2022-12

INTERPRETABILITY FOR ARTIFICIAL INTELLIGENCE IN SPEAKER RECOGNITION TASKS

Gooch, Elizabeth F.

Monterey, CA; Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

INTERPRETABILITY FOR ARTIFICIAL INTELLIGENCE IN SPEAKER RECOGNITION TASKS

by

Elizabeth F. Gooch

December 2022

Thesis Advisor: Joshua A. Kroll
Co-Advisor: Robert L. Bassett

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** December 2022	**3. REPORT TYPE AND DATES COVERED** Master's thesis

4. TITLE AND SUBTITLE INTERPRETABILITY FOR ARTIFICIAL INTELLIGENCE IN SPEAKER RECOGNITION TASKS	**5. FUNDING NUMBERS**
6. AUTHOR(S) Elizabeth F. Gooch	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release. Distribution is unlimited.	**12b. DISTRIBUTION CODE** A

13. ABSTRACT (maximum 200 words)

In the future, the DOD will more frequently incorporate AI into tasks with high consequences and, in turn, be scrutinized for mistakes. So far, much AI development has occurred in the private sector for which the consequences of error are lower than in defense. The DOD faces different incentives for the interpretability of AI and needs AI to aid decision-makers instead of replacing them. My thesis project implements techniques to improve trust in a speaker recognition task by focusing on meaningful and theoretically sound feature extraction and model simplicity. My main result is expected and elicits action from DOD. I find that a convolutional neural network (CNN) model performs substantially better than a multilayer perceptron and that logistic regression cannot discern speaker identity. Thus, the DOD needs to focus on research to develop interpretable models for complex tasks. The other result I find is surprising and motivates interpretability: When I construct features using mel frequency cepstral coefficients (MFCCs) on a human speech signal with an improperly long window, my CNN achieves an accuracy of 92%. Ill-defined MFCCs are theoretically meaningless; however, I find that they help predict speaker identity. Further confounding this result, I find that when I construct the MFCCs using the suggested 30 ms window, the model's accuracy falls to 72%. Future research should explore disentangled CNN-based models and the concept of an MFCC as windowing time grows.

14. SUBJECT TERMS audio analysis, cepstral domain, interpretable AI, automated speaker recognition, feature extraction, model accuracy, mel frequency cepstral coefficients, convolutional neural networks, CNN, multilayer perceptron, MLP, logistic regression, PyTorch	**15. NUMBER OF PAGES** 95
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
Unclassified	Unclassified	Unclassified	UU

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

INTERPRETABILITY FOR ARTIFICIAL INTELLIGENCE IN SPEAKER RECOGNITION TASKS

Elizabeth F. Gooch
Civilian, Department of the Navy
BA, Agnes Scott College, 2005
PhD, Georgia State University, 2014

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
December 2022

Approved by: Joshua A. Kroll
Advisor

Robert L. Bassett
Co-Advisor

Gurminder Singh
Chair, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

In the future, the DOD will more frequently incorporate AI into tasks with high consequences and, in turn, be scrutinized for mistakes. So far, much AI development has occurred in the private sector for which the consequences of error are lower than in defense. The DOD faces different incentives for the interpretability of AI and needs AI to aid decision-makers instead of replacing them. My thesis project implements techniques to improve trust in a speaker recognition task by focusing on meaningful and theoretically sound feature extraction and model simplicity. My main result is expected and elicits action from DOD. I find that a convolutional neural network (CNN) model performs substantially better than a multilayer perceptron and that logistic regression cannot discern speaker identity. Thus, the DOD needs to focus on research to develop interpretable models for complex tasks. The other result I find is surprising and motivates interpretability: When I construct features using mel frequency cepstral coefficients (MFCCs) on a human speech signal with an improperly long window, my CNN achieves an accuracy of 92%. Ill-defined MFCCs are theoretically meaningless; however, I find that they help predict speaker identity. Further confounding this result, I find that when I construct the MFCCs using the suggested 30 ms window, the model's accuracy falls to 72%. Future research should explore disentangled CNN-based models and the concept of an MFCC as windowing time grows.

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

List of Figures

List of Tables

THIS PAGE INTENTIONALLY LEFT BLANK

List of Acronyms and Abbreviations

AI	Artificial Intelligence
CNN	Convolutional Neural Network
DARPA	Defense Advanced Research Projects Agency
DOD	Department of Defense
FFT	Fast Fourier transform
LFCC	Linear Frequency Cepstral Coefficients
MFCC	Mel Frequency Cepstral Coefficients
MLP	Multi-Layer Perceptron
NPS	Naval Postgraduate School
SGD	Stochastic Gradient Descent
XAI	Explainable Artificial Intelligence

THIS PAGE INTENTIONALLY LEFT BLANK

Executive Summary

Interpretability should be a primary focus of AI research for Department of Defense (DOD) tasks because DOD tasks are high-stakes, and mistakes are scrutinized. In this thesis, I implement two simple techniques to increase interpretability in artificial intelligence (AI): meaningful and sparse features and whether or not a simpler model can achieve acceptable performance. The AI application I focus on is speaker recognition. To begin, I review the state-of-the-art literature on interpretability. I also review the 70-year-long literature on mel frequency cepstral coefficients (MFCC), the premier feature extraction method for human speech. Next, using PyTorch's Dataset class and Dataloader, along with several custom functions, I organize the massive VoxCeleb 2.0 dataset for speaker recognition. VoxCeleb 2.0 contains over a million utterances by celebrity speakers taken from YouTube. On average, each speaker has 150 utterances in VoxCeleb 2.0. My custom dataset creates random pairs of utterances from randomly selected speakers and creates a balanced dataset of 50,000 to 100,000 "same" speaker and "different" speaker pairs. Using a convolutional neural network (CNN), an multilayer perceptron (MLP), and logistic regression, I model speaker recognition using PyTorch's neural network class. My results show that the performance of the CNN model using 12 well-defined MFCCs performs on par with a CNN model using 40 well-defined MFCCs. This result implies that the early literature on MFCCs is correct, and the first 12 MFCCs are the most important. While sparse models are not automatically more interpretable, fewer meaningful features allow users and practitioners to understand the inputs. My results also show that with 12 well-defined MFCCs, accuracy on the test set for the CNN is 72%, the MLP is 61%, and the logistic regression is 50%. This result implies that a complex model like a CNN is necessary for sufficient prediction in a speaker recognition task. My final result is that when I use an ill-defined MFCC, one that is extracted using too large of a window to capture a quasi-stationary signal of human speech, my CNN achieves 92% accuracy. This result implies that the pursuit of accuracy without a theoretical basis can lead researchers away from interpretability.

THIS PAGE INTENTIONALLY LEFT BLANK

Acknowledgments

Thank you, Dr. Joshua Kroll and Dr. Robert Bassett, for assisting me in mastering the concepts and mechanics of machine learning and artificial intelligence. Thank you, Dr. Natalie Webb, Executive Director of Defense Resources Management Institute, for giving me the opportunity to pursue this degree in Computer Science. Thank you to Mrs. Lisa Fierman for supporting me the entire way through.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 1:
Introduction

Private sector interests have created an environment of awesome advancements in machine learning and AI. However, the consequences of most automated decisions in the private sector are trivial as compared with those made by governmental entities, particularly for defense. In defense, for example, drone targeting of tanks for destruction requires precision. If a house was incorrectly identified as a tank by AI and destroyed, an investigation would follow. In contrast, predicting clicks on advertisement links is both inconsequential to people's lives; if a prediction is incorrect, the experts may simply try again. In this research, I will address considerations for increasing trust in AI such as constructing meaningful and sparse feature extraction and comparing simple models with complex ones; I apply these techniques to a task in speaker recognition.

Audio data is central to our lives. In 2020, audio data accounts for over 11 percent of the average American adult's daily media time [1]. However, most importantly, audio data analyses like speech recognition and voice verification have real-world applications. In 2021, the Department of Homeland Security Science and Technology Directorate partnered with the Johns Hopkins University Applied Physics Laboratory and the company Think-A-Move Ltd. to develop hands-free automatic speech recognition technology to make first responders safer. The technology will share data and improve communication capabilities in high-noise environments [2]. Automatic recognition of voice commands can mean life or death for first responders.

In this research, I take the following steps: first, I closely review two kinds of literature, one on interpretability in AI and one on feature extraction in human speech. Next, I create an appropriate dataset for the speaker recognition task, which is the practical focus of my research. Finally, I model a speaker recognition task defined in the literature as the binary classification of two audio samples as to whether they recorded the voice of the same speaker. I employ different feature extraction specifications of the mel frequency cepstral coefficients (MFCC) and variations of convolutional neural networks (CNN), a multilayer perceptron (MLP), and logistic regression to predict.

Drawing on my review of the literature on interpretability in AI, I apply increasingly simple models to the speaker recognition task such that CNN is the most complex and opaque model and logistic regression is the simplest and most interpretable model. My choices in feature extraction using MFCCs also align with the literature on interpretability in that I aim to utilize fewer and more meaningful features in the speaker recognition task.

In Figure 1.1, I illustrate the broad concept underlying my thesis research. Initially, I compare the performance of an ill-defined MFCC versus a well-defined MFCC. The ill-defined MFCC uses many the of default settings for MFCC extraction from PyTorch's torchaudio package, while the well-defined MFCC implements specifications identified in the literature. Then, using well-defined MFCCs, I reduce the complexity of the models from CNN to MLP to logistic regression and create a sparser dataset.

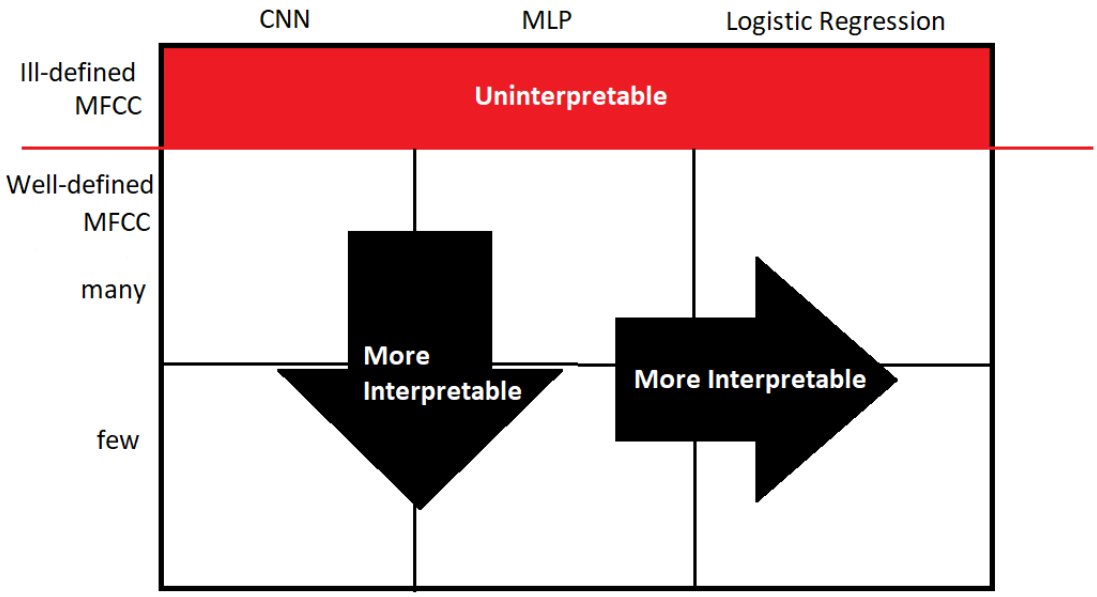

Figure 1.1. Thesis research design to explore interpretable AI

My results show three important findings about interpretability and accuracy in speaker recognition. First, in my CNN model, I find that using the ill-defined MFCCs enables much better performance than the well-defined MFCCs. In this task, ill-defined MFCCs use a window size 3-4 times larger than that required to extract a quasi-stationary human speech signal, which means that the MFCCs are not really MFCCs but something undefined in the literature. For perspective, the CNN using the 40 ill-defined MFCCs achieves an accuracy of 92%, while the CNN using 40 well-defined MFCCs achieves an accuracy of 72%.

The other two findings from my research are that once the MFCCs are well-defined, reducing the number of MFCCs from 40 down to 12 does not reduce the accuracy achieved by the CNN model. The literature on MFCCs for analyzing human speech from the early 2000s often suggests using 12 MFCCs, because each subsequent coefficient added to the group adds little new information about the speaker's vocal system. I also found that modeling with logistic regression is unable to learn to correctly predict whether a two samples are or are not from the same speaker.

The results of my research illuminate two common problems in AI. First, the high accuracy of the ill-defined MFCCs in my results shows that in the pursuit of high accuracy, modeling tasks have the potential to stray too far from the domain knowledge. Therefore, it is important to draw the parameters for feature extraction directly from expert knowledge. Second, some modeling tasks like speaker recognition need to use complex "black box" models like CNNs. Since the DOD will be increasingly reliant on complex automated tasks, investments in developing "interpretable" models like *disentangled* CNNs should be a priority.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 2:
Opportunities and Trade-offs for Trust in Artificial Intelligence

In this section, I discuss interpretable AI in terms of state-of-the-art best practices for maximizing trust and cite experts regarding what “trust" in AI means. Additionally, I compare and contrast interpretability with explainability in AI.

2.1 Understanding Interpretable AI through a Critique of Seminal Speaker Recognition Research

Rudin [3] explains that interpretability is crucial for trust in artificial intelligence models and sets these guiding principles for interpretable machine learning:

1. An interpretable machine learning model obeys a domain-specific set of constraints to allow it (or its predictions, or the data) to be more easily understood by humans. These constraints can differ dramatically depending on the domain (p. 3).
2. Despite common rhetoric, interpretable models do not necessarily create or enable trust – they could also enable distrust. They simply allow users to decide whether to trust them. In other words, they permit a decision of trust, rather than trust itself (p. 5).
3. It is important not to assume that one needs to make a sacrifice in accuracy in order to gain interpretability. In fact, interpretability often begets accuracy, and not the reverse. Interpretability versus accuracy is, in general, a false dichotomy in machine learning (pp. 5-6).
4. As part of the full data science process, one should expect both the performance metric and interpretability metric to be iteratively refined (p. 8).
5. For high stakes decisions, interpretable models should be used if possible, rather than “explained" black box models [3, p. 8].

For this thesis, I consider data and model design choices that may improve interpretability in a speaker recognition task using VoxCeleb 2.0 data. In the following section, I review the design choices made in Nagrani [4], the principal research on VoxCeleb 2.0, and suggest alternatives that I can employ for increased interpretability. In section 2.3, I concisely explain the difference between interpretability and explainability.

In this section, I juxtapose recent quintessential guidelines for interpretability and trust [3] with the seminal research on VoxCeleb 2.0 data [4], the data that I use to train my models, to illuminate the difficulties of putting the guidelines laid out in Chapter 2 into practice and the critical choices that I need to make in designing an interpretable speaker recognition system.

Later on in this thesis, in section 4.1, I detail the VoxCeleb 2.0 data. But to provide a brief overview at this point, VoxCeleb 2.0 contains over a million utterances drawn from YouTube from thousands of celebrity speakers, with multiple utterances per speaker. The speaker recognition task is centered on the question when two utterances are compared: "Are these utterances from the same speaker?"

