can be heard right at the beginning or at the end of the 1-second file
3. Preprocessed audio files are packed and saved into Tensorflows TFRecord files. I decided to implement this to improve the performance of the pipeline, as in one TFRecord file we can pack a big number of single audio files, as well handling samples from ' TFRecord Datasets' is more intuitive.
4. From this point TFRecords can be loaded into a 'Dataset' to feed the model. When training dataset is shuffled and random samples are sent to the trainer
5. Spectogram is generated from the audio and reshaped to match the input of the model

Implementation

The implementation process, as described in the '"3_Model_Training.ipynb" Jupyter Notebook, followed the next steps:

1. Create the dataset from the TFrecords files as described in the preprocessing pipeline
2. Define the custom model architectures. Two basic architectures were created in the python module "models.py":
    - "simple_stacked_fc_nn()" Builds a fully connected NN of n depth with intermediate drops layers.
    - "simple_cnn()" builds a three phases Convolutional Neural Network.
3. Define training parameters, optimizer, loss function and metrics
4. Model is trained
5. Performance is evaluated