Speaker Diarization Using Siamese Network

Submission Date: 12/13/2017

Christopher Kinsey
Munibun Billah
TJ Ferrell

Overview:

Introduction:

Speaker diarization is the process of splitting a conversation into homogenous segment according to speaker identity.The goal of diarization is to flag when in the audio a particular speaker is speaking. This is commonly used for speech to text recognition, speaker recognition, and information retrieval from audio or video.

Reasoning:

This problem has been previously solved with 85% accuracy using only decibel analysis of the tracks. We reasoned that rather than using a neural network to classify this data from scratch, we would use decibel analysis as a starting point and use a neural network to make classifications on the segments where the decibel analysis failed.

After the decibel analysis, we converted the audio data to spectral data using a discrete Fourier transformation. The spectral data showed the frequency decomposition in small time frames. This was useful for two reasons:

1. Sounds are most distinguishable from each other due to their different frequency compositions. Frequency compositions are difficult to recognize in waveform format, but are shown directly by spectral analysis. This method puts a more powerful representation of the audio into the network using fewer data points.
2. Using spectral analysis reduced the breadth of the input, while increasing the dimensionality. The waveform format used 44100 samples per second across a single dimension. This means that a wider kernel would have to be used to catch lower frequencies, resulting in less accuracy, more parameters to train, longer training time, and a greater need for more training data. A convolutional kernel sliding across spectral data, however, would have one dimension for time, and a second dimension for frequency. This means that any given cell in the kernel must generalize across time still, but it only needs to learn a single frequency.

Since different voices have different characteristics, we had to either choose to train the model on a subset of the possible vocal varieties and hope that the subset was sufficiently representative, or have a network that learned specific characteristics of the speakers' voices in each audio track. We opted for the latter, which meant that the network's learning would be done on every track it diarized. Siamese networks have proven to be an optimal choice for one-shot learning in image comparison, so we adapted them for this problem.

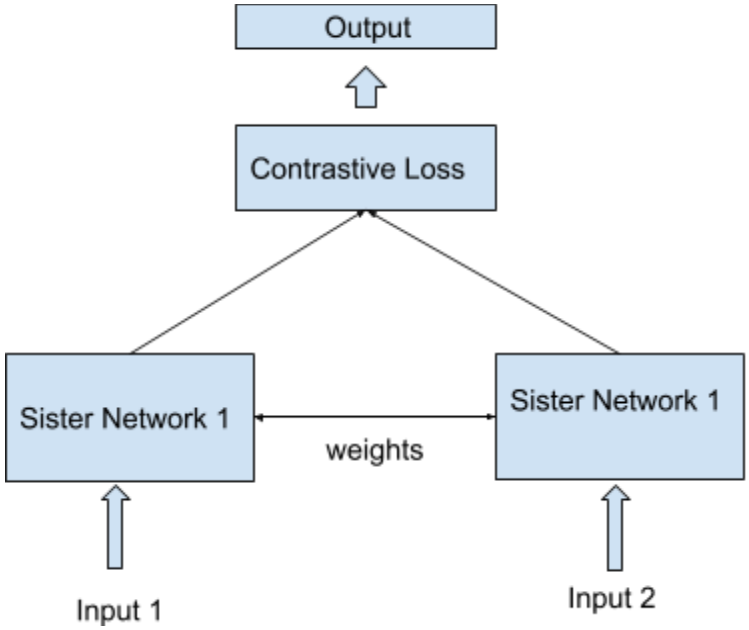
Siamese Network:

A Siamese network is a unique application of convolutional neural networks in which the network attempts to compare two samples in order to determine their similarity with minimal training. It does this by using two identical networks with identical weights. By taking these

networks and feeding unique inputs into each network, the siamese network then can use a special contrastive loss function to determine the input's similarity. The contrastive loss function is defined as follows:

$$(1-Y)\,\tfrac{1}{2}\,D_w^2 + Y\,\tfrac{1}{2}\{max(0, m - D_w)\}^2$$

Where D_w is the euclidean distance between the outputs of the sister networks. The **Y** value represents the label for the class 1 if similar 0 if not. The m is a marginal value greater than 1 which indicates the dissimilarity between inputs.