Research by Nagrani [4] on speaker recognition from VoxCeleb 2.0 is the seminal analysis of the dataset. Important choices in the design of their analysis include:[1]

- Goal
  - Our aim is to move from techniques that require traditional hand-crafted features, to a CNN architecture that can train end-to-end for the task of speaker recognition (p. 7)
  - ...to achieve state-of-the-art performance on the VoxCeleb1 speaker verification task, outperforming all other traditional methods and recent deep learning methods (p. 2)
- Model
  - Deep CNN-based neural speaker verification system, which they name VGGVox (p. 2).

[1]Speaker recognition and speaker verification and the classification of two utterances from the *same speaker* or *different speakers* are equivalent in [4]. In this thesis, I will use speaker recognition to refer to the classification of two utterances from the *same speaker* or *different speakers*.

  - VGGVox is trained to map voice spectrograms to a smaller embedding space. Specifically, “a deep neural network trunk architecture is used to extract frame-level features, and the features are aggregated to obtain utterance-level speaker embeddings (p. 7).
  - Measure the similarity between speakers using the cosine distance between vectors in the embedding space (p. 2).
- Data
  - The model is trained on short-term magnitude spectrograms extracted directly from raw audio segments, with no other pre-processing [4, p. 7].

While I critique Nagrani [4] concerning the discord between design choices and interpretability, I am not condemning the research. First, I examine the author’s goals. Regarding interpretability, “while interpretable AI is an enhancement of human decision making, black box AI is a replacement of it" [3, 5]. The creation of an end-to-end speaker recognition technique that removes hand-crafted features epitomizes the replacement of human decision-making and harms interpretability according to guideline #2 in Chapter 2. Likewise, Nagrani, et al.’s other goal of state-of-the-art performance beyond another other model does not embody guideline #4 listed in Chapter 2.

Nagrani [4] designed their speaker recognition technique to alleviate the burden of speaker recognition from humans and, likely, improve accuracy because, at a large scale, humans would likely make errors. Therefore, is Nagrani, et al.’s method appropriate for some domains? Yes. In low-consequence tasks, a fully automated speaker recognition technique is necessary if we, the audience, want the information. For example, if Nagrani, et al.’s technique can label billions of audio clips for perusal via YouTube, and without the advent of this technique, those clips would be unlabeled and difficult to search through, then it was helpful. Moreover, a labeling error during this task would not be harmful.

A key to interpretability, though, is figuring out when it is needed and what interpretability means in that domain. For speaker recognition, biometric voice access could be a subset of the domain where mistakes matter, either false positives or false negatives. Or, if a future iteration of AI like Amazon’s Alexa voice ID, where:

> You can train Alexa to better recognize you by creating a Voice ID. After you set up your ID, Alexa can call you by your name and deliver personalized results based on your voice. Alexa can even distinguish your voice from those of other people in the house. Adults, teenagers, and children can all create a voice profile [5, paragraph 2].

becomes an integrated part of defense systems, then speaker recognition techniques will need to be trusted.

Next, I will critique modeling choices from Nagrani [4]. The authors explain their choice of a CNN as follows:

> We use 2D CNNs as feature extractors and treat 2D spectrograms as single-channel images. It is perhaps unnatural to treat spectrograms in this manner where the same convolution is used at every point since, unlike in a visual image where an object may appear at any location, a pattern can appear at any point on the time axis but we would not expect patterns to also be frequency independent. However, deep networks can potentially learn frequency-specific filters if they are needed for solving a downstream task; for instance, some filters can only fire on specific patterns existing in the low frequency region, whilst fully connected layers can be position dependent. Therefore, even if a 2D CNN uses shared filters on the spectrogram, the model has the capability to divide the filters into low/high frequency groups [4, p. 7].

The CNN does a lot of work for feature extraction and embeddings, however, CNN is a “black box" which is “difficult to troubleshoot" and “often predict[s] the right answer for the wrong reason" [3, p. 2]. However, Rudin [3] also outline the draws of black box models for designers. Black box models are easier than interpretable models because of:

1. the computational complexity of the optimization problem (4).
2. the data are problematic and require troubleshooting (4).
3. an unclear definition of interpretability in the domain [3, p. 4].

A black box model violates guiding principle #1 from Chapter 2 because the CNN is not easily understood by humans. For this domain in which raw inputs are endemic, disentangled neural networks are suggested to increase interpretability [3].

The next step in [4]'s model is the creation of utterance-level speaker embeddings and minimizing the cosine distance between the vectors. The speaker embeddings are similar in concept to case-based reasoning, a technique to improve interpretability. Rudin [3, p. 22] states that "Case-based reasoning is a paradigm that involves solving a new problem using known solutions to similar past problems...[and] model that performs case-based reasoning is appealing, because by emulating how humans reason, the model can explain its decision-making process in an interpretable way" [3, p. 22]. Recent research is integrating case-based reasoning into deep learning through prototype and prototype-parts-based classification.

Finally, Nagrani [4] utilize one of the most hands-off depictions of audio data, the spectrogram. The purposeful omission of human data wrangling goes together with the authors' overall end-to-end design. However, a spectrogram of an audio file contains a lot of information. The authors are expecting their model to pick out the important data for the speaker recognition task.

Speaker recognition feature extraction has a long and large literature which I detail in Chapter 3. Nagrani [4] design choice of raw inputs that do not draw on any voice expertise violates guiding principle #5 since it is possible to make the model more interpretable through point feature extraction.

Nagrani [4] do state why they want to use the spectrogram:

> ...not only does the performance of MFCCs [mel frequency coefficients] degrade rapidly in real-world noise...but by focusing only on the overall spectral envelope of short frames, MFCCs may be lacking in speaker-discriminating features (such as pitch information) [4, p. 2].

However, per guiding principle #4, "as part of the full data science process, one should expect both the performance metric and interpretability metric to be iteratively refined" [3, p. 8], the reality of Nagrani et al.'s claim should be explored further.

2.2 Designing an Interpretable Machine Intelligence

Regarding the design of an interpretable speaker recognition technique, I will follow the guidance from [6], “to begin mapping the landscape of methods for accountability of artificial intelligence (AI) systems...mapping the categories of methods that one could help us to assess whether an AI system is meeting its objectives" [6, p. 47]. The authors “define accountability as being able to ascertain whether an AI system is behaving as promised, which is necessary for determining blame-worthiness" [6, p. 47] and point out explicitly that accountability is essential “as cases involving AI systems behaviors are adjudicated via litigation and prescribed via regulation and legislation" [6, p. 48].

The categories of approaches are:

1. Transparency in the process and software [6 p. 48].
2. Interpretable models [6 pp. 48-49].
3. Post hoc inspection of model outputs [6 pp. 49-50].
4. Empirical performance [6 p. 50].
5. Properties guaranteed by design [6, pp. 50-51].␣

To increase transparency, standards regarding data and model description and should be improved and unified throughout the literature. Additionally, open-access code and training environments should be released for inspection and additional testing [3], [6], [7].

Interpretable models are diverse and include employing regularizers to reduce nonzero parameters or using logic-based models like decision trees. Rudin [7] “encourage policy-makers not to accept black box models without significant attempts at interpretable (rather than explainable) models, that would be even better" [7, p. 15] and “since interpretability is domain-specific, a large toolbox of possible techniques can come in handy" [7, p. 9].

Post hoc inspection uses visualization, classic statistical methods, and algorithmic methods to identify properties of a black box or already developed AI system. Kim and Doshi-Velez [6] recognizes the pitfalls of post hoc inspection, and [3] more clearly defines the fault of this method. Rudin [3] states “Explanations for black boxes are often problematic and misleading, potentially creating misplaced trust in black box models" [3, p. 8] and goes on to discussing saliency maps. The authors explain:

> One particular type of posthoc explanation, called saliency maps (also called attention maps) have become particularly popular in radiology and other computer vision domains despite known problems...Saliency maps highlight the pixels of an image that are used for a prediction, but they do not explain how the pixels are used...Saliency maps also tend to be unreliable; researchers often report that different saliency methods provide different results, making it unclear which one (if any) actually represents the network's true attention [3, pp. 8-9].

Another particular post hoc technique that I would like to highlight for this thesis is SHAP and LIME. Rudin [3] points out that

> techniques such as SHAP and LIME are tools for explaining black box models, are not needed for inherently interpretable models [and even] [i]nterpretable supervised deep neural networks that use case-based reasoning...do not need SHAP values because they explicitly reveal what part of the observation they are paying attention to, and in addition, how that information is being used... [t]hus, if one creates an interpretable model, one does not need LIME or SHAP whatsoever [3, p. 9].

Next, Kim and Doshi-Velez [6] describe methods for accountability that do not require humans to understand the model at all, like empirical performance and properties guaranteed by design. A good example of empirical performance is:

> part of a pre-market safety process for a self-driving car may involve measuring certain kinds of safety violations (such as near misses) in a variety of settings over a series of test runs [6, p. 50].

Checks like the pre-market safety check described above can be executed by a regulatory agency or company. Properties guaranteed by design are hard to obtain in real-life situations. Credit scoring can provide a good example of this design. When using a monotone model, "increasing a feature (although others are held constant) will always increase or decrease

the output (which may be a valuable guarantee such as in a credit scoring system, where we may want to guarantee that increasing income will increase the score)" [8, p. 50].

2.3 Interpretability versus Explainability

According to Rudin [3], the concepts and research around explainability and interpretability arose apart from one another, yet have been merged recently in the literature in a confusing way. Explainability arose out of the function approximation literature (circa 2016), while interpretability arose from expert systems and decision trees (circa 1950). The authors want to make clear that interpretability is not explainability, nor is explainability helpful for trust. Moreover, Rudin [3] drive home that

> Explainability and interpretability techniques are not alternative choices for many real problems, as the recent surveys often imply; one of them (XAI) can be dangerous for high-stakes decisions to a degree that the other is not [3, p. 8].

Looking at the defense literature, however, the authors of [9] characterized the new machine-learning systems in the bottom panel of Figure 2.1 as able to (1) explain the rationale, (2) characterize strengths and weaknesses, and (3) convey an understanding of future behavior. Additionally, XAI should utilize a portfolio of methods with diverse design options and have an interface for human users to translate models and provide evidence of explanations. It is important to note that the explainable model in the bottom panel of the picture is not clearly defined but can elicit an answer to the user's questions posed in the top panel.

While the authors of [9] provide an encouraging overview of XAI, National Institute of Standards and Technology (NIST) researcher Broniatowski [10], defines explainability in more detail. Machine learning systems have explainability requirements that are distinct from what is known as interpretability. Drawing on the experimental psychology literature, Broniatowski [10] makes the following contrast:

- *Explanation* aids in improving algorithms because the system designer can accurately describe the implementation that led to the algorithm's output [10].
- *Interpretation* relates the system's purpose to the goals and preferences of the end user through a contextualized output [10].

Simply put, in computer science, explanation lends itself to debugging or improving algorithms [11]. Looking back at the DARPA motivational image in Figure 2.1, the users could be either a designer or an end user per the questions he is asking. Broniatowski [10] points out that the interests of the user dictate whether explainability or interpretability should be sought in the development of new machine learning systems. However, to the general public, the two concepts are relatively interchangeable [12, p. 52141].

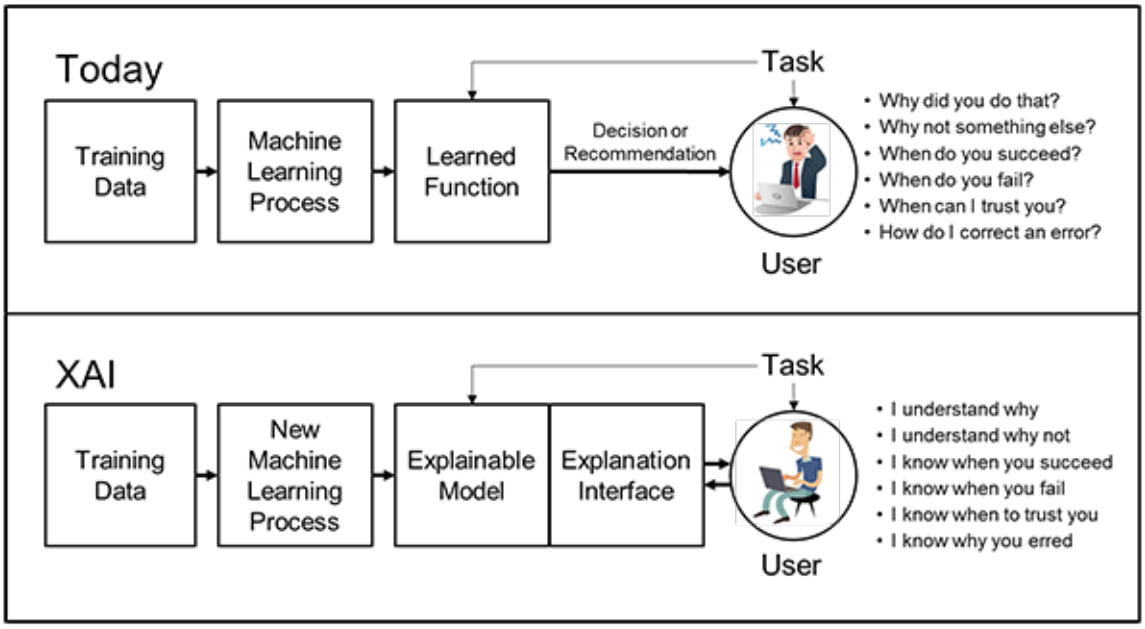

Figure 2.1. Concept of XAI, Source: [9]

A simple yet illustrative example:

> Consider a car with a “check engine" light that is illuminated. An *explanation* might indicate that the check engine light turned on because the car’s internal programming detected fuel flow irregularities. However, the *interpretation* for the driver is that the car needs to be taken to a mechanic for further evaluation [10, p. 4].

Negative attention towards AI use in predicting good renters. The algorithms have been denounced as discriminatory. Additionally, there have been concerns about data privacy.

Here, interpretability could help by creating transparency into why a decision is made. Human users contextualize output (applicant in question represented by data point) given their background knowledge. While ML used training data and algorithmic models. The human assessor concludes an applicant as risky because of the absence of rental history. Machine learning algorithms may associate the length of rental history with success. While both interpretations point to the importance of rental history, the human interpretation is flexible while the algorithmic one is "brittle." The interpreted output gives the applicant a useful insight into "how" to be successful in the future which is to establish a rental history. In the end, domain knowledge is key to interpretability [10].

Explanation in the context of the rental applicant's rejection goes about understanding the situation from a different perspective - How did the decision to reject come to be? If the model was logistic, and when the data point for the applicant is plugged in, then the probability of success is below the threshold. With access to the equation and expertise, the human assessor can observe the largest marginal contribution to the algorithm's decision and again arrive at the importance of rental history in the decision-making process. However, with modern models, the outputs are not easily assessed by humans. Human-made causal explanations are subject to systemic bias [10].

CHAPTER 3:
Feature Extraction for Audio Analysis using Mel Frequency Cepstral Coefficients

The fundamental task in audio analysis is to make computers sense the acoustic environment. Many simple sound waves make up complex sound waves [13]. Machine hearing is complex because acoustic surroundings are diverse with many sounds existing at a point in time, the presence of background noise, the physical distance of the sound source, and the nature of the sound as artificial or natural [14]. To improve accuracy in machine hearing tasks, experts suggest that analysts focus on a subdivision of sound as outlined in the sound classification Figure 3.1 [15], [16].

