Machine Learning Engineer Nanodegree

Capstone Report

Raafat Abualazm

09/11/2019

Project Overview

The project is about speech recognition through the use of Convolutional Neural Networks. The input is a wav at first, from which an image like structure, the histogram, is generated, that the Convolutional layers can work with and extract features from in an ordinary fashion. The result of the Convolutional layers is then fed into dense layers to categorise each input into one of 30possible outputs available from the dataset.

Domain Background

Speech recognition is an integral part of human-computer interfaces (HCI). They are present in personal assistants like Google Assistant, Microsoft Cortana, Amazon Alexa and Apple Siri to self-driving car. It is also an easily accessible method of interacting with the computers for people with disability. An accurate speech recognition is vital to avoid unnecessary hardships and to provide a safer experience for critical applications. Past work in this area used something called Hidden Markov Models to predict speech and the command. The project falls under the category of speech recognition. I am personally motivated for this project, because I am working on a NLP project at college and we need a speech recognition step.

Problem Statement

The problem to be solved is rather simple, however, it serves as the basis of any more complicated system. In short, human speech is composed of short utterances and silent sounds, to be able to recognize various commands, one must be able to recognize each utterance, around 1 second each. I, here, confine myself to recognition of each utterance and individual words and maybe very simple sentences. I need to build a model that predicts as much as possible right words from voice. This is a classification problem in which the algorithm takes a sound file and produces an output, a word that is most likely to have been in the file.

Dataset

The dataset used here is Google Speech Commands dataset. It contains over 65,000 utterances, around 1s each, of around 30 words from thousands of speakers, collected through Google's AIY website. The recordings are recorded under various circumstances and from many devices, all in a quiet room though to simulate the actual setting in which speech recognition might be used. It's released under a Creative Commons BY 4.0 license. It is an excellent starting point to learn how to develop a speech recognition model. The dataset will be used to train a Deep Learning model, as intended and authorized by Google.

Solution Statement

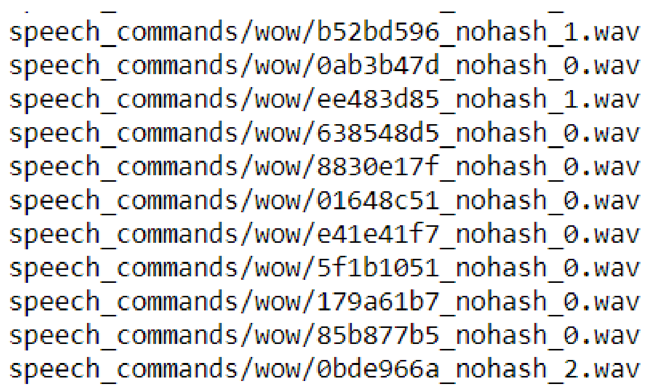
The solution will be quite unorthodox. For this, rather simple, problem, I will be using a CNN to be able to predict the word from its spectrogram. That is, we will visually represent each word with a unique spectrogram and train a CNN on it, which is widely used for image classification. We reduced the speech recognition problem to an image classification problem! The expected output will be a word, that is our label. We have 30 words in this dataset and a successful model should classify utterances into one of these words, our classes, as correctly as possible.

Evaluation metrics

The metric used for evaluation will be accuracy score. That is: score = $\frac{Right\ predictions}{Total\ sample\ size}$ as we are interested in increasing the sheer number of right predictions and the need for other more involved scoring methods is not needed due to the nature of the problem.

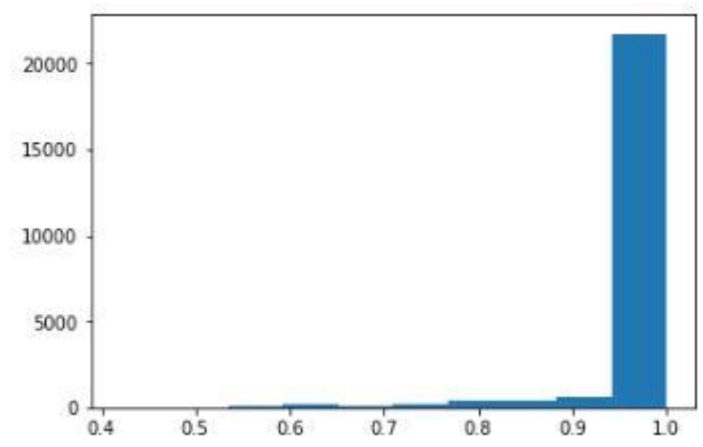
Exploration

The dataset is made of almost 65,000 wav file of 30 words. Where each file is in a folder that is its label. The name of file is produced, possibly, by a hash function and is repeated in different folders, i.e. the names are not unique for each file.

