CS230

Who just spoke in the conference call?

Adrian Ng
Stanford University
adrianng@stanford.edu

Abstract

Many of us have this problem during meetings - we call into a meeting, we hear what is being talked about but, we don't actually know who is talking. This project aims to build a Neural Network that can recognize any speaker's speech characteristics such that it can identity that speaker during ANY conversation. This is a challenging task for many reasons, the primary ones being varying background noise and the quality of recorded speech even for the same speaker. Another challenge is we don't have enough data to capture a wide range of speech characteristics for any one speaker. This report describes how I used a CNN (Convolutional Neural Network) architecture similar to VGG-16 to identify a speaker. In my experiments, I saw a success rate of 67% in predicting a particular speaker in a conversation. In this report, I also describe a data augmentation technique that I used to boost prediction accuracy by 15 percentage points from 49% to 64%. Then I did more hyper-parameter tuning to get up to 67%.

1 Introduction

A lot of us have this problem during meetings - we call into a meeting, we hear what is being talked about but, we don't actually know who is talking. This is a very prevalent problem in business, but unfortunately, there is no existing commercial solution. Having software that identifies a speaker in real-time will help the audience better understand the discussion and improve meeting productivity.

For this project, I built an algorithm that could identify in real-time who was doing the talking in a conversation. To simulate an environment to develop such an algorithm, I produced audio files by converting YouTube videos. Converted audio files capture real-life conversations (e.g. 2016 Presidential debate between Trump and Hillary). Then I broke up the audio files into 5-sec audio clips and fed them in sequence to my model. For each of the 5-sec clips, my model made a prediction of who the speaker was. And this is how I simulated real-time predictions.

My model takes pre-processed versions of the 5-sec audio clips as inputs and makes a prediction on who the speaker is. Internally, my model uses a 4-layer CNN network followed by a Softmax layer, to make a prediction. My CNN architecture has similar characteristics to VGG-16 [1] as I wanted to simplify the hyper parameter-tuning process for CNN network. (Details about preprocessing and Neural Network architecture will be discussed in subsequent sections.) As a side note, I experimented with different audio clip sizes and 5 seconds represented the best trade-off between model performance and real-time requirement. Presumably this is because, in real-life speech, we tend to pause and speak at different rate; 5-sec is a reasonable time window for the model, and even for human, to capture speech characteristics of a speaker.

CS230: Deep Learning, Fall 2019, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

2 Related work

This project falls into the sub-field of speaker identification of speech recognition. I reviewed a few papers in the literature such as [1], [2], [3], [4], [5]. While [1], [3], [5] provided good overview and interesting algorithms on how I could tackle this problem, I examined and evaluated GitHub projects for [2] and [4] as they are more relevant to this project. I started with evaluating [2], for which the author also tries to solve the same real-time speaker identification problem. His approach uses various signal-processing-based pre-processing techniques such as LPC, MFCC, VAD, followed by Gaussian Mixture Model (GMM) to make predictions. However, I have found the model sensitive to noise and changing of test set; and did not perform well for my initial dataset. I did a performance comparison between my CNN-based model the GMM model. which will be described in section 5. I then evaluated [4], which also uses Convolutional Neural Network for speaker identification. Such a model has a similar basis to what I was planning to build and had a reasonable performance for my test set, as a result, I use such GitHub project as the basis for this project.

3 Dataset and Features

Data source: To successfully predict individual speaker in a conversation, subject to noise and different backgrounds, there are two key factors that determine the model accuracy. The first factor is the model needs to be able to "distinguish" and identify features of speech (e.g. intonation, pitch) for different speakers. For that, we needed to train our models with a sufficient number of unique speakers. The second factor is the model needs to "distinguish" voices from background noises (e.g. echo in the room, people coughing). For that, we needed to train our model with sufficient background noises. As a result, I am using VoxForge and YouTube as audio file data sources.

VoxForge: VoxForge is an open-sourced human-recorded speech library with speakers who read text from a book and record the speech using a computer microphone. Each speaker has a roughly 1 min long speech, broken into audio clips between 4-sec to 9sec long, and there are roughly 900 speakers. I have about 900 minutes of speech data from VoxForge.

YouTube: YouTube is an excellent data source of conversational speech. For instance, one example audio file is the YouTube video Presidential Debate between Trump and Hillary. I captured an audio version of it in wav format. Then I broke up the wav files into 5-sec audio clips. Each of the 5-second clips serves as a training/testing sample. From YouTube, I got 150 speakers and each has roughly a 2-min long speech. The reason why the YouTube speech is longer than the VoxForge speech is they are real-life conversation rather than people reading text from a book. In real-life conversations, people tend to speak slower and have pauses, therefore, I wanted to make sure the audio was long enough for my Neural Network to capture speech characteristics.