Characteristics of different sounds vary. For example, human speech and music exhibit repeated stationary patterns, while environmental noises can have theoretically infinite structures or none all. Therefore, it is important in audio analysis to choose the feature extraction methodology that highlights the targeted signal characteristics most accurately [17]. Experts have developed techniques to highlight components of sounds specific to human senses known as perceptual. In Figure 3.1, the differentiation between perceptual and physical techniques is presented succinctly.

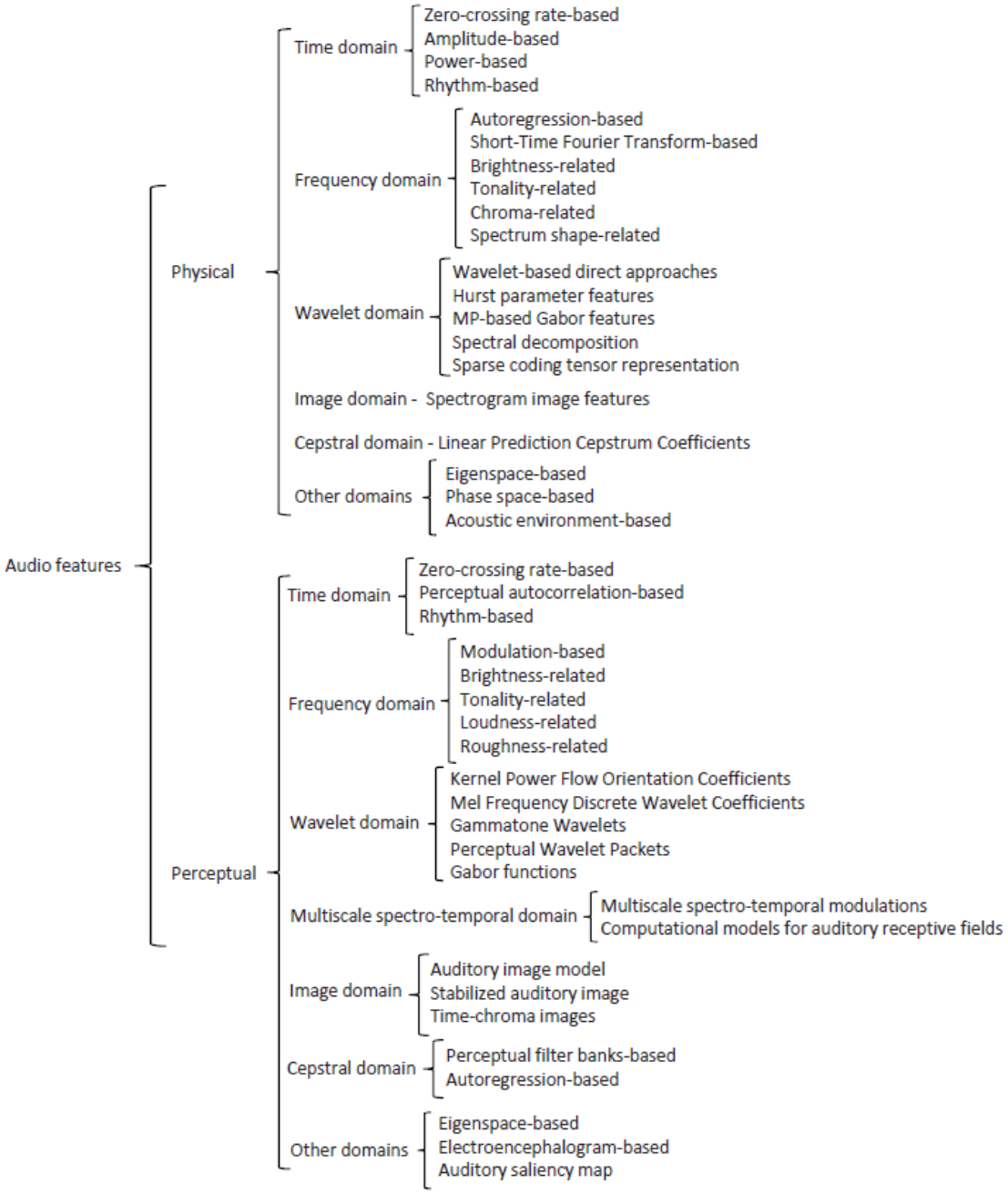

Figure 3.1. Taxonomy of audio feature extraction techniques Source: [15, figure 5]

Speaker recognition is a subset of analyses of the human voice as shown in figure 3.2. In speaker recognition, a machine learning algorithm draws on voice characteristics to identify the speaker. Human speech and the human voice have been at the center of scientific discovery in audio for decades [18]. In this research, I will draw on the mel frequency cepstral coefficients (MFCC) which stand out as the most important feature extraction methods for capturing qualities of the human voice speech recognition [18].

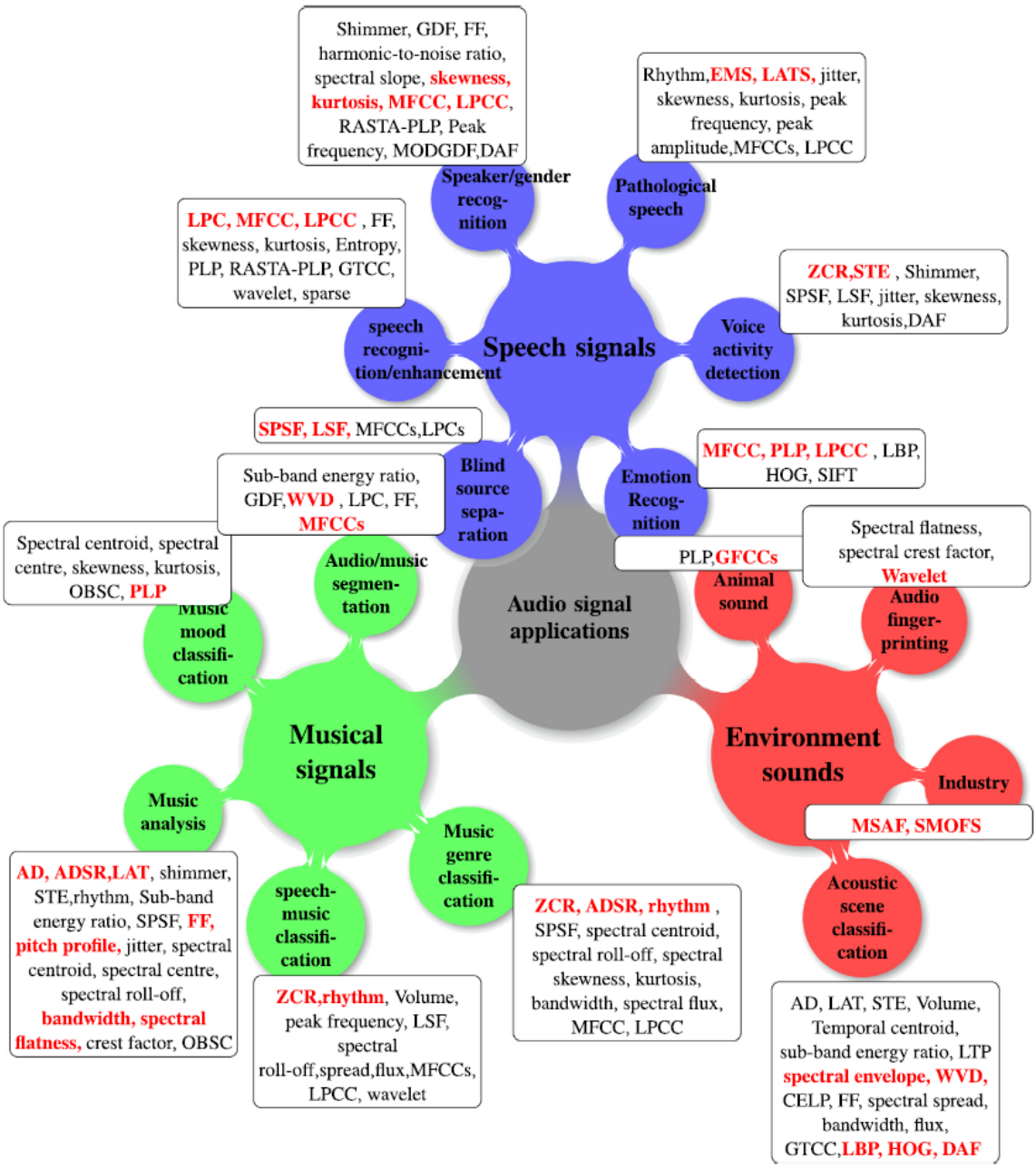

Figure 3.2. Prominent feature extraction methods by audio analysis applications. Source: [17, figure 21]

In contemporary speaker and speech research, the use of the MFCC is mixed. For example, in the task of detecting spoofed audio in the automatic speaker verification competition of 2017, MFCC is listed as an important front-end feature extraction method [19]. While Nagrani et al. [4] use deep learning for speaker recognition, the MFCC is characterized as less useful.

3.1 Mel Frequency Scale

While the cepstrum is the principal concept in the MFCC, I will start with the simpler concept for this section on feature extraction, the mel. The mel scale characterizes the psychological magnitude of pitch. Stevens [20] assigned the frequency 1000 Hz a new unit of measurement for perceived pitch (frequency), such that 1000 Hz was equivalent to 1000 mels. From 125 to 12,000 Hz, the mel scale quantifies the pitch of sinusoidal tone based on half-pitch estimates for frequencies.

Warren [21] reviews the early literature that established the mel scale and points out the role of the octave. An octave is a half-frequency which is also half-pitch. Therefore, the octave is the physical correlate of the mel scale within the musical scale with an upper bound at 4,500 Hz. Pitch above the limit of the musical scale is meaningless [22].

Mel-scaling emphasizes the mid-frequency bands. In seminal research on MFCC's, Davis and Mermelstein [23] succinctly describe the usefulness of the mel in the cepstral feature extraction as suppressing "insignificant spectral variation in higher frequency bands" [23, p. 364]. The authors find that using the mel-frequency is advantageous relative to a linear-frequency.

For speech recognition, the relationship between the real frequency in Hz (F_{Hz}) and perceived frequency in mels (F_{mel}) is approximated by Fant [24] as:[2]

$$F_{mel}(F_{Hz}) = \frac{1000}{log\, 2}\left(1 + \frac{F_{Hz}}{1000}\right) \tag{3.1}$$

which Koenig [26] maps as more or less linear below 1000 Hz and above as logarithmic [27,

[2] A single formula for the mel scale does not exist [25]

p. 380]

Listener-dependent bias has been established in the mel scale [28]. Researchers have suggested improvements by focusing on equal energy [29], [30]. In Greenwood's own response, he states:

> ...the mel scale's popularity has had little justification. There have been good reasons for not using the mel scale for many years. A major one was that it (the 1940 mel scale) was not replicated by [31] nor checked by anyone else (so far as I know) until 1956 (at Steven's behest [32]). The full results of that 1956 check (not actually intended to be a check of the mel scale itself - though the results turned out that way) were published in Hearing Research in 97 [28] [29, paragraph 5]

Therefore, it seems that while the mel scale is flawed the concept is working in practice. Improving upon the mel-scaled cepstrum may be a fruitful area of research [30].

3.2 Cepstral Analysis

Before moving on to the cepstral coefficients, let us understand the concept of cepstral analysis. In Chapter 6, "Cepstral Analysis," Deller [27] draws an enlightening analogy between the use of the *spectrum* in "frequency domain" and the *cepstrum* in the "spectral domain."

Engineers are well-versed in decomposing added components in a complex sequence. Drawing from Deller [27, pp. 352-355], if we have a magnitude spectrum of $x(n)$ and n is a discrete value of time:

$$x(n) = x_1(n) + \omega(n) \tag{3.2}$$

where $x_1(n)$ is a low-frequency signal and $\omega(n)$ is high-frequency noise, then the Fourier transform will allow us to examine these components of $x(n)$ individually. We can assess

the "separation" of the $x_1(n)$ and $w(n)$ and derive information about the parts when the signal is represented with the *spectrum*. Moore [33] states,

> By applying the Fourier transform to the waveform of a sound, we can mathematically determine just which amounts of which frequencies are responsible for that particular waveshape. [33, p. 43]

A linear separation of a signal is useful in practice in order to remove noise from a signal. For example, from equation 3.2, we can obtain $x_1(n)$ by applying a lowpass filter, F, to remove the high-frequency component $w(n)$. From [27, p. 353],

$$F\{x(n)\} = F\{x_1(n) + \omega(n)\} = F\{x_1(n)\} + F\{w(n)\} \approx x_1(n) \tag{3.3}$$

Therefore, applying the filter, F on $x(n)$ we get $x_1(n)$ and transform the results back to the time domain.

Speech is not additive, but "is composed of an excitation sequence convolved with the impulse response of the vocal system model" [27, p. 352].

$$s(n) = e(n) * \theta(n) \tag{3.4}$$

The *cepstrum* was developed specifically to represent a transformation on the speech signal and is a special case of homomorphic signal processing [34], [35]. The transformation has two properties:

1. The representatives of the component signals will be separated in the cepstrum.
2. The representatives of the component signals will be linearly combined in the cepstrum [27, p. 353].

Following, Deller [27, p. 354], we notate for signal $s(n)$ the real cepstrum is $c_s(n)$ and the short-term real cepstrum frame ending at m is $c_s(n;m)$. While the short-term real cepstrum

is used in practice, Deller [27] uses the real cepstrum to describe the concepts. The real cepstrum is defined as:

$$c_s(n) = \mathscr{F}^{-1}\{log|\mathscr{F}\{s(n)\}|\} \tag{3.5}$$

Where $\mathscr{F}$ is the Discrete-time Fourier transform. In practice, the component of equation 3.5, $log|\mathscr{F}\{s(n)\}|$ is real and even which in turn makes the application of $\mathscr{F}^{-1} \equiv \mathscr{F} \equiv$ Discrete Cosine Transformation.

Figure 3.3 depicts the operations leading from $s(n)$ to $c_s(n)$. The steps are:

1. The operations $DFTF \equiv \mathscr{F}$ and $log|\cdot|$ transform $s(n)$ into a "linear" domain. Conceptually, the operator is unraveling the convolution.
2. Then apply Fourier analysis to the new “signal" to view the “frequency domain" properties.
3. Interpret the real cepstrum as the Fourier series “line spectrum" of the “signal."

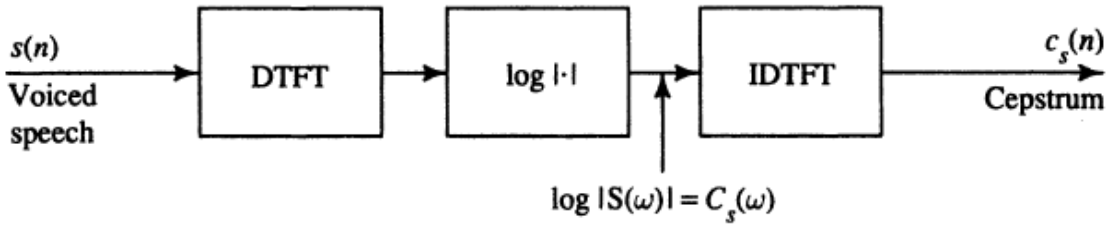

Figure 3.3. How to compute the real cepstrum. Source: [27, p. 356] figure 6.1

However, all the terms in quotations in the operation are special as the “signal" being transformed into the frequency domain is already in the frequency domain. What we understand as the role of spectrum in the frequency domain, here is the role of “cepstrum" in the “quefrency" domain. With real cepstrum analysis, we can resolve the dilemma in equation 3.4 as

$$c_s(n) = c_e(n) + c_\theta(n) \tag{3.6}$$

The convolution theorem underlies the decomposition shown in equation 3.6. Deller [27, p. 361] illustrates the steps laid out in figure 3.3 in figures 3.4-3.6. The speech signal spectrum $S(\omega) = E(\omega)\Theta(\omega)$. First, in figure 3.4, $|S(\omega)|$ is shown to be composed of spectral variation $|E(\omega)|$ convolved with the vocal system $|\Theta(\omega)|$.