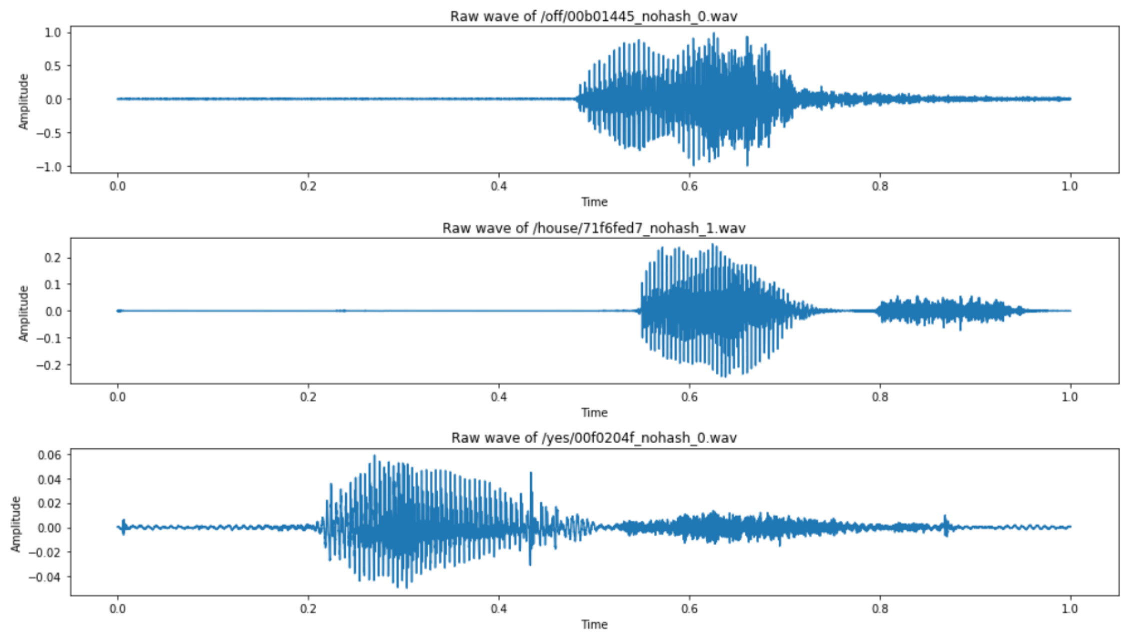


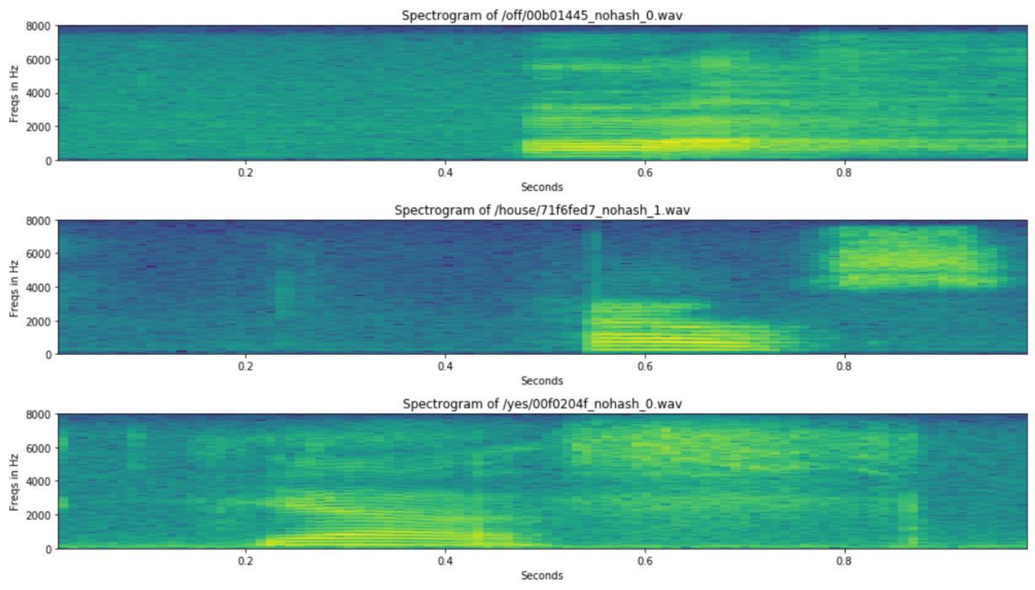
```
speech_commands/wow/b52bd596_nohash_1.wav
speech_commands/wow/0ab3b47d_nohash_0.wav
speech_commands/wow/ee483d85_nohash_1.wav
speech_commands/wow/638548d5_nohash_0.wav
speech_commands/wow/8830e17f_nohash_0.wav
speech_commands/wow/01648c51_nohash_0.wav
speech_commands/wow/e41e41f7_nohash_0.wav
speech_commands/wow/5f1b1051_nohash_0.wav
speech_commands/wow/179a61b7_nohash_0.wav
speech_commands/wow/85b877b5_nohash_0.wav
speech_commands/wow/0bde966a_nohash_2.wav
```