Table 1 summarizes my data. In total, I have 1200 minutes of speech data from 1050 speakers before data augmentation. After data augmentation, I have 3600 minutes of data. For train/dev/test set, I used a 90/10/10 train/dev/test ratio split among the 5-sec clips.

	Number of unique speakers	Total Speech length (mins)	Average speech length per speaker (mins)	Average audio clip length (secs)
VoxForge	~900	~900	1	5
YouTube	~150	~300	2	5
Combined audio	~1050	~1200	1.14	5
Combined audio including augmented audio	~1050	~3600	3.42	5

Table 1: Audio Data Distribution Summary from different audio sources

Data-augmentation: After some early experiments described in section 5, I found the data that I gathered from VoxForge and YouTube was not sufficient enough to train my model. (Details of the experiment and the results will be described in section 5.) As a result, I augmented my data by capturing the noise from the YouTube audios (which were richer in background noise) and created new speech samples for each speaker by overlaying the noise on top of the original 5-sec audio clip. More specifically, in the first step, I removed the speech portion of the audio (using a Pydub API that I created) and kept the "background audio" in a conversation setting. After this, I roughly had 150 minutes of audio that captured background noise in various settings. In the second step, for each of the speakers, I created another 2 minutes of augmented audio where I randomly picked noise from step 1 and overlaid this on top of the original speech. So, for each speaker, I now have 2 more minutes of additional speech. It turns out that this technique increased my model accuracy by 15 percentage points.

Preprocessing and feature extraction: To minimize audio property differences, all audio input files are first converted to 5-sec clips in wav format. Then, they are normalized to the same audio properties – same sampling rate is at 16Khz, same max amplitude in DBFs, etc. Then, the audio is converted to MFCC (Mel-frequency cepstral coefficients) for feature extraction as shown in Figure 1. MFCC can model human audio features more closely than typical audio waveform.

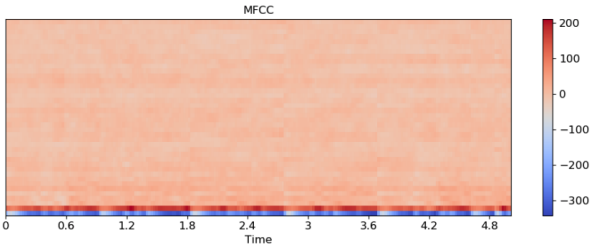


Figure 1: MFCC diagram captured by using librosa library [6]

4 Methods

I use CNN (Convolutional Neural Network) as my model. The reason is this task of speaker identification is similar to image classification in the sense that, we want the model to identify pattern in the input speech by learning the parameters of convolution filters for proper shape detection. A final Softmax activation layer is used to compute probability of each speaker. For instance, to train his model, inputs parameters to my model has a dimension of [samples, 20, 160, 1] and the label has a dimension of [samples, speakers]. The model uses one-hot vector for labeling and scoring. Figure 2 shows the network diagram for this model. My model has 6 million parameters.

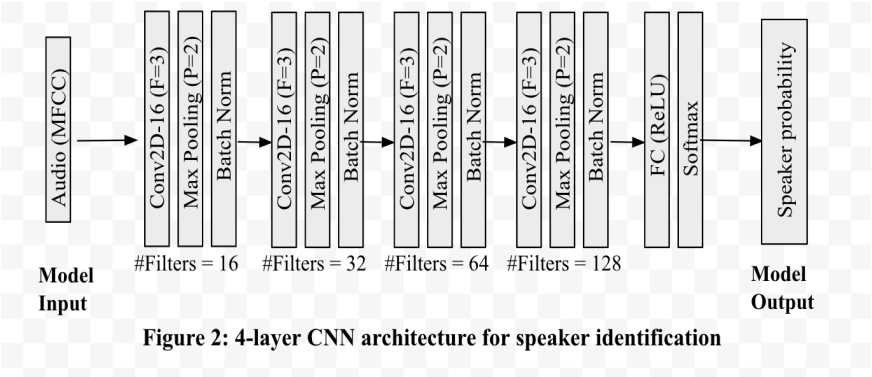


Figure 2: 4-layer CNN architecture for speaker identification

My network resembles the VGG-16 network architecure [1] taught in class in the sense that my network uses "same" filter and doubling number of filters as the Network goes deeper. Despite the fact this architecture tends to have more hyperparameters, it works well and reduces efforts needed for tuning hyper-parameters.