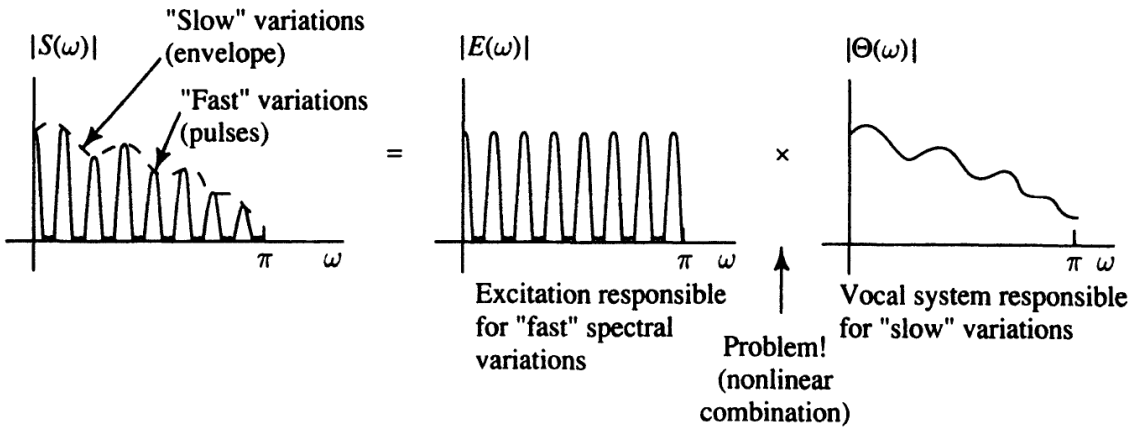

Figure 3.4. Convolved speech spectrum. Source: [27, p. 361] figure 6.3 (a)

Next, in figure 3.5, the logarithm of the spectral magnitudes the additive correlates for $C_s(n)$ are revealed.

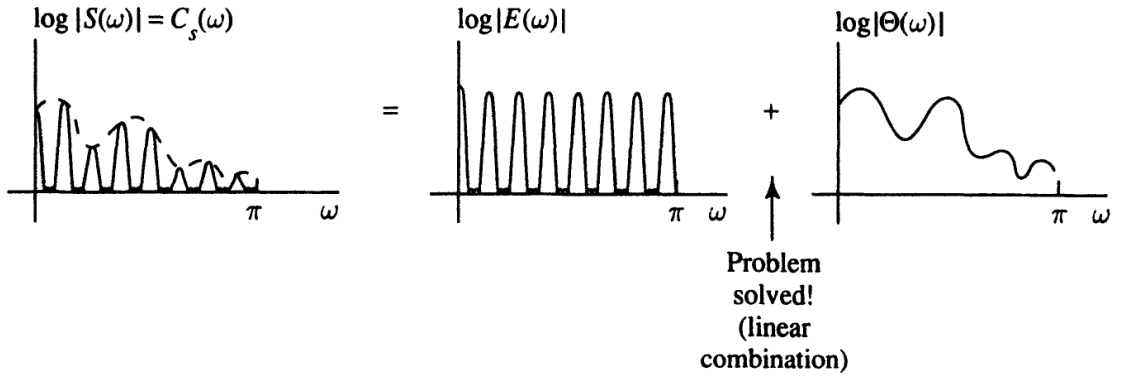

Figure 3.5. Linearly separable speech spectrum. Source: [27, p. 361] figure 6.3 (b)

Finally, in Figure 3.6, the IDTFT an approximation provides the quefrency domain properties

$P,\ 2P,\ 3P, \ldots$ of the components of $c_s(n)$ from equation (3.6). "The low-quefrency part of the cepstrum therefore represents an approximation to the cepstrum of the vocal system impulse response, $c_\theta(n)$. The high-quefrency part corresponds to the cepstrum of the excitation, $c_e(n)$" [27, p. 361].

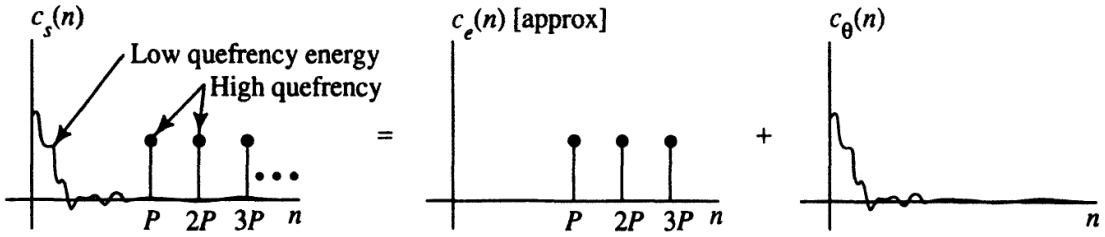

Figure 3.6. Cepstral analysis of linearly separable spectrum. Source: [27, p. 361] figure 6.3 (c)

3.3 Mel Frequency Cepstral Coefficients

To summarize the seminal results of Davis and Mermelstein [23] concerning their task of differentiating between two speakers, they find that six MFCCs captured most of the relevant information. The authors also conclude that a trade-off exists between the number of cepstral coefficients and the frame size, adding that computationally it is more advantageous to have a coarser resolution and more coefficients. Fast-forward 40 years and the MFCC extraction algorithm is virtually unchanged and 2 to 13 coefficients are suggested by experts.

In addition to the cepstral coefficients, their "derivatives" are used in speech recognition and identification systems to measure the vocal system spectrum and dynamics [27]. The "derivatives" can be computed as follows:

$$\Delta c_s(n;m) \equiv c_s(n;m+\delta Q) - c_s(n;m-\delta Q) \tag{3.7}$$

where $\Delta c_s(n;m)$ is the mel-cepstrum for the frames of the signal $s(n)$ ending at time $m+\delta Q$ differenced with the cepstrum at frame $m-\delta Q$ for all n. In equation (3.7), Q is the number of samples by which the window is shifted for each frame and δ smooths the estimate. $\Delta c_s(n;m)$ captures spectral changes since the previous frame, but lack their own meaning.

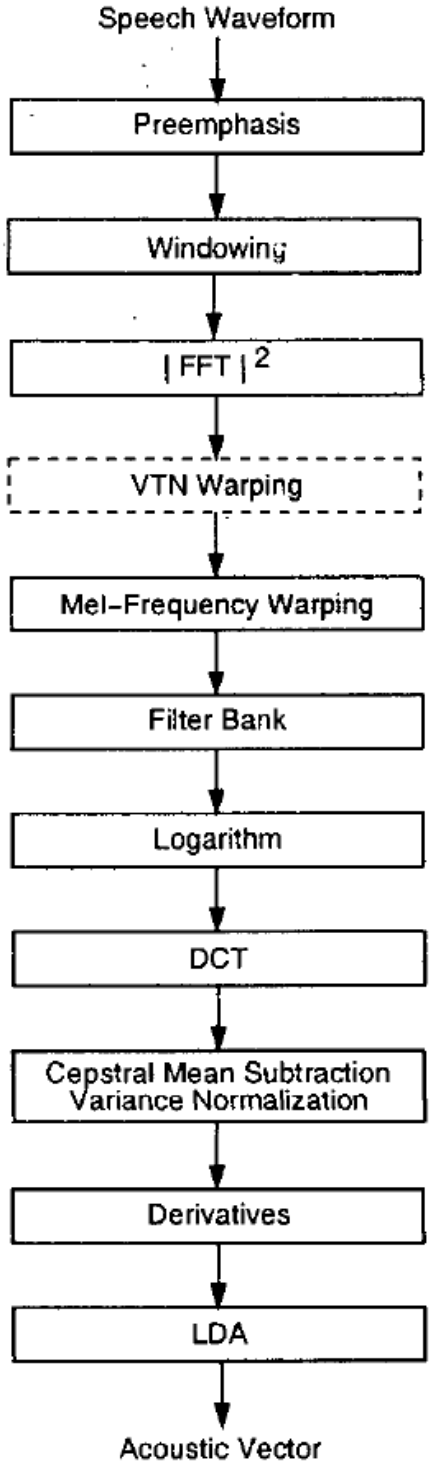

Figure 3.7. Signal analysis using the cepstral coefficients. Source: [36, figure 1, p. 27]

Following figure 3.7, Molau [36] lays out the steps of computing cepstral coefficients. Specifically, these steps create an acoustic vector of dimension 25 to 50 for every 10 ms of sound:

1. The speech waveform, sampled at 8 or 16 kHz, is first differentiated (preemphasis) (p. 73).[3]
2. The waveform is cut into a number of overlapping segments (windowing), each 25 ms long and shifted by 10 ms. (p. 73)
3. A Hamming window is multiplied and the Fourier transform (FFT) is computed for each frame (p. 73).
4. The power spectrum is warped according to the Mel-scale in order to adapt the frequency resolution to the properties of the human ear (p. 73).
5. The spectrum is segmented into a number of critical bands by means of a filterbank. The filterbank typically consists of overlapping triangular filters (p. 73).
6. A discrete cosine transformation (DCT) applied to the logarithm of the filterbank outputs results in the raw MFCC vector. The highest cepstral coefficients are omitted to smooth the cepstra and minimize the influence of the pitch which is irrelevant for the speech recognition process (p. 73).[4]
7. The mean of each cepstral component is subtracted, and the variance of each component may also be normalized (p. 73).
8. The MFCC vector is augmented with time derivatives (p. 73).
9. (Optional) transformations like linear discriminant analysis (LDA) may further increase the temporal context and the discriminance of the acoustic vector [36, p. 73].

Sigurdsson [39] details the steps. Starting with step 3 from the above enumerated list, the DFT × each time window for the discrete time signal is computed using the equation:

$$X(k) = \sum_{n=0}^{N-1} \omega(n)x(n) \exp\left(\frac{-j2\pi kn}{N}\right) \qquad k = 0, 1, \ldots, N-1 \tag{3.8}$$

[3]The first differentiation is a type of filter and aims to make the audio signal stationary [37].
[4]The highest cepstral coefficients are omitted through a “liftering" using a lowpass window [38].

where $x(n)$ is the discrete-time signal and N is the length of $x(n)$. The variable k corresponds to the frequency, $f(k) = \frac{kf_s}{N}$ where f_s is the sampling frequency (hz). The Hamming window is often used in speech analysis to compute $w(n)$, where:

$$w(n) - 0.54 - 0.46\cos\left(\frac{\pi n}{N}\right) \tag{3.9}$$

Next, Sigurdsson [39] combine steps 4 and 5 from the above enumerated list using the equation:

$$X'(m) = \ln\left(\sum_{k=0}^{N-1} |X(k)| \cdot H(k,m)\right) \qquad m = 1, 2, \ldots, M \tag{3.10}$$

where $X'(m)$ is the natural log of the magnitude (power) spectrum $|X(k)|$ is logarithmically scaled in frequency by the Mel filter bank $H(k,m)$. The number of filter banks is given by M such that $M << N$. Computations with $H(k,m)$, scale not only the frequency of $|X(k)|$, but also its magnitude [39].

Finally, the raw acoustic vector of MFCC described in step 6 shown in the enumerated list above is obtained with the discrete-cosine transform of $X'(m)$ as follows:

$$c(l) = \sum_{m=1}^{M} X'(m)\cos\left(l\frac{\pi}{M}\left(m - \frac{1}{2}\right)\right) \qquad l = 1, 2, \ldots, M \tag{3.11}$$

where $c(l)$ is the lth coefficient.

Transitioning through the equations 3.8 to 3.11, we can follow the unit of observation from time, $n = 1, 2, \ldots, N$, in $x(n)$ and $w(n)$ as inputs to equation 3.8 which results in a frequency unit, $k = 1, 2, \ldots, N-1$, for the spectrum $X(k)$. Then the spectrum, $X(k)$, is transformed from frequency, k, to mel-scaled frequency, $m = 1, 2, \ldots, M$. Finally, in equation (3.11) all the mel-scaled frequencies, m, used to calculate each MFCC, l where number of possible MFCCs is limited by the number of mel filters chosen in equation (3.10).

Paliwal [40] bases his research into filterbank energies on a critique of MFCCs. The authors

state that:

> Though MFCCs have been very successful in speech recognition, they have the following two problems: 1) They do not have any physical interpretation, and 2) Liftering of cepstral coefficients, found to be highly useful in the earlier dynamic warping-based speech recognition systems, has no effect in the recognition process when used with continuous observation Gaussian density hidden Markov models [40, p. 1].

Part of this thesis will examine the loss in accuracy that may occur due to using few MFCCs instead of the more typical features like the spectrogram, over a high number of MFCCs.

3.4 Visualizing the Cepstrum and Mel Cepstral Coefficients

This thesis aims to be more familiar with the feature input to increase interpretability. While the training dataset is built on Vox Celeb 2.0 (see section 4.1), those utterances are heterogeneous [4]. To visualize the MFCCs, instead, I will use the utterances assembled into a dataset in [41]. One portion of the [41] dataset, compiles utterances from 150 speakers all of which speak the phrase, “Machine learning 1, 2, 3, 4, 5, 6, 7, 8, 9, 10" in a noisy environment.

In the following two figures, Figure 3.8 and Figure 3.9, I graph the first, second, and third MFCC for one male and one female speaker who are both 21 years old. For perspective on the default implementation of MFCC in torchaudio from PyTorch, the default number of MFCCs returned is 40, which is far above the number suggested in the early literature.