Now the dataset documentation says that each wav file is 1 sec in length and recorded with a sampling rate of 16KHz. However, this is not entirely the truth, as there are some files of length less than 1 sec. as evidenced by this histogram. It is clear the majority are 1 sec but we should make sure to take note of this when processing the data.



Now let’s look at individual files in their wav forms and their resultant spectrograms to have a feel of the uniqueness of spectrograms that correspond to each waveform.





It is quite clear that each wave form produces a noticeably different spectrogram, and as a result CNN will do a great job at identifying and classifying words from sound.

Algorithms and Techniques

The project shall follow the following strategy:

1. Split the data into training and validation datasets using train test split method.
2. Pre-process every wav file to compute its log spectrogram through computing Short Time Fourier Transform then calculating the log for amplitudes. Using Scipy.signal.stft and np.log to produce an image like structure, the spectrogram that can be used as an input to a 2D Convolutional layer.
3. Feed the resultant spectrogram into the CNN to be trained on it. An input layer followed by 4 convolutional layers interleaved by Max Pooling layers, then 3 dense layers that will act as an output layers, the last of which has 30 neurons to match the number of categories. Each layer has the relu as the activation function as the literature says it is the working best in CNNs. As

to why use CNNs, it is an active areaata of research and a viable alternative to RNNs and they are doing very great job working with images of all sorts, so it makes sense to work with them and use spectrograms to represent words.

4. Score the training using accuracy score.

Benchmark model

The model will be benchmarked against Google's Neural Network trained on this same dataset. It achieved an accuracy score of 94%. We can run an accuracy scorer on my model and Google's model and compare the results of each.

Data Preprocessing

- We check the number of samples present to be equal 16000 in a file to make sure the resultant transformations of the preprocessing steps are of suitable dimensions for further processing
- Data is initially a wav file. We first read the wav file and convert it into numpy array, where each index represents a sample and contains the associated amplitude of this sample. We then normalize the amplitudes to be between 0 and 1 and convert the array to be of type float64 to preserve as much as details.
- We take the resultant array and we compute the spectrogram using Short Time Fourier Transform, and this outputs a 2D numpy array where the rows represent frequencies and columns represents time, they contain the amplitude of a given frequency at a given time; in effect, making a visual representation of the sound that the CNN can efficiently work with.

Implementation

1. Reading the file paths through os.walk in the dataset folder to get all available sub-paths and store them in a list
2. Reading every wav file from the its of paths and checking its length
3. Pre-process every wav file to compute its log spectrogram through computing Short Time Fourier Transform then calculating the log for amplitudes. (Using Scipy.signal.stft and np.log)
4. Split the data into training and validation datasets using train test split method.
5. Use Keras Conv2D to make initially 3 Convolutional layers, interleaved between them are Droupout and MaxPooling layers, then a flatten layer which is followed by 2 Dense layers.
6. I used 'adam' optimizer and the loss function is categorical cross-entropy
7. Feed the resultant spectrogram into the CNN to be trained on it.
8. Score the training using accuracy score (Using sklearn.metrics.accuracy_score).
9. Run the validation set and score the result.