In my model, I use cross-entropy loss function which is described below. This is a reasonable loss function for a classification problem like my use case. Cross-entropy loss increases as predicted labels diverges from the true labels. This penalizes bad predictions. During the training process, as the training goes through forward propagation and backward propagation, the model will identify a set of parameters that fits the training data well with the lowest cost.

$$Loss_{crossentropy}(y, \hat{y}) = -\sum_{i=1}^{N} y_i log(\hat{y}_i)$$

$$Softmax(y_i) = e^{y_i} / \sum_{j=1}^{N} e^{y_j}$$

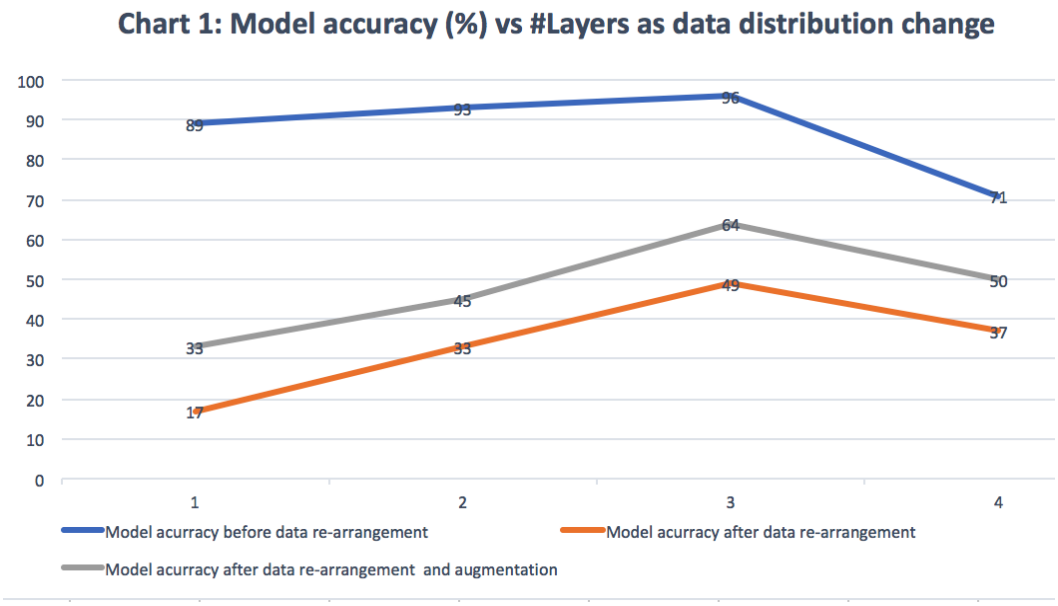
In my use case, I assume each 5-sec audio clip would have one speaker only. In real-life conversation, of course, it is expected a 5-sec audio clip would capture times when multiple speakers are speaking but I do not expect my model to detect that case (to cover that case, other signal processing techniques can be applied to filter these clips before feeding them to the model).

5 Experiments/Results/Discussion

Evaluation metrics: Recall that I have 90/5/5 train/dev/test ratio. I used accuracy as my metrics. The reason for that is my training data is balanced enough in that each speaker has a similar share of training data. Second, there is not a high-cost associated with a false positive or false negative sample in my use case, unlike a fraud detection case. Accuracy is as defined as the following:

$$Accuracy = Number\ of\ True\ positives / Number\ of\ predictions$$

Result Discussion: As shown in Chart 1, with 4-layer CNN, my model gets the best performance with 96% accuracy. This means that given a 5-sec audio clip, our model has a 96% chance of correctly predicting the speaker.



Initially, I was excited by the high accuracy of the result. However, on a closer look, this was not the correct test given what I wanted to achieve. For instance, suppose I have a 30-sec speech from Trump and I break it down into six 5-sec clips.. My train/dev/test data could be partitioned in a way that 4 clips out of the 6 to go to the training set and 2 clips to go to the test set. In this scenario, a prediction accuracy of 96% mean that it has a 96% chance of correctly predicting that the 2 clips in the test set

are Trump. This was not exactly what I was after since all 6 clips belong to the same speech sharing the same background characteristics.

Instead, what I want to achieve is - given a speech sample from a person like Trump, when this person speaks again in another independent conversation, predict that the same person is speaking. The way I originally partitioned the data does not capture this distinction of independent conversations.