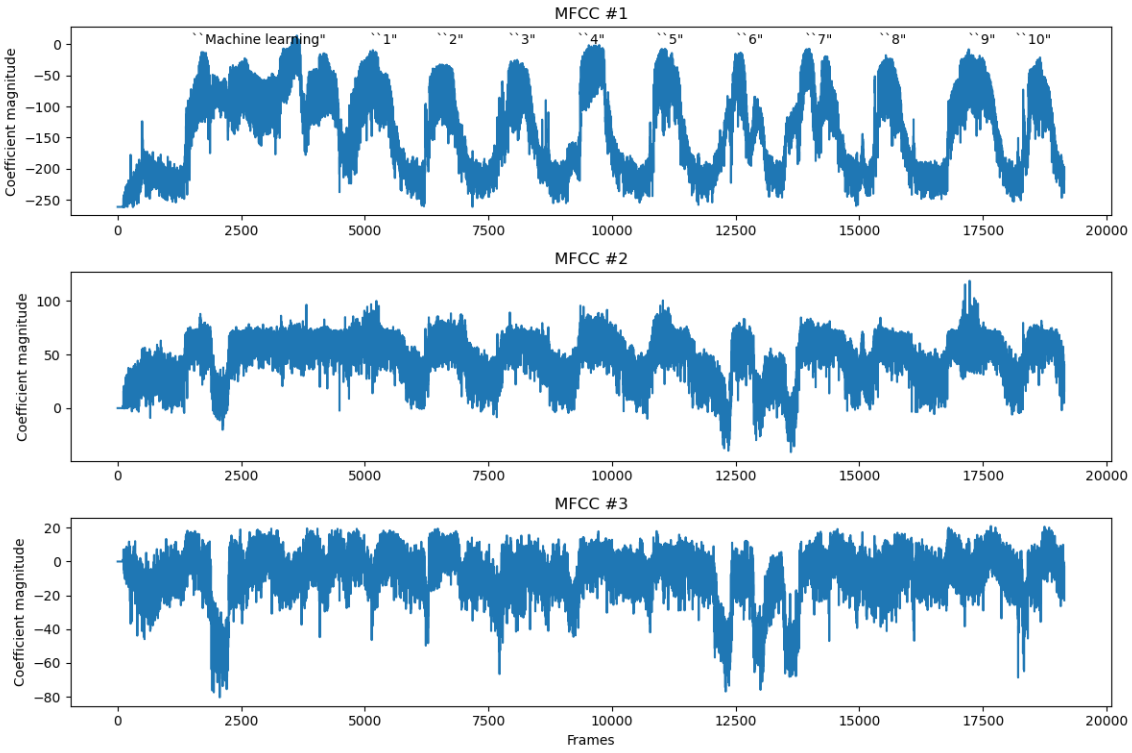

Figure 3.8. Mel frequency cepstral coefficients for the speech from 21-year-old male speaker saying "Machine learning 1, 2, 3, 4, 5, 6, 7, 8, 9, 10"

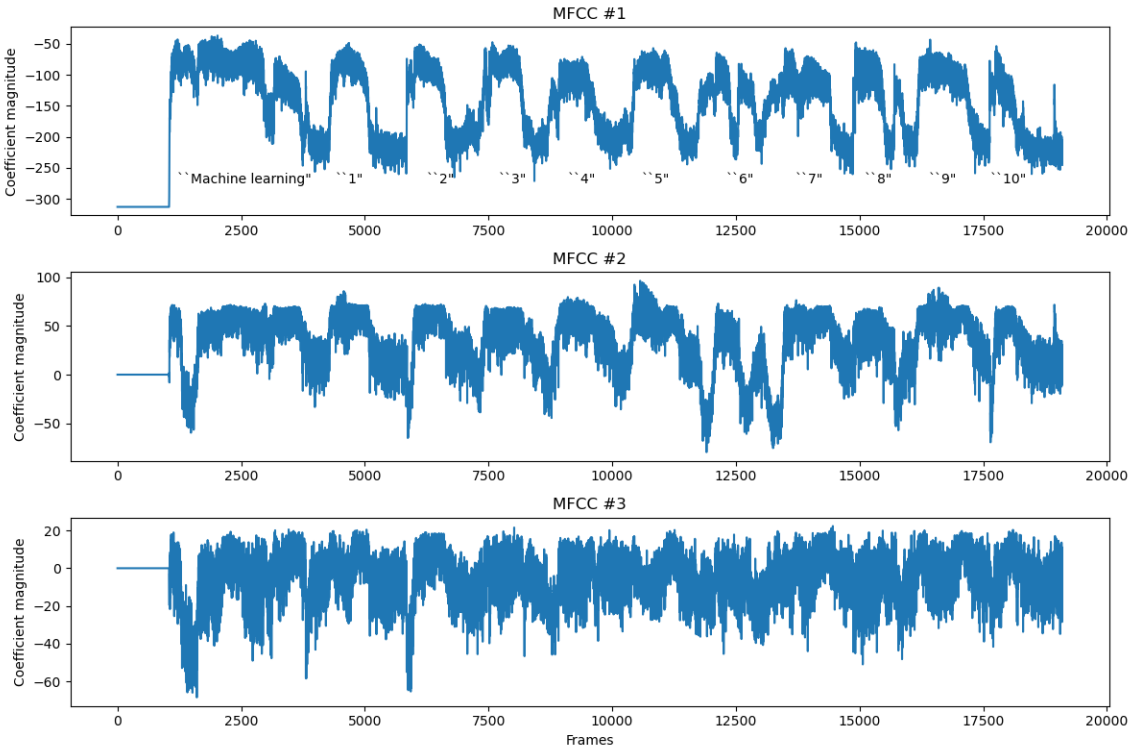

Figure 3.9. Mel frequency cepstral coefficients for the speech from 21-year-old female speaker saying "Machine learning 1, 2, 3, 4, 5, 6, 7, 8, 9, 10"

To be transparent about the parameters that I used to create these coefficients, I detail the code in listing 3.1.

Listing 3.1: Creating a visual of the first three MFCCs

```
import matplotlib.pyplot as plt
import numpy as np
import argparse
import os
import torch, torchaudio

    filename = os.path.join(args.alsaify_directory, same_speaker)
    data, fs = torchaudio.load(filename)

    # sample rate is 48,000 for Alsaify data
    sample_rate = fs
    n_fft = int(sample_rate*0.03) #30 ms window
    n_mels = 20
    n_mfcc = 3

    transform = torchaudio.transforms.MFCC(
        sample_rate=sample_rate,
        n_mfcc=n_mfcc,
        melkwargs={
            "n_fft": n_fft,
            "n_mels": n_mels,
            "mel_scale": "htk",
            "window_fn": torch.hamming_window
        },
    )

    mfcc = transform(data)
    mfcc = torch.squeeze(mfcc)
    mfcc_np = torch.Tensor.numpy(mfcc)
    mfcc_first = mfcc_np[0, :]
    total_time = mfcc_np.shape[1]
    time = list(range(total_time))
```

I have explored the window size that I needed to find a relatively stationary wave for the

speech from the female speaker. I found that around 30 milliseconds during a speaking segment were stationary. Lei, et al. Lei, et al., use “[t]he 0th through 12th coefficients of either the MFCCs, LFCCs, or a-MFCCs (with 25 ms windows and 10 ms intervals) with deltas and double deltas are used" [42, p. 2].

The number of mels (filterbanks) used in the default transform function is 128, which is much larger than that used in the earlier literature. Paliwal [40] identifies the use of 20 to 60 filterbanks as typical in the literature.

CHAPTER 4:
Data

Building the dataset for modeling speaker recognition was a major task in this research project. I downloaded the audio files as a massive dataset and converted over a million files into the .wav format. Next, I organized the file paths such that I could script a function to make random combinations of speakers and files. I also calibrated the feature extraction method and applied it to the raw audio data to make a multidimensional audio file into a numerical matrix that a computer can analyze.

4.1 VoxCeleb 2.0 Dataset

I train the machine learning model using the VoxCeleb dataset of humans to determine speaker recognition [4]. The dataset contains over 1 million utterances that were captured from YouTube videos of celebrity interviews, news shows, talk shows, and debates. In addition to the utterances of focus, the tracks also contain background chatter, laughter, and overlapping speech. The tracks are curated and each segment is 3 seconds long.

VoxCeleb includes a heterogeneous set of more than 7000 speakers. Each track is labeled with the identity of the celebrity speaker. Speakers' genders are about 1:2 female to male and 1:2 American to non-American speakers [43].

4.2 Creating Same and Different Speaker Combinations

Datasets with multiple speakers often keep audio samples of the same speaker in a folder together. For this research, I need to draw an equal quantity of randomly paired audio files from the same speakers and different speakers. I undertook this task in a series of steps.

First, I populated a list with the full file paths of every .wav file in the train, validation, and test datasets together using a custom recursive function presented in Listing 4.1.

Listing 4.1: Custom function to record the full file path of audio files within the complex VoxCeleb 2.0 dataset file structure

```
import pathlib
```

```
def find_all_files(directory):
    all_files = list(pathlib.Path(directory).rglob("*.[mw][4a][av]"))

    pure_paths = []

    for line in all_files:
        pure_paths.append(line.as_posix())

    return pure_paths
```

Next, I split the file path to identify the speaker and the dataset type and instantiated a dictionary for each dataset type to store the speaker identifiers as the keys and the list of associated full file paths to the audio clips for the respective speaker at the values.

Using these dictionaries, I utilized a nested set of custom functions to create a balanced sample of pairs of audio. The function that pairs randomly selected audio clips from two different randomly selected speakers first identifies two random keys in the dictionary (speakers) and then randomly selects a value from the list associated with each key. More simply, the function that pairs randomly selected audio from one randomly selected speaker, just randomly selected a key from the dictionary and then randomly selects two items from the list of values.

Finally, the broader custom function unites the two lists, “same" speaker and “different" speaker, is presented in the following code snippet, Listing 4.2.

Listing 4.2: Four functions to randomly find and record pairs of audio clips from the same speaker and different speakers

```
import random
import os
import pickle

def include_keys(dictionary, keys):
    """Filters a dict by only including certain keys."""
    key_set = set(keys) & set(dictionary.keys())
    return {key: dictionary[key] for key in key_set}
```

```
def random_different(dictionary, size, different_list = []):
    """Create list of random pairs from different speakers"""
    for _ in range(0,int(size)):
        # randomly select two id's
        keys = [random.choice(list(dictionary)) for i in range(2)]
        if keys[0]!=keys[1]:
            pair = []
            for key in keys:
                files = dictionary.get(key)
                random_file = random.choice(list(files))
                pair.append(random_file)
            pair.append(False)
            different_list.append(pair)

        else:
            continue

    return different_list



def random_same(dictionary, size, same_list = []):
    """Create list of random pairs from different speakers"""
    for _ in range(0,int(size)):
        # randomly select two id's
        key = random.choice(list(dictionary))
        files = dictionary.get(key)
        random_files = [random.choice(list(files)) for i in range(2)]
        random_files.append(True)
        same_list.append(random_files)

    return same_list


def create_sample(id_type, id_type_name, type, total_size, sample = []):
    """Create list of random pairs from same and different speakers"""

    # bring in dictionary_pickle – dev or test
```

```
annotation_dictionary = "/annotation_dict_" + type + ".pickle"
filepath = os.path.join(args.annotation_directory + annotation_dictionary)
with open(filepath, "rb") as f:
    my_dict = pickle.load(f)

# refine dictionary to split typee
new_dict = include_keys(my_dict, id_type)
print("Old dictionary length: ", len(my_dict))
print("New dictionary length: ", len(new_dict))

same_list = random_same(new_dict, total_size/2)
print("Same list length: ", len(same_list))

diff_list = random_different(new_dict, total_size/2)
print("Diff list length: ", len(diff_list))

sample = list(islice(reversed(same_list), 0, int(total_size/2)))
       + list(islice(reversed(diff_list), 0, int(total_size/2)))
sample.reverse()
print(len(sample))
sample_file = '/sample_' + id_type_name + '.pickle'
savepath = os.path.join(args.annotation_directory + sample_file)
with open(savepath, 'wb') as g:
    pickle.dump(sample, g)
```

I make a persistent list of audio clip combinations for each needed dataset type ready for the custom PyTorch Dataset classes detailed in section 4.3.

4.3 Pytorch Dataset Class and Data Loader

I created a custom dataset for VoxCeleb 2.0. Pytorch custom dataset is an abstract class that inherits the Dataset class. A custom Dataset override the *__init__* and *__getitem__* methods. In my custom *__init__*, I call on the persistent list of audio combinations created using the combination generator functions described in section 4.2. I leave the extraction of the audio data to *__getitem__* to only read the files as required and not store the entire dataset in memory.

For each audio clip in the paired clips, *__getitem__* uses PyTorch audio method to extract the audio contents. Within *__getitem__*, apply a series of private methods to process the audio content for analysis.

Listing 4.3: Custom VoxDataset

```
class VoxDataset(data.Dataset):
    """
    A standard PyTorch definition of Dataset which defines the functions
    __len__ and __getitem__.
    """

    def __init__(self, pickle_name, frontend):
        # def __init__(self, data_dir, transform):
        """
        Store the filenames of the wav to use. Specifies transforms to
        apply on files.
        Args:
            pickle (string): Path to the pickle file with annotations
            root_dir (string): Directory with all the .wav files
            feature_extraction_method (torchvision.transforms): Transform
            to be applied to sample
        """
        filepath = os.path.join(args.pickle_directory + '/'
                                + pickle_name + '.pickle')
        with open(filepath, "rb") as f:
            list_of_list = pickle.load(f)
        self.vox = pd.DataFrame(list_of_list)

        self.frontend = get_feature_extractor(frontend)

    def __len__(self):
        # return size of dataset
        return len(self.vox)

    def __getitem__(self, idx):
        """
        Fetch index idx audio and labels from dataset. Transforms audio.
        Args:
```

```
            idx: (int) index in [0, 1, ..., number_of_files-1]
        Returns:
            waveform, sample rate: (Tensor) audio
            label: (int) corresponding label of audio
        """
        if torch.is_tensor(idx):
            idx = idx.tolist()

        wavefiles = self._get_audio_sample_paths(idx)
        label = self._get_audio_sample_label(idx)

        t1, t2 = [self._process_file(file) for file in wavefiles]

        assert t1[1] == t2[1], "The pairs of tensors differ in shape"

        return t1[0], t2[0], label

    # private functions
    def _get_audio_sample_paths(self, idx):
        wavefiles = self.vox.iloc[idx, 0:2]
        return wavefiles

    def _get_audio_sample_label(self, idx):
        label_boolean =  self.vox.iloc[idx, 2]
        return label_boolean.astype(float)

    def _cut_if_necessary(self, signal):
        if signal.shape[1] > 48000:
            offset = 8000
            signal = signal[:, offset:(offset+48000)]
        return signal

    def _process_file(self, filepath):
        signal, sr = torchaudio.load(filepath)
        signal = self._cut_if_necessary(signal)
        signal = self.frontend(signal)
        signal = torch.squeeze(signal)
        signal = self._standardize(signal)
        signal = torch.transpose(signal, 0, 1)
```

```
        dim = signal.size()
        return signal, dim

    def _standardize(self, signal):
        # Shape: D x Tmax
        signal -= signal.mean(axis=0)
        signal /= signal.std(axis=0)
        return signal
```

PyTorch's dataloader allows a user to pass samples in "minibatches" from the VoxCeleb 2.0 to my model. At every epoch, the dataloader reshuffles the data and the dataloader also easily allows multiprocessing in data retrieval. The dataloader is an iterable object and returns training set features and labels. My implementation of the dataloader function is presented in Listing 4.4.

Listing 4.4: Custom VoxDataset PyTorch DataLoader

```
def fetch_dataloader(pickle_name, feature_extraction_method,
                                                batch_size, params):

    dl = data.DataLoader(VoxDataset(pickle_name, feature_extraction_method),
                                        batch_size,
                                        shuffle=True,
                                        num_workers=params.num_workers,
                                        pin_memory=True)

    return dl
```

4.4 Extracting Mel Frequency Cepstral Coefficients

Using well-known MFCCs is a fundamental choice for interpretability in my thesis modeling task for speaker recognition. Below I provide an example of my methods for extracting MFCC using the PyTorch torchaudio function that creates a tensor. The sample rate of 16,000 is specific to the VoxCeleb 2.0 dataset. I choose to use the default 128 mel filters and extract the first three MFCCs. According to Rudin [3], humans are only able to be familiar with $7 +/- 2$ features, which is why I want to start with very few powerful features.