.

The data source:

The data consists of 37 wav files of 2 channel, sampled at 44k Hz and each is around 11 minutes long. Each audio file contains two tracks, each with its own speaker. Each audio segment consists of a combination of four states, speaker "A" is talking, speaker "B" is talking, both speaker "A" and "B" are talking, or both speakers are silent.

Preprocessing:

Caching the Data: The entire dataset have been cached so that it can be reused without loading the entire dataset over again.

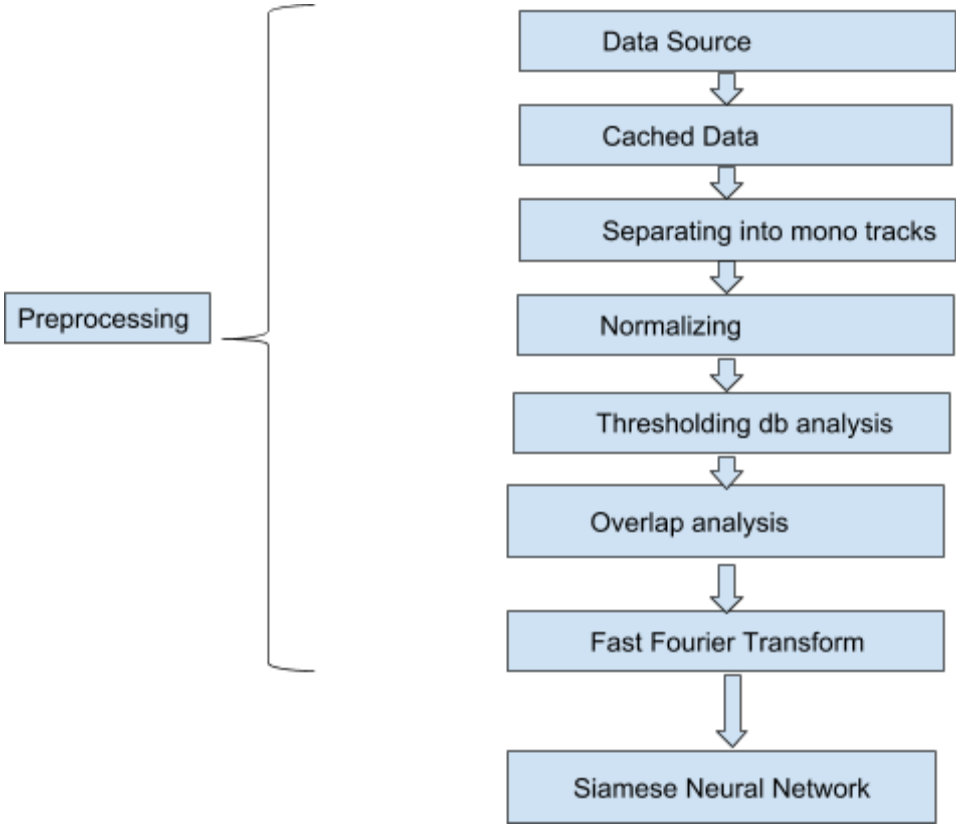
Splitting the Training Data: The audio signal first the wave files have been splitted to mono using `split_to_mono()`.

Normalization: The normalization is done by taking the maximum value of each channel and then divide the target sample by the maximum value and converted the result to db.Finally the amplitude of each sample of the mono channels have been changed for normalization by multiplying the resultant db value with the amplitude of the each sample.

Decibel Analysis: Decibel analysis is used determine speech vs non speech in each mono track using `detect_nonsilent()`. The minimum time for detecting silence is 500 ms and the amplitude is -30 db to flag breathing as non speech.

