Speaker Identification in a Noisy Environment
Raw Waveforms Vs MFCC

Name (SUNet IDs): 1) David Grogan (dgrogan), 2) Phathaphol [Peter] Karnchanapimonkul (phatk)

Abstract

Our work was motivated by helping early- or mid-stage Alzheimer's patients identify who is speaking to them over the phone when multiple people are on the other end. Alzheimer's patients forgetting their loved ones is hurtful for both the patient and their family. We apply a deep neural network to identify speakers in noisy environments, which can be used in a phone app to recognize who's currently speaking and show their picture and brief description to jog the patient's memory. Previous related work [1][2][3] only uses audio in clean environments. We found that preprocessing audio into MFCCs, which were used in most prior work[1][2], are insufficient in the context of noisy audio. Our final CNN model, which used raw audio waveform as input, achieved 86% accuracy on testing data sets not previously seen.

1 Introduction

Alzheimer's disease deteriorates sufferers' memory and ability to function. There is no cure. Memory problems are the first overt symptom. For a time the sufferer is embarrassed of their forgetfulness, choosing to withdraw from the family instead of allowing their family members precious last moments with the sufferer. This project aims to alleviate a small part of this problem, namely when a group of family members calls an Alzheimer's sufferer, the sufferer has a hard time matching the voice to the speaker's identity. We built a neural network that will perform this mapping based on the speaker's voice.

We note that our project can be employed for business conference calls with no or trivial modification.

The input to our model is half-second of a voice recording. The output is a probability distribution over a known set of 20 speakers. We used 20 because most families have at most 20 members who regularly speak to a relative with Alzheimer's. Similarly, we haven't seen a team that uses audio conference calling which has more than 20 regular speakers.

2 Related work

The Future Work section from Alfredo's project and Manish's project [1][2] suggest to implement a speaker recognition model on noisy data.

An approach by McLaren et al [4] explores a novel method to leverage MFCC to transform our model. This approach introduces two new data-driven methods of utilizing DCT coefficients for speaker recognition. Nonetheless, this alternative is very complex and we decide to just explore the MFCC coefficients.

An interesting approach is the one by Sainath et al [5]. He proposes that it is possible to modify waveform-based model, which would be similar to how we try to leverage MFCC in our fully connected model, such that the accuracy of this model is equivalent to Convolutional, Long Short-Term Memory Deep Neural Network (CLDNN). Although his approach is worth imitating, we will try to create a CNN to try to achieve higher accuracy.

An approach to create a CNN proposed by Torfi et al [6] explores a new way to train a CNN. That is, the approach suggests the usage of “3D-CNNs for direct speaker model creation in which … an identical number of spoken utterances per speaker is fed to the network for representing the speakers’ utterances and creation of the speaker model.” Nonetheless, the complexity and time requirement would increases tremendously to try to implement it and we choose not to do it.

3 Dataset and Features

We're using audiobooks from LibriVox [7], whose tagline is "Free public domain audiobooks. Read by volunteers from around the world." We've seen datasets that are derivatives of LibriVox (e.g. LibriSpeech dataset out of Johns Hopkins University [8]) but they are organized for Speech Recognition, not Speaker Identification.