Refinement

1. The results of training and validation scores were initially satisfactory, however I decided to add a fourth convolutional layer to extract more features and a third dense layer, it is positioned just before the output layer of 30 neurons.
2. This did not result in a noticeable improvement, it was around 1% improvement.

3. I noticed due to the number of epochs, which was 500, at the end of it, the model tended to overfit.
4. I added made dropout layers a little more aggressive than before, and I added an early stopping call back function to stop the training whenever there was not any change in the validation loss function with patience of 50 epochs to be very sure of this.

   .

Model Evaluation and Validation

The model is based on speech recognition using Convolutional Neural Networks, and this is a very active research area and a direct competitor to the now-famous Recurrent Neural Networks. The final model was derived through successive improvements and adding layers one by one to achieve the desired output. It produces fairly consistent and reliable results.

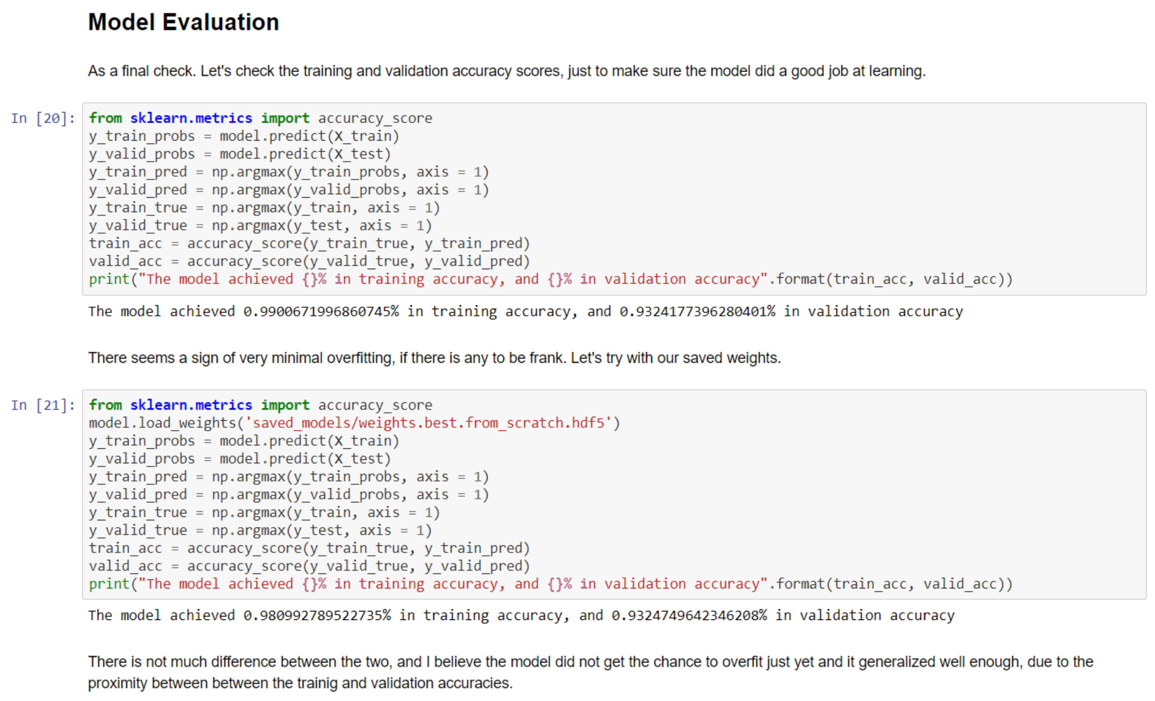
The model achieved 98% in training accuracy and 93% in validation accuracy. I believe it is a very good result and worthy of being trusted as the variation between training and validation scores are narrow enough to hint at decent generalization.

The model is robust enough, because when I changed the parameters a little and even its architecture a very little bit, the results did not vary widely. I, also, trained it with fresh random weights multiple times and I got very close results and as such it is a fairly robust and stable model.

Justification

The model scored 98% testing accuracy and validation accuracy of 93%, which is very close to Google’s benchmark of 94% and this is really an outstanding result. It surpassed it in some aspects and came very close to it in some other aspects. A relatively simple, like presented in this project to score this high indicates that it is reliable and lightweight. This can be used on low-processing-powered device without internet connectivity to do the recognition. I believe, that this model is able to solve the problem initially proposed.