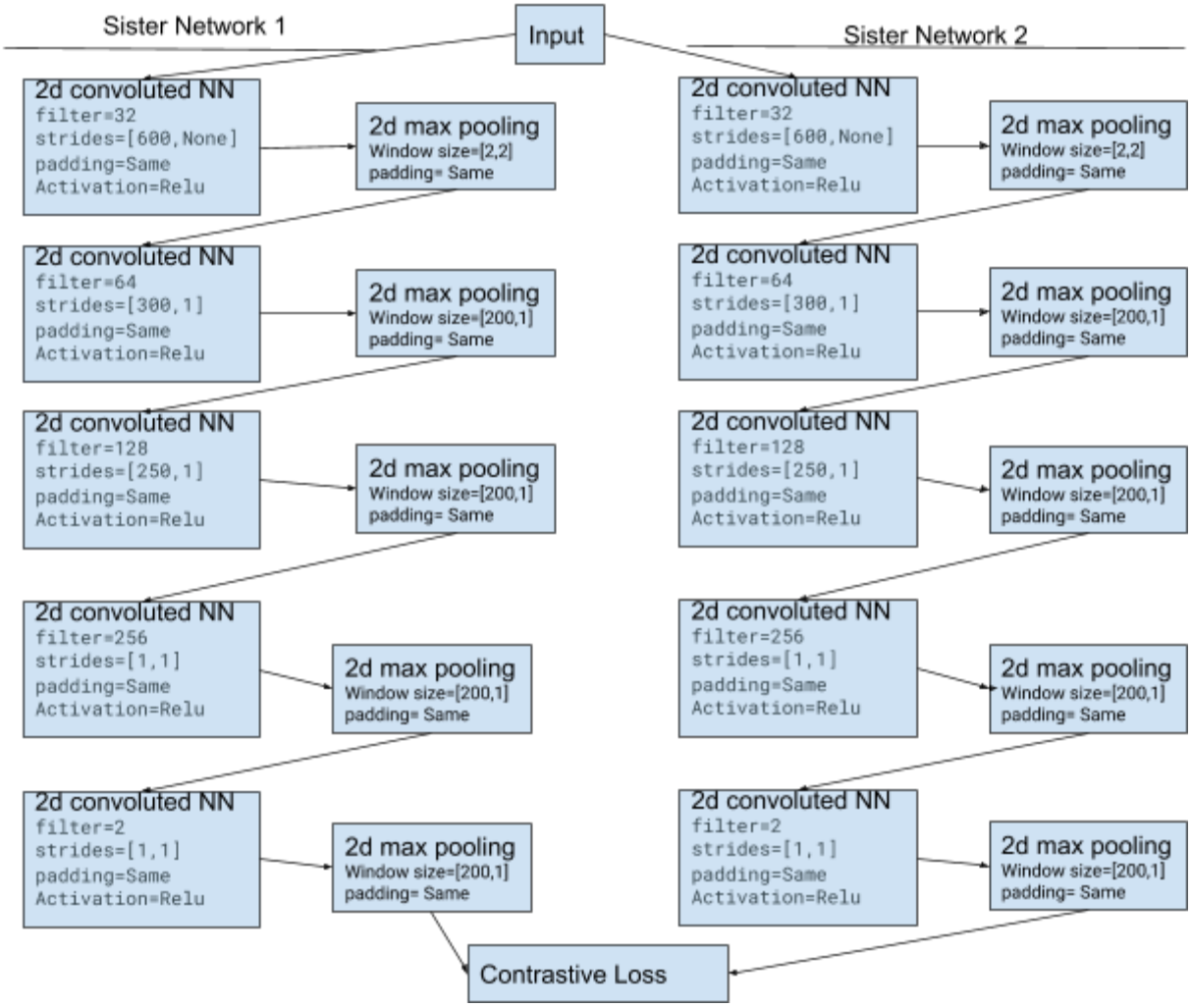
Overlap analysis: On the speech samples extracted from normalized mono tracks, overlap analysis was done to determine when each of the speakers is talking. If there is no overlap in time, then one of the speakers is talking. Segments where one speaker is clearly talking and the other is not is marked as “clean” audio and used to train the network. If there is overlap in speaking times, then both of the speakers may have been talking at that time. These segments are marked as “dirty” audio, and the purpose of the neural network is to correctly identify which speakers are speaking during these segments.