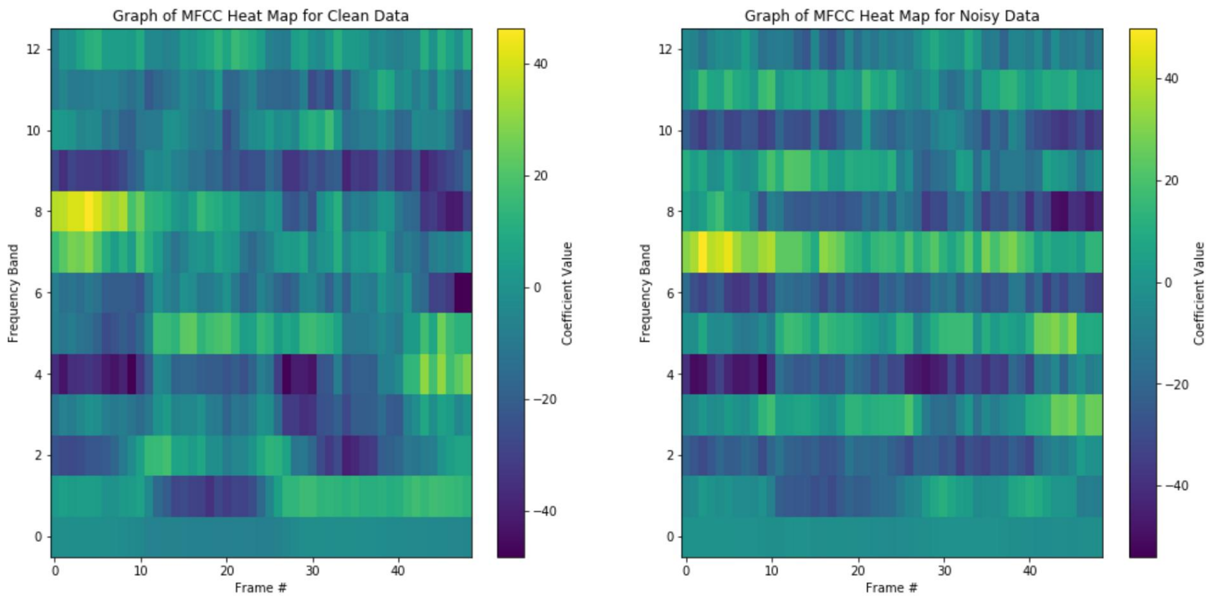
The audiobooks are mono-channel, 22050Hz mp3s, commonly with one mp3 per chapter. We downloaded >500 books after determining which had only one speaker per book and being careful not to have any books with duplicate speakers [9]. We then chose the top 20 books in terms of length whose mp3s had non-corrupt headers. We removed corrupt metadata in a few instances. We ended up with 20 books of length at least 18.5 hours. We trimmed the longer books to 18.5 hours so that we'd have **balanced classes**.

After listening to some samples, we found that half-second is the minimum time a human can reliably identify the speaker so we decided to feed our network half-second data clips. That led us to have **2,664,000 training samples**. We conservatively set aside 5% of these as a **test set**, knowing that if we needed more training/validation data, we could pull some of the test data if we determined that we wouldn't need so many samples to get a reliable performance estimate. When we trained the model, we usually passed in 10% of the data as a **validation set,** again knowing we could revisit later, but we never needed to.

3.1 Preprocessing

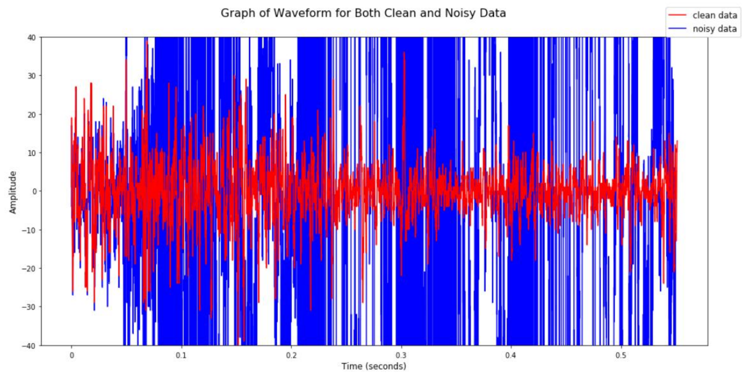
We spent a long time determining the type of predictors to feed our model. We thought feeding wave samples directly to the neural network would be too computationally expensive. An alternative is to run a Fourier transform on the waveform and use the resulting spectrogram. (One project [9] used spectrograms but their results were not good. This paper [6] also stops at the spectrogram stage because of their goal to get closer to an end-to-end learning system. They also specifically eschew noisy environments.) We could further analyze the spectrogram at a few different stages, but generating MFCCs (Mel Frequency Cepstral Coefficients—a transformation of a spectrogram informed by the human ear's sensitivities) is most common so that's what we used. We use the standard number of MFCC cepstrum, 13 [10], and a common Discrete Fourier Transform (DFT) window of 25ms with a 10ms stride [11]. Further, we've read the first and second derivatives of the MFCC coefficients are informative, so we added those derivatives to our MFCC dataset.

Figure 1: MFCC of Clean and Noisy Data



3.2 Data Augmentation

Initially, we added noise samples from soundjay to the audio books using SoX, a command-line interface written in C. We explored using SoX commands to insert noise to the file but it was too slow so we used the python library pydub. We also first overlaid one noise file at a time but did not do volume normalizing for the noises even though some of our sounds were way louder than others; we got bad performance. In the end, we overlaid 20 seconds at a time and did volume normalization.