Davis and Mermelstein [23] suggest that a vast majority of useful information about the speaker is contained in the first few coefficients. In Listing 4.5, I present my custom function for defining MFCCs. I use this function as the *feature_extraction_method* input for the dataloader function presented in Listing 4.4.

Listing 4.5: Extracting a Tensor of MFCCs

```
def get_feature_extractor(method, sr):
    """
    Args:
        method: (str) audio feature extraction
    Returns:
        transform method
    """
    sample_rate = sr
    n_fft = int(sample_rate*0.03) #30 ms window
    n_mels = 128


    if method == 'mfcc3':
        n_mfcc = 3

        transform = torchaudio.transforms.MFCC(
            sample_rate=sample_rate,
            n_mfcc=n_mfcc,
            melkwargs={
                "n_fft": n_fft,
                "n_mels": n_mels,
                "mel_scale": "htk",
                "window_fn": torch.hamming_window
            },
        )


    if method == 'mfcc12':
        n_mfcc = 12

        transform = torchaudio.transforms.MFCC(
            sample_rate=sample_rate,
```

```
            n_mfcc=n_mfcc,
            melkwargs={
                "n_fft": n_fft,
                "n_mels": n_mels,
                "mel_scale": "htk",
                "window_fn": torch.hamming_window
            },
        )

    if method == 'mfcc40':
        n_mfcc = 40

        transform = torchaudio.transforms.MFCC(
            sample_rate=sample_rate,
            n_mfcc=n_mfcc,
            melkwargs={
                "n_fft": n_fft,
                "n_mels": n_mels,
                "mel_scale": "htk",
                "window_fn": torch.hamming_window
            },
        )

    return transform
```

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 5:
Results

In this chapter, I present comparisons between models and feature extraction methods. In Section 5.1, I compare the performance of two CNNs using 40 MFCCs but with different window sizes used during the feature extraction. Interestingly, I find that the feature extraction method that uses a window size at odds with the literature on human speech performs extremely well, while the one using a suggested window size to extract a quasi-stationary speech signal performs moderately well. Next, in Section 5.2, using the suggested window size, I compare the performance of two CNNs using 40 MFCCs and 12 MFCCs. Using 13 or fewer MFCCs was common practice before the last decade. I find that the performance is not depleted by using fewer MFCCs. In Section 5.3, using 12 MFCCs and the suggested window size, I compare the performance of using a CNN model versus an MLP model. I find that the MLP does not perform as well. Finally, in Section 5.4, I use a Logistic Regression to model the speaker recognition task with 12 MFCCs and the suggested window size during feature extraction. I find that the model cannot learn any patterns and labels all speaker pairs as the “same" speaker.

5.1 Identifying a Conundrum with CNN modeling of human speaker recognition task using MFCCs

In this section, I compare two attempts at modeling the speaker recognition task using 40 MFCCs with window sizes: 128 ms and 30 ms, where 30 ms aligns with the literature on human speech windowing and 128 ms is the default window size from PyTorch Audio.

In the first iteration, before I learned the specifics of modeling human speech, I modeled the VoxCeleb 2.0 speaker recognition task using 40 MFCCs constructed using the default specifications from the PyTorch torchaudio package, which sets the default number of FFTs equal to 2048. The number of FFTs translates into a window size and hop length by construction. Therefore, when the number of FFT is equal to 2048, then the window size is 128 ms, and the hop length of 64 ms. Below, in Listing 1 in the Appendix, I include the PyTorch class for the CNN using 40 MFCCs and a 128 ms window which I refer to

henceforth as CNN 1.

In CNN 1, I include four convolutional layers and four fully connected layers. The hyperparameters I tuned were the learning rate, dropout, batch size, optimizer (SGD or Adam), and epoch number. The best results I achieved were when I set the SGD momentum scheduler at steps 8, 16, and 24 epochs. Looking at the curves in figure 5.1, the steep loss curve shows that the CNN learned quickly, arriving near its minimum in the test data just after 10 epochs. The training and test curves look similar until then as well. However, this specification still exhibits overfitting fitting characteristics, since the training loss goes to zero. A positive note about the test loss, though, is that it does **not** have a rising trend across the last 20 epochs.

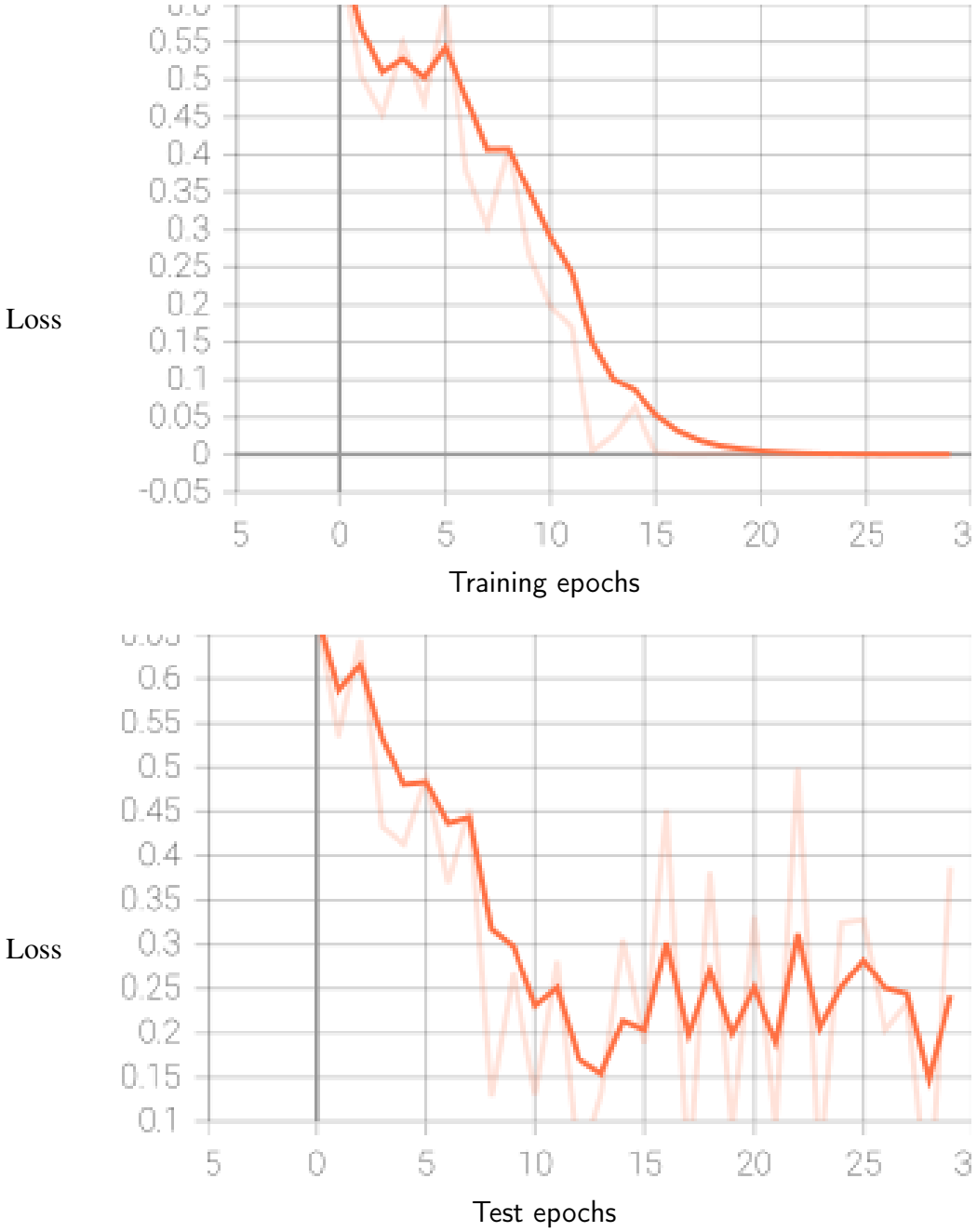

Figure 5.1. CNN 1 with SGD Optimizer, learning rate = 0.3, scheduler set at 8, 16, 24 epoch for momentum = 0.9. The dark line is the smoothed (factor = 1.0) loss curve and the light line is the non-smoothed (factor = 0.0) loss curve.

The detailed information on the evaluation metrics for the model responsible for the training and test curve presented in Figure 5.1 are presented in the bottom rows of the column entitled CNN1 in Table 5.1. Using a large window size of 128ms, my CNN model can achieve a test accuracy of 0.928 and F1 Score = 0.928. The false negatives and false positives are both small, but the false positives are almost three times as likely as the false negatives. Together with the relatively smaller true negative label, it seems like this model over-predicts the “same" speaker.

	CNN 1	CNN 2
MFCCs		
Number of FFT	2048	480
Number of seconds of audio	3	3
Window length (ms)	128	30
Hop length (ms)	64	15
Number of MFCCs	40	40
Number of Mel filters	40	128
CNN		
Training size	100,000	50,000
Test size	20,000	10,000
Number of epochs	30	100
Architecture	see Listing 1	see Listing 2
Convolutional layers	4	3
Linear layers	4	5
Dropout	No	Yes, 0.3
Pooling	Yes	Yes
Non-linearity function	ReLu	ReLu
Learning rate	0.3	0.03
Scheduled step, ($\gamma = 0.1$)	8, 16, 24	35, 40, 45
Test performance, batch = 64		
True positive (# in batch)	30.2	29.6
True negative (# in batch)	29.2	15.6
False positive (# in batch)	2.8	16.5
False negative (# in batch)	1.8	2.4
Accuracy	0.93	0.71
F1 score	0.93	0.76
Loss	0.47	0.53
Loss graphic	see Figure 5.1	see Figure 5.2

Table 5.1. Feature extraction, modeling, and performance details for two CNNs using 40 MFCCs, CNN 1 and CNN 2

However, a 128 ms window is inappropriate for human speech, which is only quasi-stationary from 25-30 ms. When I use the appropriate window length for human speech, window (ms) $\in [20, 30]$, then accuracy on the test set maxes out at a little above 70 percent as shown in the column entitled CNN2 in Table 5.1.

In the code shown in Listing 2 in the Appendix, which I call CNN 2. In CNN 2, I increase the number of fully connected layers to five to ensure the model can learn the patterns and include a dropout of 30 percent to help avoid overfitting, otherwise, CNN 1 and CNN 2 are fairly similar in terms of their architecture. The training specifications I chose for CNN 1 and CNN 2 differ regarding the number of training and test observations. Additionally, I use 128 mel filters with the 30 ms window and 40 mel filters with the 128 ms window. I did not explore the implications of the number of filters. For CNN 1, when the window is 128 ms and there are 40 mel filters, I chose to mimic the PyTorch torchaudio example which sets the number of mel filters equal to the number of MFCCs. However, for CNN2, when the window is 40 ms and there are 128 mel filters, I chose to use the PyTorch torchaudio default number of mel filters.

I found training CNN 2 to be much more difficult than training CNN 1. CNN 2 quickly overfit the training data. It took around 40 attempts over four days to arrive at the model detailed in Listing 2 in the Appendix and column CNN2 in Table 5.1. In the two panels of Figure 5.2, I present the loss curves for the training and test data for CNN 2. The model described learned slowly, shown by the gradual slope of the loss curve. However, the specification does not severely overfit the training data with these parameters, which was a common characteristic of models for this data with a window size of 30 ms.

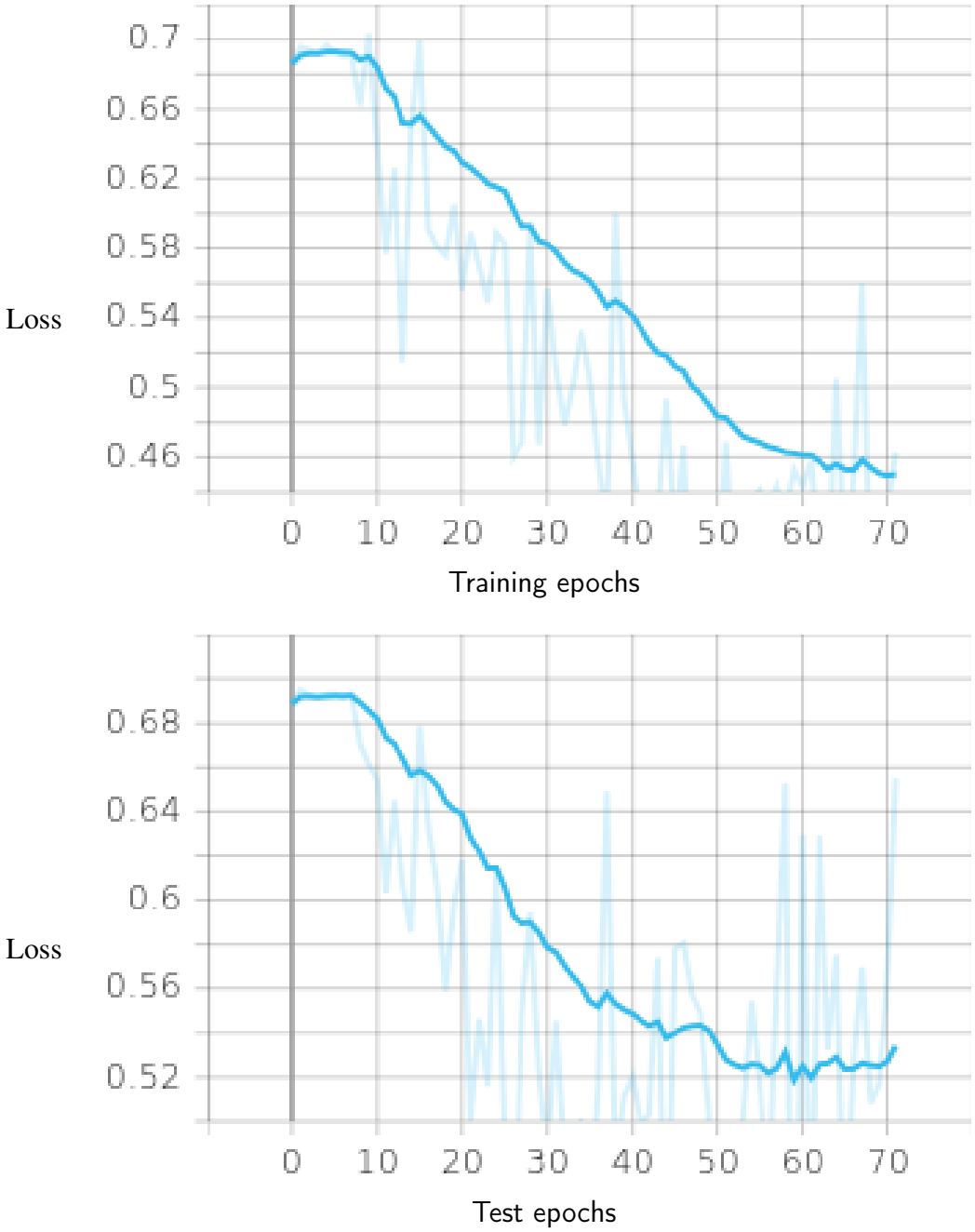

Figure 5.2. CNN 2 with SGD Optimizer, learning rate = 0.03, scheduler set at 35, 40, 45 epochs for momentum = 0.9. The dark line is the smoothed (factor = 1.0) loss curve and the light line is the non-smoothed (factor = 0.0) loss curve.