Spectral Analysis: Spectral analysis gives a better representation for the siamese network to extract features from. Here the normalized data have been transformed to spectral domain and fed it to the neural network.



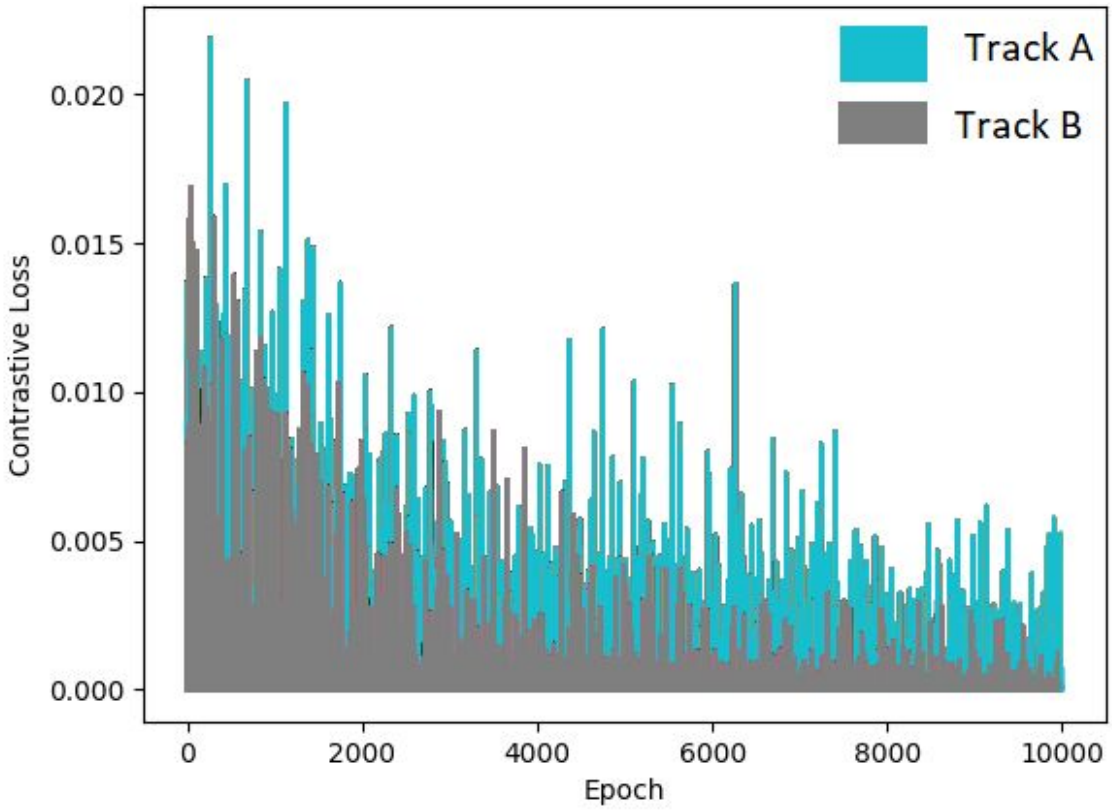
Training:

Siamese Network: The siamese network we use consists of two sister networks with identical trained weights. The purpose of this is to create a network capable of extracting uniform, meaningful representations from inputs, and comparing those representations for similarity. Each of the sister networks has five 2d convolutional layers with 2d max pooling. The inputs for the training were fed in the two sister networks and the outputs of the networks were fed to contrastive loss function. We used Tensorflow's minimizer to minimize the contrastive loss function.



Batch: For batch training, the spectral graphs of all segments of clean data are concatenated into a single continuous audio stream. This audio stream is then broken up into 500 ms chunks. Each of these chunks is an 8x32 Numpy array (8 time frames, 32 frequency responses). These chunks are used as the training input to the neural network.