Test set re-arrangement: To fix this, I re-partitioned my train/dev/test data according to independent conversations from the speakers. More specifically, the inputs to the model are still 5-sec clips. In my data, each speaker still has 1 or 2 minutes of speech before any augmentation depending on the data source. However, I re-arranged the data such that each speaker would have 3 speeches in 3 independent settings - e.g. in my data set, Trump has a speech in the 2016 Inauguration ceremony, one speech in 2015 before he became the US President and one speech in 2019 about China. The 3 speeches still has a total length of 2 minutes. Then, I did a 90/5/5 split among individual speeches for each speaker. After this split, I could have the 2015 and 2016 speeches from Trump in the training set and the 2019 speech in my test set. By re-arranging how I split up the data, the accuracy metric indicates how well the model can identify a speaker when he gives a different speech.

After the data re-arrangement, the result, as shown in chart 1, has an accuracy of 49% It indicates that my model does not predict speakers from independent speech well.

Error Analysis: What the results tell me is that 1) my model can predict the speaker of a clip from the same speech really well (96% accuracy) but 2) my model does not predict the speaker from a different speech well enough (49% accuracy). After some analysis, my conclusion is that my model could not generalize speech characteristics of a speaker well enough. Hence, my model cannot distinguish speaker speech characteristics from the background noise. This is due to lack of data.

Error Mitigation: To address this issue, I used the data augmentation technique described in section 2 to increase the data set size. The goal of data augmentation is to train the network so that it can distinguish background noise. With that, I got this result described in section 1, which had a 15 percentage point increase in accuracy. This is a significant boost to my model performance.

Hyper-parameters tunings: The first hyper-parameter that I tuned was the number of layers as shown in chart 1. Once I reached 64% with the 4 layers, I started to tune other hyper-parameters to get to a higher accuracy. In the tuning process to get to the lowest model cost, I searched through different activation functions such as ReLU, Tanh and Sigmoid, different optimizers such as ADAM and RPMSProp, different batch size such as 64, 128, 256 and various dropout rate. At the end, I used ADAM, batch size of 128 and ReLU activation and dropout of 0.2. After using the more optimal hyperparameters, my model got to an accuracy rate of 67%.

Performance comparison with the GMM model [2]: In section 2, I described how another project tried to solve the same real-time speaker identification problem with a GMM model. I did a performance evaluation. With my data set, I found such GMM model only had an accuracy of 31% whereas my CNN-based model reached 67%. By examining the code and performing error analysis, I found that such a GMM model relies on a lot of hand-crafted signal-processing features with another set of hyper-parameters (e.g. LTSD window size, picking a default noise signal for noise filtering, etc). It is possible that I did not tune the GMM model properly to my data set and hence the GMM model did not perform well. But it appears that too many hyper-parameters can make the model fragile. This can explain why the GMM model is very sensitive to noise.

6 Conclusion/Future Work

In this paper, I described a CNN-based network that has a 67% accuracy in predicting speakers in an independent conversation setting. In addition, I described a technique that resulted in 15 percentage point gain in accuracy. This shows that the biggest challenge was due to lack of data. Because of that, the model could not generalize and distinguish speech features from background noise. To improve accuracy, we need more data so the model can capture speakers' speech characteristics better. However, I know this is not always feasible.

If I had more time, I would try to use Siamese network and Triplet Loss to train the algorithm, because I'd like the model to be trained really hard on computing differences among speakers' speech characteristics and among various background noise. I think it will lead to better prediction accuracy.

7 Contributions

I am the only person who worked on all aspects of this project, which include these – researching prior work done in this field, evaluating prior work and performing initial analysis on GitHub projects, data collection, data-pre-processing, experimenting with different neural network architectures, implementing the data-augmentation technique, hyper-parameter tuning, writing this report, etc.

References

[1] Very Deep Convolutional Networks For Large-scale Image Recognition (*https://arxiv.org/pdf/1409.1556.pdf*)
[2] Digital Signal Processing: Speaker Recognition (*https://github.com/ppwwyyxx/speaker-recognition/raw/master/doc/Final-Report-Complete.pdf*)
[3] Intelligent Voice Identification with Neural Networks (*https://cs230.stanford.edu/files_winter_2018/projects/6924104.pdf*)
[4] Speaker Identification: Text Independent Context (*https://cs230.stanford.edu/projects_spring_2018/reports/8270111.pdf*)
[5] Kera Face (*https://github.com/chen0040/keras-face*)
[6] Open-source librosa library for MFCC transformation (*https://librosa.github.io/librosa/*)
[7] Python Pydub API (*https://github.com/jiaaro/pydub*)