4 Methods

We trained two model architectures. The first is a fully connected network of 4 dense layers of 100-200 neurons each that used a flattened 2-dimensional MFCC matrix of size (49, 39) as input. The second architecture we used is a CNN that used a 1-dimensional vector of 11025 audio samples as input. Because our task is a multi-class single-label model, we used the standard categorical cross-entropy loss function for both:

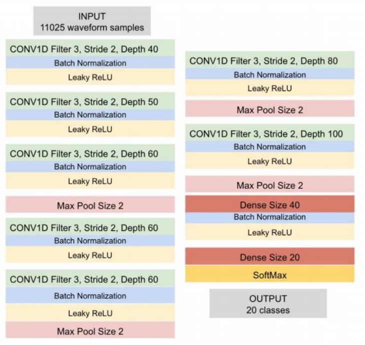
$$-\sum_{i=1}^{m} y_i \log \hat{y}_i$$

We switched architectures and inputs because we weren't getting good results on even the training data with the fully connected network when working with the augmented noisy data. Expanding that network until it learned the training data well would have led to a steep increase in the number of parameters, eventually requiring ever more training data to not overfit, leading to very long training times.

Instead, we switched to a CNN for parameter reuse. The 1D convolutions in the CNN takes advantage of the structure of a waveform: the relationship between *k* consecutive samples at time 0 is related to the relationship between *k* consecutive samples at time *t*. A CNN allows for that similar relationship to be exploited, while a fully connected network ignores it.

Our final model is a 13-layer CNN, including seven 1D convolution layers, 4 max-pooling layers, and 2 Dense layers before a softmax output.

Figure 3: Final Architecture of CNN.



5 Experiments/Results/Discussion
We experimented with many hyperparameters in our CNN. We started with a standard Adam optimizer with learning rate 0.001. Our model required relatively few epochs for the validation set to plateau (see below figures), indicating a possibly overaggressive learning rate. But when we added a very small learning rate decay the model slowed to a crawl, so we reverted to a constant 0.001. This is the first thing we'd revisit if we had more time.

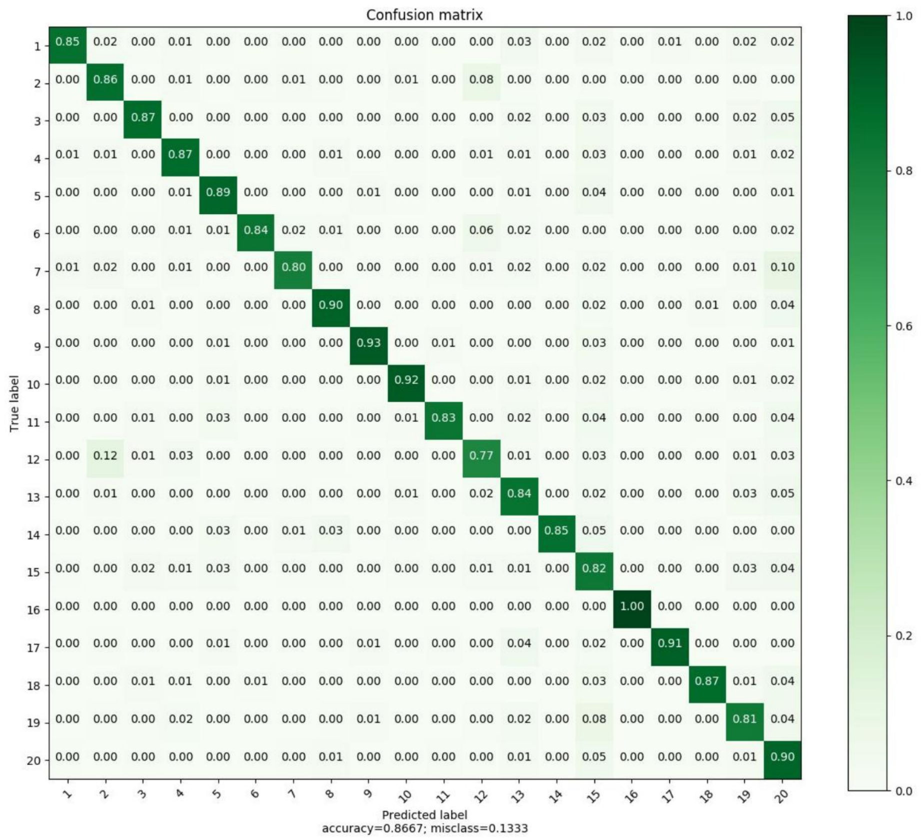
Our final mini-batch size was 64. We found that was a good tradeoff between training speed and performance. When we tried smaller mini-batches, training speed increased and the performance gain wasn't great. Increasing mini-batch size had little effect on training speed, so settled on 64.

For our CNN layers, we chose a stride of 2 so that the width of the data cube would decrease and the Dense layer would have a manageable amount of hidden neurons. We could have used more max-pooling to achieve the same effect, but we read that Geoffrey Hinton thinks max-pooling layers are a mistake [12], so we chose to use more convolution layers.