Freeform visualization



Model Evaluation

As a final check. Let's check the training and validation accuracy scores, just to make sure the model did a good job at learning.

In [20]:
```
from sklearn.metrics import accuracy_score
y_train_probs = model.predict(X_train)
y_valid_probs = model.predict(X_test)
y_train_pred = np.argmax(y_train_probs, axis = 1)
y_valid_pred = np.argmax(y_valid_probs, axis = 1)
y_train_true = np.argmax(y_train, axis = 1)
y_valid_true = np.argmax(y_test, axis = 1)
train_acc = accuracy_score(y_train_true, y_train_pred)
valid_acc = accuracy_score(y_valid_true, y_valid_pred)
print("The model achieved {}% in training accuracy, and {}% in validation accuracy".format(train_acc, valid_acc))
```
```
The model achieved 0.9900671996860745% in training accuracy, and 0.9324177396280401% in validation accuracy
```

There seems a sign of very minimal overfitting, if there is any to be frank. Let's try with our saved weights.

In [21]:
```
from sklearn.metrics import accuracy_score
model.load_weights('saved_models/weights.best.from_scratch.hdf5')
y_train_probs = model.predict(X_train)
y_valid_probs = model.predict(X_test)
y_train_pred = np.argmax(y_train_probs, axis = 1)
y_valid_pred = np.argmax(y_valid_probs, axis = 1)
y_train_true = np.argmax(y_train, axis = 1)
y_valid_true = np.argmax(y_test, axis = 1)
train_acc = accuracy_score(y_train_true, y_train_pred)
valid_acc = accuracy_score(y_valid_true, y_valid_pred)
print("The model achieved {}% in training accuracy, and {}% in validation accuracy".format(train_acc, valid_acc))
```
```
The model achieved 0.980992789522735% in training accuracy, and 0.9324749642346208% in validation accuracy
```

There is not much difference between the two, and I believe the model did not get the chance to overfit just yet and it generalized well enough, due to the proximity between between the trainig and validation accuracies.

At some point I have suspected that the patience of early stopping call back is a little large, and I had a question how much does the model learn as the epochs progress in the later stages. So, I decided to let the model run till it stopped and save the model whenever the loss function of the validation improved and run a final test to see what is the difference between the two, after all they are 50 epochs apart.

The results were surprising to me at first, why there is not much difference between the two states? I know we are close to 100% and variations in the testing are not expected to vary largely, but this was smaller than expected, and why did the accuracy score of the validation vary between the two states, albeit in a very trivial manner, while both have the same loss function. I concluded that the model did not overfit and learn any thing in particular between the two states and my patience value was reasonable in this case.

Reflection

The process of project can be

1. Load the paths into a list and iterate through each entry to read the file, check its length and extract the label from path
2. Pre-process every wav file to compute its log spectrogram through computing Short Time Fourier Transform then calculating the log for amplitudes.
3. Split the data into training and validation datasets using train test split method.
4. Feed the resultant spectrogram into the CNN to be trained on it. (An input layer followed by 2 to 3 convolutional layers interleaved by Average or Max Pooling layers, then 1 to 2 dense layers that will act as an output layers)
5. Score the training using accuracy score.
6. Run the validation set and score the result.

The hardest part for me was searching for a problem that I am interested in, and will help me in my current projects as well, and for which I can find public datasets. So, the speech recognition problem was the winner candidate here. I also needed some time to figure out how to import all the files and extract the labels from the path. The problem is very interesting as its one of the stepping stones towards conversational AI and general AI. The most surprising aspect was for me that I can use CNNs as a solution, as I had the impression it was only used for images, well the bridge was to compute the spectrogram and thus having an image that corresponds with the sound file.

Improvement

There is always some place for improvement. I believe that the improvement can be summarized into

- Improvement of the architecture, like number of layers, numbers of neurons, sizes of filters and experimenting with other activation functions.
- Having a larger more varied dataset. To capture more words in more diverse settings and from a larger set of speakers to be more universal
- Create a pipeline to deploy the model into production. I suggest an App that will create the sound file array representation using PCM-16 and slice it into files of 1 second length each. This will allow for a creation of an API that can accept the array and run al the preprocesssing and have the ready to use model in it and output the word uttered.