Training: The contrastive loss function gives the similarities between the inputs, where 0 means they are identical, and larger numbers indicate dissimilarity. At training time, the siamese network was fed chunks of “clean” spectral data, and the contrastive loss between these chunks was minimized using TensorFlow. This allowed the network to learn the similarities between different segments of speech for a single speaker’s voice. From the below graph, it can be seen that with time the network learned the similarities for both of the tracks as the contrastive loss fell. The graphs very shaky since the network is trained using many different pairs of audio chunks. We ran 10,000 training batches using 5 pairs of audio chunks per batch.



Testing:

Preparing the Test Data: The test data is prepared in a similar format to the training data, except the segments are not concatenated into a single continuous track. Instead, chunks that are too short are padded using the average across the chunk at each frequency. This ensures that the padding on the data has minimal effect on its representation.

Testing Method: For the testing, immediately after we ran all training iterations, we ran once through all the test data. We took the speaker segment and assigned that to the primary network (or sister network 1). We took the unknown sample and assigned it to the test network (or sister network 2). We then ran them through a contrastive loss function and then a threshold analysis of the results. What we found by examining the data, and the raw results was that a difference less than 0.015 indicated that it was similar to the speaker, while a difference greater

indicated that it was not similar. We then collected the timestamped results of all this data, combined continuous segments of similar data, and wrote the identified audio segments out to a results file for later analysis.

Performance Analysis:

Preparation For Analysis: To prepare the analysis data, we read in all the raw test results form the test phase, then parsed the times to be in the same format as the CSV_Files_Final timestamps. We then output these results to a final csv file for the full analysis.

Analysis Methodology: This was one of the most difficult parts of this assignment. We tried several methods for automated analysis of our networks performance, including raw time comparisons and comparisons with a margin of error. However none of these methods worked because we found is that our network seemed to be splitting some segments that the provided data did not. With further analysis it appears that this was caused by the human not separating the audio segments because there was only a 0.5 second gap. Because of this we were forced to do a visual comparison between the result data and the expected data. Through this analysis we allowed up to a 0.5 second difference between the start and end times, and we allowed for a single segment in the expected data to span a few segments in our result data.

Analysis Results: Due to the time consuming nature of doing manual analysis of our result data, we only had time to do a full analysis of the results for the first audio file (HS_D01.wav). The results were both promising, and informative. For audio track 0 (classified as Track A) we had 13 misclassifications where our network either had a greater than 0.5 time difference with the start or end, or classified positive a result not found in the expected data. We then had 42 misses with the expected data where the network did not have any reported segment close to the expected segment. The expected data had a total of 211 expected tracks giving us a 74% success rate, I will discuss this in the pitfalls below. However, for track 1 we did not have as good results. We had a 100% miss rate, I will discuss this in the pitfalls.

Network Pitfalls

Our results helped us to identify some pitfalls that we would have liked to fix if we had more time. The first pitfall was that our decibel analysis was static. In our initial code phase, the first 30 seconds of all the audio files and attempted to find a static threshold that would work for all files. From our findings we found that track 0 from the first .wav file hand the lowest decibel threshold and so it was the threshold we used. The issue with this is that as the speaker is closer or further away from the microphone that threshold changes. In the analysis from track 0 from the first .wav file we had 42 misses, we have determined that the cause of this is these segments dipped below our threshold and were not picked up for analysis. We only had 13 misclassifications by the neural network so if we had analyzed them it is feasible to say that we

may have had as high as a 94% success rate. With track 1 on the first audio file we had a 100% miss rate. When I looked at the raw audio for this track we found it to be extremely breathy. We hypothesize that with our train method caused this neural network to accidentally train for breath instead of speaking. Because of this if we had implemented a filter for breath as well as an adaptive threshold for speaking analysis we could have seen similar results as track 0.

Appendix I: Results Table

Appendix II : Code