We also used a filter size of 3 and many channels so that many low level features could be learned. With our dataset, noise is constantly interrupting the signal, so our model would be well-served to learn on a narrow window of samples so that some windows would capture only signal and no noise. But the tradeoff is that many channels are needed because a lot of them will see an adulterated signal or even only noise.

Our primary metric is accuracy. Had our confusion matrix been grossly non-uniform, we may have expanded to include a floor on the lowest-performing speaker, but as the confusion matrix in the appendix shows the performance across speakers was not uniform, but not overly disproportionate.

Figure 4: Confusion Matrix of CNN model



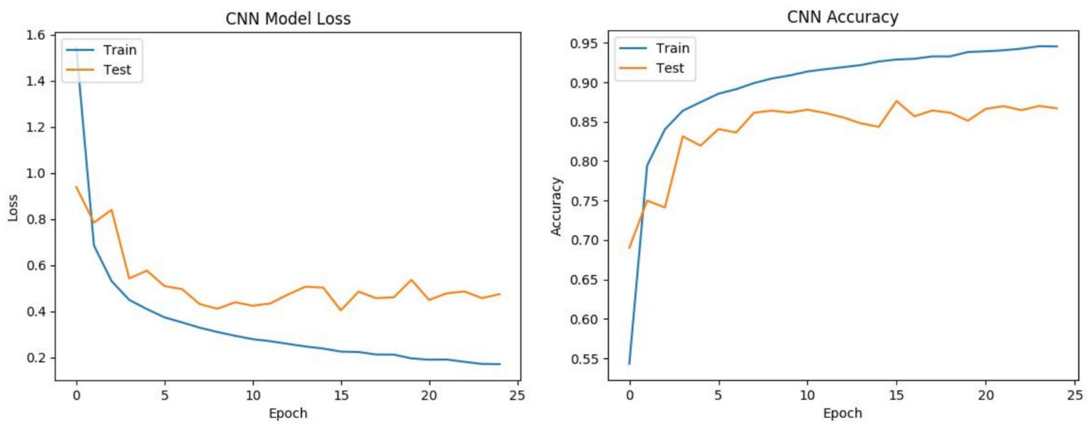
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	0.85	0.02	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.03	0.00	0.02	0.00	0.01	0.00	0.02	0.02
2	0.00	0.86	0.00	0.01	0.00	0.00	0.01	0.00	0.00	0.01	0.00	0.08	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
3	0.00	0.00	0.87	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.00	0.03	0.00	0.00	0.00	0.02	0.05
4	0.01	0.01	0.00	0.87	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.01	0.01	0.00	0.03	0.00	0.00	0.00	0.01	0.02
5	0.00	0.00	0.00	0.01	0.89	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.01	0.00	0.04	0.00	0.00	0.00	0.00	0.01
6	0.00	0.00	0.00	0.01	0.01	0.84	0.02	0.01	0.00	0.00	0.00	0.06	0.02	0.00	0.00	0.00	0.00	0.00	0.00	0.02
7	0.01	0.02	0.00	0.01	0.00	0.00	0.80	0.00	0.00	0.00	0.00	0.01	0.02	0.00	0.02	0.00	0.00	0.00	0.01	0.10
8	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.90	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.00	0.00	0.01	0.00	0.04
9	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.93	0.00	0.01	0.00	0.00	0.00	0.03	0.00	0.00	0.00	0.00	0.01
10	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.92	0.00	0.00	0.01	0.00	0.02	0.00	0.00	0.00	0.01	0.02
11	0.00	0.00	0.01	0.00	0.03	0.00	0.00	0.00	0.00	0.01	0.83	0.00	0.02	0.00	0.04	0.00	0.00	0.00	0.00	0.04
12	0.00	0.12	0.01	0.03	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.77	0.01	0.00	0.03	0.00	0.00	0.00	0.01	0.03
13	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.02	0.84	0.00	0.02	0.00	0.00	0.00	0.03	0.05
14	0.00	0.00	0.00	0.00	0.03	0.00	0.01	0.03	0.00	0.00	0.00	0.00	0.00	0.85	0.05	0.00	0.00	0.00	0.00	0.00
15	0.00	0.00	0.02	0.01	0.03	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.00	0.82	0.00	0.00	0.00	0.03	0.04
16	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00
17	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.04	0.00	0.02	0.00	0.91	0.00	0.00	0.00
18	0.00	0.00	0.01	0.01	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.03	0.00	0.00	0.87	0.01	0.04
19	0.00	0.00	0.00	0.02	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.02	0.00	0.08	0.00	0.00	0.00	0.81	0.04
20	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.01	0.00	0.05	0.00	0.00	0.00	0.01	0.90