In the last few rows of column CNN 2 of Table 5.1, I present the details of a confusion matrix for the model from Listing 2 in the Appendix and in the second panel of Figure 5.2. I find that the model is systematically over-predicting the “same" speaker.

In this section, I have detailed two variations of modeling a CNN for a human speaker recognition task using VoxCeleb 2.0. The principal difference between the two modeling attempts is that the model, CNN 1, uses MFCCs extracted from 128 ms windows, and the model, CNN 2, uses MFCCs extracted from 30 ms windows. I find that the CNN using the data extracted at a 128 ms window is easier to train while avoiding overfitting and achieves greater than 90% accuracy.

Why should the reader care that a 128 ms window is used? Because an MFCC created on a 128 ms window of human speech is nonsense. The literature on human speech declares that speech signals are quasi-stationary for windows between 20 and 35 ms. Therefore, using MFCCs for a 128 ms window is at odds with the literature. Why then I am achieving substantially higher accuracy for speaker recognition using an “invalid" MFCC? What is the MFCC at a 128 ms window capturing? It cannot be the vocal system of the speaker as defined in my thesis section on the cepstrum detailed in Section 3 because that vocal system is convolved with the excitation signal for a quasi-stationary piece of the speech signal. This is a question for future research.

5.2 Modeling Speaker Recognition using a CNN with 12 MFCCs

In this section, I reduce the number of MFCCs to 12 coefficients used in feature extraction from the same utterance segment size of 3 seconds. Using 12 coefficients was common until computing capabilities improved over the last decade. From this point forward, I use the 30 ms window to extract the MFCCs, so my CNN with 12 coefficients is a similar manner to CNN 2 presented in the last section except for the number of outputs from the convolutional layers. In CNN 2, the first fully connected layer accepted 9600 outputs, and in this CNN, which I call CNN 3, the first fully connected layer accepts 3200 outputs ($\frac{1}{3} \times 9600$). In the code below, Listing 3, I present the CNN class with 12 MFCCs which I refer to as CNN 3.

Manually tuning the hyperparameters, e.g., learning rate, dropout, and batch size, for CNN 3

was not as difficult as CNN 2. I quickly found hyperparameters for the CNN which achieved performance equivalent to CNN 3 with 40 MFCCs. In Figure 5.3, I present the training and validation loss curves for the highest-performing version of CNN 3.

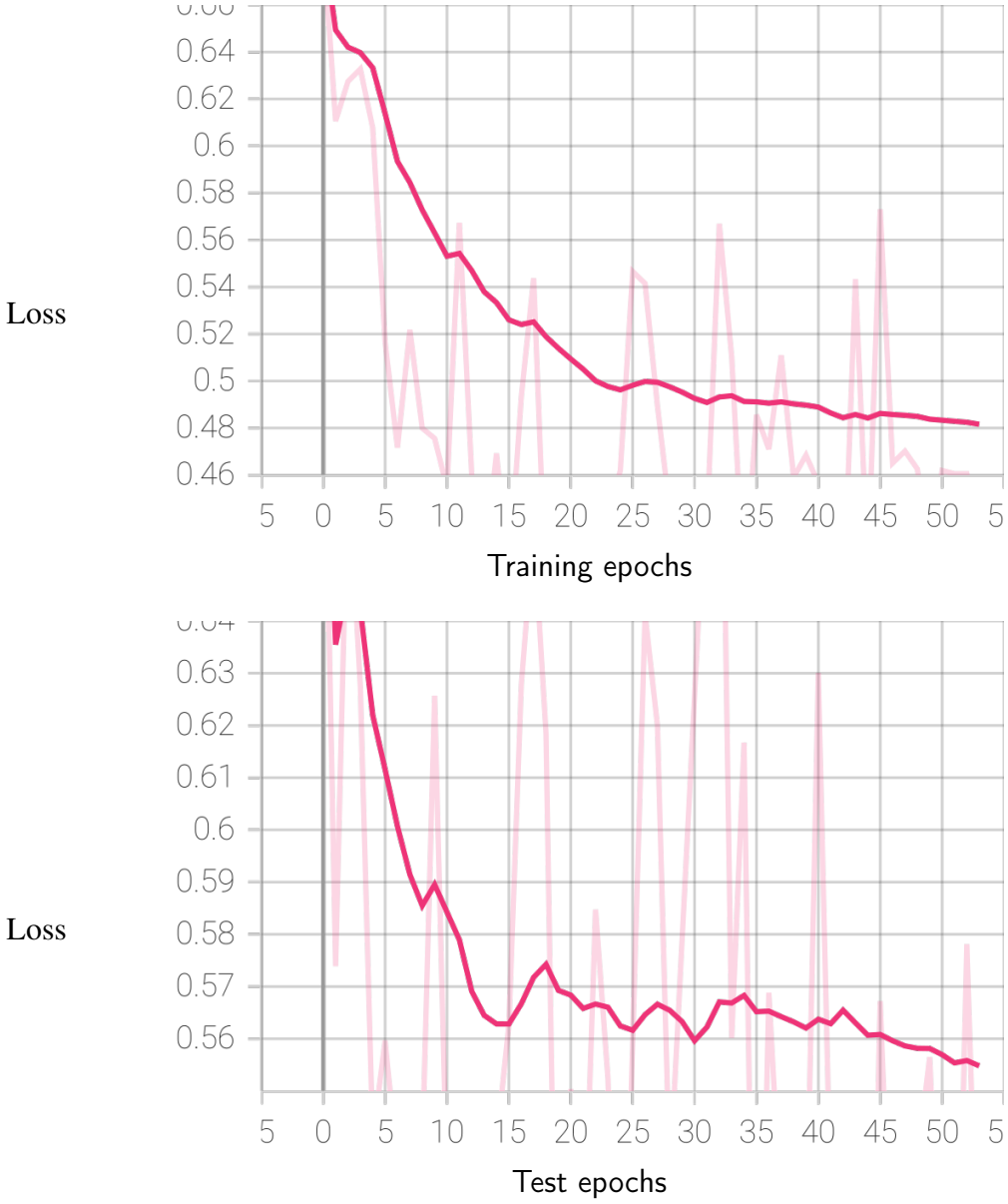

Figure 5.3. CNN 3 with SGD Optimizer, learning rate = 0.3, scheduler set at 6, 8, 10, 12, 14 epochs for momentum = 0.9. The dark line is the smoothed (factor = 1.0) loss curve and the light line is the non-smoothed (factor = 0.0) loss curve.

In Table 5.2, I outline the hyperparameters for CNN 2 and CNN 3 for comparison. For CNN 3, I was able to start with a larger learning rate of 0.3 without causing major fluctuations in the validation loss. I quickly applied momentum ($\gamma = 0.1$) every two epochs after epoch 6 to control overfitting.

As displayed in Figure 5.3, I limited the number of epochs over which training occurred for CNN 3 because the loss curve on the validation data came close to its minimum after only 13 epochs. For reference, in CNN2 shown in Figure 5.2, the validation loss curve continued to fall until the 60th epoch.

	CNN 2	CNN 3
MFCCs		
Number of FFT	480	480
Number of seconds of audio	3	3
Window length (ms)	30	30
Hop length (ms)	15	15
Number of MFCCs	40	12
Number of Mel filters	128	128
CNN		
Training size	50,000	50,000
Test size	10,000	10,000
Number of epochs	70	54
Architecture	see Listing 2	see Listing 3
Convolutional layers	4	3
Linear layers	5	4
Dropout	Yes, 0.3	Yes, 0.15
Pooling	Yes	Yes
Non-linearity function	ReLu	ReLu
Learning rate	0.03	0.3
Scheduled step, ($\gamma = 0.1$)	35, 40, 45	6, 8, 10, 12, 14
Test performance, batch = 64		
True positive (# in batch)	29.6	29.2
True negative (# in batch)	15.6	15.3
False positive (# in batch)	16.5	15.8
False negative (# in batch)	2.4	2.8
Accuracy	0.71	0.70
F1 score	0.76	0.75
Loss	0.53	0.55
Loss graphic	see Figure 5.2	see Figure 5.3

Table 5.2. Feature extraction, modeling, and performance details for two CNNs using 40 MFCCs or 12 MFCCs

In the bottom rows of Table 5.2, the performance measures are presented for CNN 2 (40 MFCCs) and CNN 3 (12 MFCCs). The performances of the two CNNs are very similar. CNN 3 mislabeled positives at a lower rate than CNN 2, however, when each CNN mislabels, it is much more likely to assign a speaker pair as “same" when the true label for the speaker pair was “different." The resulting performance for CNN 3 is evidence that coefficients 13 to 40 contained little helpful information as compared with coefficients 1 to 12. Additionally, hyperparameters for the model with 12 coefficients (CNN 3) was easier than the model with 40 coefficients (CNN 2).

5.3 Modeling Speaker Recognition using an MLP

Following, the guidance provided by Rudin [3] and described in detail in Chapter 2, in this section, I replace the CNN with a simpler model, the MLP, to examine relative performance. Like the early sections in this chapter, I present the MLP class code first in Listing 4 in the Appendix. The MLP I designed is three fully connected layers with a ReLU activation function between each one. With three layers, the MLP was able to correctly make some predictions. With fewer layers, the MLP was not making any correct predictions. The initial inputs total 4824 = 12 MFCC $\times$ (201 frames + 201 frames).

In Figure 5.4, I present the loss curves for the training and validation sets using the MLP. The details of the model, hyperparameters, and training decisions are presented in Table 5.3 for CNN 3 and the MLP. The loss curve from the validation set in Figure 5.4 shows that the useful learning taking place during training was inconsistent for correct predictions on the validation set.

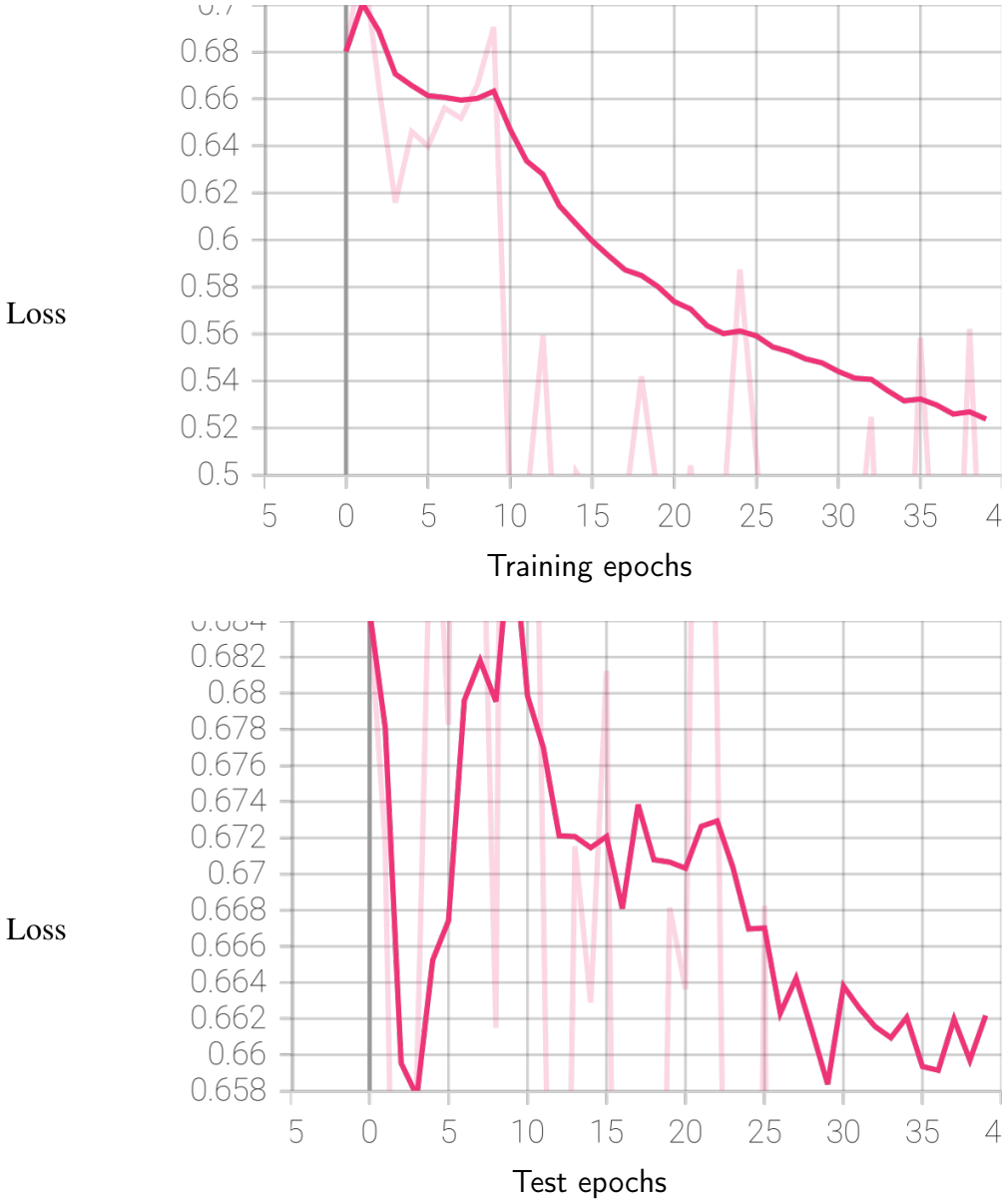

Figure 5.4. MLP with SGD Optimizer, learning rate = 0.003, scheduler set at 10, 20, 20 epochs for momentum = 0.9. The dark line is the smoothed (factor = 1.0) loss curve and the light line is the non-smoothed (factor = 0.0) loss curve.

Training the MLP was difficult in that the model quickly overfit the training data. To achieve a performance accuracy of 0.61, I had to enact a small learning rate of 0.003 which I reduced at regular intervals during the 50 epochs of training. Like CNN 1, 2, and 3, presented earlier in this chapter, the MLP also labels false positives at a greater rate than false negatives. However, the incremental change in mislabeling between CNN 3 and MLP more or less equally affected the false positive and false negative rates.

	CNN 3	MLP
MFCCs		
Number of FFT	480	480
Number of seconds of audio	3	3
Window length (ms)	30	30
Hop length (ms)	15	15
Number of MFCCs	12	12
Number of Mel filters	128	128
CNN		
Training size	50,000	50,000
Test size	10,000	10,000
Number of epochs	54	50
Architecture	see Listing 2	see Listing 4
Convolutional layers	3	0
Linear layers	5	4
Dropout	Yes, 0.3	NA
Pooling	Yes	NA
Non-linearity function	ReLu	ReLu
Learning rate	0.03	0.003
Scheduled step, ($\gamma = 0.1$)	6, 8, 10, 12, 14	10, 20, 30
Test performance, batch = 64		
True positive (# in batch)	29.2	25.7
True negative (# in batch)	15.3	13.3
False positive (# in batch)	15.8	18.8
False negative (# in batch)	2.8	6.3
Accuracy	0.70	0.61
F1 score	0.75	0.67
Loss	0.55	0.69
Loss graphic	see Figure 5.3	see Figure 5.4

Table 5.3. Feature extraction, modeling, and performance details for CNNs and MLP using 12 MFCCs

5.4 Applying a Logistic Regression to the VoxCeleb Speaker Recognition Task

I use PyTorch's implementation of the logistic regression. By using PyTorch for logistic regression, I can train the model on batches from the 50,000 training and 10,000 validation observations like the other models I use. In Listing 5 in the Appendix, I present the PyTorch custom class for logistic regression I wrote for this speaker recognition task. Like the MLP class presented in Listing 4 in the Appendix, the inputs total 4824.