Table 1: Comparison of Accuracy of Fully Connected Network Vs CNN on Different Set-ups

Results for Fully connected network	**Results for CNN with Raw Wave Input**
Clean Training Data → Clean Predictions 88%	Clean Training Data → Clean Predictions 94%
	Clean Training Data → Noisy Predictions 14%
Noisy Training Data → Noisy Predictions 63%	Noisy Training Data → Noisy Predictions 86%

We think our model overfit to the training set somewhat, because as you can see in the plots below, there's a persistent gap between training and validation set performance. We didn't use regularization because of our virtually limitless supply of training data — we only used 15% of our training data to train any one model. When we trained a model that got good training set performance and decent validation set performance, we had the liberty of applying more data.

Figure 5: CNN loss and accuracy over 25 epochs



6 Conclusion/Future Work

From this project, it is clear that neural network are powerful tools for the purpose of speaker identification even under noisy environments. Noisy environments have been implemented by inserting noises that are ubiquitous such as the sound from keyboard being pressed, crowd talking, and plastic crumbling. Initially, our goal is to create a fully connected network to train and predict the result from the training set. A convolutional neural network is implemented later to see if the accuracy improves and the result section confirms this sentiment.

The algorithm that performed the best is our final model with 13-layer CNN, including seven 1D convolution layers, 4 max-pooling layers, and 2 Dense layers before a softmax output. This CNN algorithm performs better than the fully connected network that leverages MFCC, because CNN takes advantage of repetitive structure in the waveform, while the latter does not.

For future work, if we have more time and more computational resources, we would downsample the audio from 22050 to 16000 or 8000 to allow for more channels in each layer with the same amount of training data and training time. In addition, we would also perform more aggressive hyperparameter search, including experimenting with cyclical learning rate and bayesian optimization.

References [ranked in order of appearance of research paper]

1. Méndez, Alfredo. “Speaker Identification with Deep Neural Networks.” Stanford CS 230 Past Projects – Winter 2019.
2. Gupta, Rish, Manish Pandit, Sophia Zhen. “Speaker Identification: Text Independent Context.” Stanford CS 230 Past Projects - Spring 2018.
3. Lukic, Yanick, et al. "Speaker identification and clustering using convolutional neural networks." 2016 IEEE 26th international workshop on machine learning for signal processing (MLSP). IEEE, 2016.
4. Sainath, Tara N., et al. "Learning the speech front-end with raw waveform CLDNNs." Sixteenth Annual Conference of the International Speech Communication Association. 2015.
5. McLaren, Mitchell, and Yun Lei. "Improved speaker recognition using DCT coefficients as features." 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, 2015.
6. Torfi, Amirsina, Jeremy Dawson, and Nasser M. Nasrabadi. "Text-independent speaker verification using 3d convolutional neural networks." 2018 IEEE International Conference on Multimedia and Expo (ICME). IEEE, 2018.
7. “LibriVox.” Free Public Domain Audiobooks, librivox.org/.
8. Povey, Daniel. “LibriSpeech ASR corpus”. Open Speech and Language Resources, 2015, https://www.openslr.org/12/.
9. Gill, Christopher. “Automatic Speaker Recognition Using Transfer Learning.” Dec 12, 2017, https://towardsdatascience.com/automatic-speaker-recognition-using-transfer-learning-6fab63e34e74
10. Poorjam, Amir Hossein. “Why We Take Only 12-13 MFCC Coefficients in Feature Extraction?” ResearchGate, 5 May 2018, https://www.researchgate.net/post/Why_we_take_only_12-13_MFCC_coefficients_in_feature_extraction.
11. Haytham Fayek. “Speech Processing for Machine Learning: Filter Banks, Mel-Frequency Cepstral Coefficients (MFCCs) and What's In-Between.” Haytham Fayek, 21 Apr. 2016, https://haythamfayek.com/2016/04/21/speech-processing-for-machine-learning.html
12. “Geoffrey Hinton on max pooling (reddit AMA)”. 11 Nov. 2014, https://mirror2image.wordpress.com/2014/11/11/geoffrey-hinton-on-max-pooling-reddit-ama/

Contribution

David: Github setup. Audio Format and Preprocessing. Applying MFCC. CNN Architecture and coding. Poster and Write Up.
Peter: Analysis of MFCC. Audio Research. NN Architecture and coding. Poster and Write Up.

Github: https://github.com/davidsgrogan/cs230