To try to optimize performance, I manipulated the hyperparameters: learning rate and the number of inputs for the second fully connected layer. I found that a high learning rate (> 0.05) resulted in an exploding loss for the training set. Therefore, I constrained the learning rate to 0.05 to enable the model to continue running. In Figure 5.5, the loss curves are flat (blue curves) and show that the logistic regression is unable to learn the patterns necessary for the speaker recognition task. I include the MLP curve from Figure 5.4 in pink for reference. The test set accuracy using this logistic regression model is 0.50, f-score of 0.66, with almost all observations being labeled as the "same" speaker.

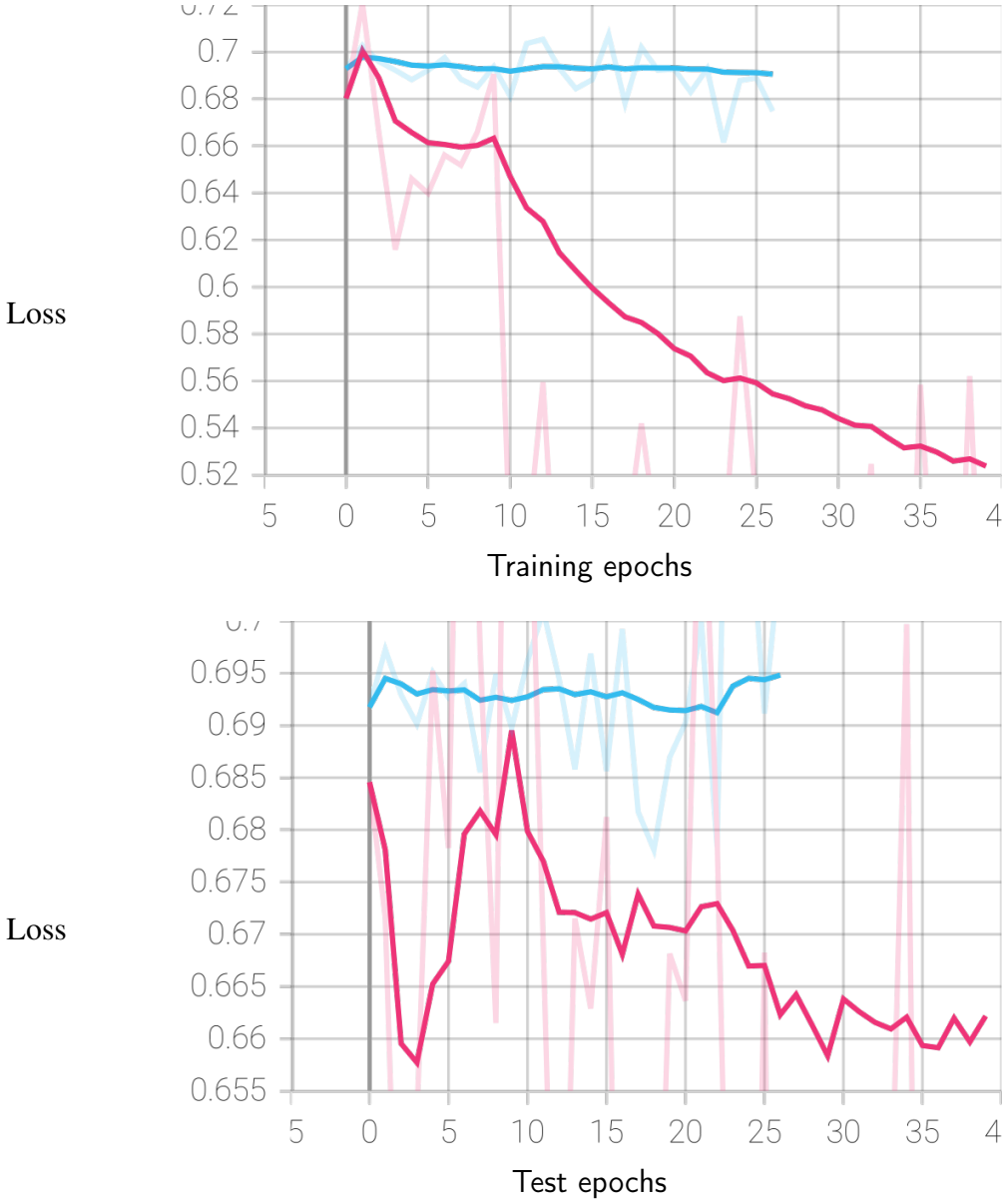

Figure 5.5. LR in Blue with SGD Optimizer, learning rate = 0.05, Pink curves are from the MLP model details in Section 5.3. The dark line is the smoothed (factor = 1.0) loss curve and the light line is the non-smoothed (factor = 0.0) loss curve.

CHAPTER 6: Conclusion

In this chapter, I provide an overview of the implications for DOD and concluding remarks. An overview of the results was presented earlier in the preface to Chapter 5.

6.1 Implications for Defense

This thesis focuses on a speaker recognition task in audio analysis, but the implication of interpretability uncovered in this research is important for applications of AI more broadly. For DOD, speaker recognition may become more important as technology continues to advance, for example, speaker-identified transcription is an important task.

Future approaches to AI in DOD need to be trustworthy which means that interpretability needs to be developed. Since DOD applications of AI are of high consequence and likewise scrutinized, DOD cannot rely on private sector AI development to meet DOD needs for interpretability [3], [6]. While high-consequence tasks do exist in private sector AI development, like self-driving cars, much of the private sector efforts in AI development focus on performance and not on interpretability [9].

My research explored the use of simpler models and fewer important features in prediction. My results show that only the most complex model performed well. The CNN model achieved an acceptable level of performance in speaker recognition tasks. Speaker recognition like other “hard tasks" will encompass a large portion of AI development going forward. DOD needs to be ready to make use of promising yet complex AI applications. Regarding feature extraction, fewer MFCCs (12 coefficients) performed as well as many MFCCs (40 coefficients.)

During this research, I discovered that interpretable CNNs and other complex modeling practices are at the frontier of AI research. Methods like *disentangled* CNN-models, among a few others, allow a practitioner and user to understand what features a CNN uses in its decision-making framework, which is only just being developed. DOD should consider spearheading interpretable AI development for the public good.

6.2 Concluding Remarks

People think that AI is new but quality AI relies on decades of research. In this thesis, I drew on papers about human speech recognition from the 1950s and cutting-edge research on interpretability from this year, and I tried to unite them. It was difficult. As an economist, I come from a field where measurement is essential. Measurement of features and keeping track of the features' use in the modeling task is important for interpretability. In speaker recognition, I found little care for the conceptual integrity of human speech and did find the common criticism of AI which is a sole focus on accuracy.

In my own small research project on speaker recognition for this thesis, I experienced first-hand the inverse relationship between accuracy and feature integrity. When modeling a CNN, a misspecified MFCC yielded impressive accuracy. But how? I would like to know what exactly 40 MFCCs are capturing about a human speaker over a 128 ms window. From the theoretical literature on the cepstrum and MFCC, the answer should be "nothing." But the impressive accuracy means that "something" meaningful about a speaker's identity was captured. It's a conundrum - and one without an explanation since CNN defies interpretation.

My conclusion about speaker recognition tasks is that feature extraction is fundamental. More specifically, uniting feature extraction with the 70-year-long literature on human speech is the most important. Human speech is the most studied form of audio because it is the most important to us! Then once the features have a theoretically sound basis then tune the model for performance.

APPENDIX: PyTorch Neural Network Code

Listing 1: CNN 1 with 40 MFCC and 93 frames using 128 ms window

```
class CNN(torch.nn.Module):

    def __init__(self):
        super(CNN, self).__init__()

        self.layer1 = torch.nn.Sequential(
            torch.nn.Conv2d(1, 32, kernel_size=2, stride=1, padding=1),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(kernel_size=2, stride=2),
            )

        self.layer2 = torch.nn.Sequential(
            torch.nn.Conv2d(32, 64, kernel_size=2, stride=1, padding=1),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(kernel_size=2, stride=2),
            )

        self.layer3 = torch.nn.Sequential(
            torch.nn.Conv2d(64, 128, kernel_size=2, stride=1, padding=1),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(kernel_size=2, stride=2),
            )

        self.layer4 = torch.nn.Sequential(
            torch.nn.Conv2d(128, 256, kernel_size=2, stride=1, padding=1),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(kernel_size=2, stride=2),
            )

        # L4 FC 12x3x256 inputs -> 9216 outputs
        self.hidden1 = nn.Linear(256*12*3, 2000)
        kaiming_uniform_(self.hidden1.weight, nonlinearity='relu')
        self.act1 = nn.ReLU()
        # Second hidden layer
```

```
        self.hidden2 = nn.Linear(2000, 500)
        kaiming_uniform_(self.hidden2.weight, nonlinearity='relu')
        self.act2 = nn.ReLU()
        # Second hidden layer
        self.hidden3 = nn.Linear(500, 75)
        kaiming_uniform_(self.hidden3.weight, nonlinearity='relu')
        self.act3 = nn.ReLU()
        # Third hidden layer
        self.hidden4 = nn.Linear(75,1)

    def forward(self, t1, t2):
        X = torch.cat((t1, t2), dim=1)
        X = X[None, :]
        X = X.permute(1,0,2,3) # rearrange tensor
        out = self.layer1(X)
        out = self.layer2(out)
        out = self.layer3(out)
        out = self.layer4(out)
        out = out.view(out.size(0), -1)   # Flatten for FC
        #Input to the first hidden layer
        X = self.hidden1(out)
        X = self.act1(X)
        # Second hidden layer
        X = self.hidden2(X)
        X = self.act2(X)
        # Third hidden layer
        X = self.hidden3(X)
        X = self.act3(X)
        # Fourth hidden layer
        X = self.hidden4(X)
        return X
```

Listing 2: CNN 2 with 40 MFCC and 480 frames using 30 ms window

```
class CNN(torch.nn.Module):

    def __init__(self):
        super(CNN, self).__init__()
```

```
self.layer1 = torch.nn.Sequential(
    torch.nn.Conv2d(1, 32, kernel_size=2, stride=1, padding=1),
    torch.nn.ReLU(),
    torch.nn.MaxPool2d(kernel_size=2, stride=2),
    torch.nn.Dropout(0.3)
    )

self.layer2 = torch.nn.Sequential(
    torch.nn.Conv2d(32, 64, kernel_size=2, stride=1, padding=1),
    torch.nn.ReLU(),
    torch.nn.MaxPool2d(kernel_size=2, stride=2),
    torch.nn.Dropout(0.3)
    )

self.layer3 = torch.nn.Sequential(
    torch.nn.Conv2d(64, 128, kernel_size=2, stride=2, padding=1),
    torch.nn.ReLU(),
    torch.nn.MaxPool2d(kernel_size=2, stride=2),
    torch.nn.Dropout(0.3)
    )

# L4 FC 25x3x128 inputs -> 9600 outputs
self.hidden1 = nn.Linear(25*3*128, 5000)
kaiming_uniform_(self.hidden1.weight, nonlinearity='relu')
self.act1 = nn.ReLU()
# Second hidden layer
self.hidden2 = nn.Linear(5000, 1000)
kaiming_uniform_(self.hidden2.weight, nonlinearity='relu')
self.act2 = nn.ReLU()
# Second hidden layer
self.hidden3 = nn.Linear(1000, 250)
kaiming_uniform_(self.hidden3.weight, nonlinearity='relu')
self.act3 = nn.ReLU()
# Third hidden layer
self.hidden4 = nn.Linear(250, 75)
kaiming_uniform_(self.hidden4.weight, nonlinearity='relu')
self.act4 = nn.ReLU()
self.hidden5 = nn.Linear(75,1)
```

```
    def forward(self, t1, t2):
        X = torch.cat((t1, t2), dim=1)
        X = X[None, :]
        X = X.permute(1,0,2, 3) # rearrange tensor
        out = self.layer1(X)
        out = self.layer2(out)
        out = self.layer3(out)
        out = out.view(out.size(0), -1)   # Flatten for FC
        #Input to the first hidden layer
        X = self.hidden1(out)
        X = self.act1(X)
        # Second hidden layer
        X = self.hidden2(X)
        X = self.act2(X)
        # Third hidden layer
        X = self.hidden3(X)
        X = self.act3(X)
        # Fourth hidden layer
        X = self.hidden4(X)
        X = self.act4(X)
        # Fourth hidden layer
        X = self.hidden5(X)
        return X
```

Listing 3: CNN 3 with 12 MFCC and 480 frames using 30 ms window

```
class CNN(torch.nn.Module):

    class CNN12(torch.nn.Module):

    def __init__(self):
        super(CNN12, self).__init__()

        self.layer1 = torch.nn.Sequential(
            torch.nn.Conv2d(1, 32, kernel_size=2, stride=1, padding=1),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(kernel_size=2, stride=2),
            # torch.nn.Dropout(0.15)
            )
```

```
        self.layer2 = torch.nn.Sequential(
            torch.nn.Conv2d(32, 64, kernel_size=2, stride=1, padding=1),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(kernel_size=2, stride=2),
            # torch.nn.Dropout(0.15)
            )

        self.layer3 = torch.nn.Sequential(
            torch.nn.Conv2d(64, 128, kernel_size=2, stride=2, padding=1),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(kernel_size=2, stride=2),
            # torch.nn.Dropout(0.15)
            )

        # L4 FC 25x1x128 inputs -> 3200 outputs
        self.hidden1 = nn.Linear(25*1*128, 1000)
        kaiming_uniform_(self.hidden1.weight, nonlinearity='relu')
        self.act1 = nn.ReLU()
        # Second hidden layer
        self.hidden2 = nn.Linear(1000, 250)
        kaiming_uniform_(self.hidden2.weight, nonlinearity='relu')
        self.act2 = nn.ReLU()
        # Third hidden layer
        self.hidden3 = nn.Linear(250, 75)
        kaiming_uniform_(self.hidden3.weight, nonlinearity='relu')
        self.act3 = nn.ReLU()
        self.hidden4 = nn.Linear(75,1)

    def forward(self, t1, t2):
        X = torch.cat((t1, t2), dim=1)
        X = X[None, :]
        X = X.permute(1,0,2, 3) # (batch size= 64, input_channels=1, signal_length=5)
        out = self.layer1(X)
        out = self.layer2(out)
        out = self.layer3(out)
        out = out.view(out.size(0), -1)   # Flatten them for FC
        #Input to the first hidden layer
        X = self.hidden1(out)
```

```
        X = self.act1(X)
        # Second hidden layer
        X = self.hidden2(X)
        X = self.act2(X)
        # Third hidden layer
        X = self.hidden3(X)
        X = self.act3(X)
        # Fourth hidden layer
        X = self.hidden4(X)
        return X
```

Listing 4: MLP with 12 MFCC and 480 frames using 30 ms window

```
class MLP(nn.Module):

    def __init__(self):
        super(MLP, self).__init__()
        self.hidden1 = nn.Linear(4824, 1000)
        kaiming_uniform_(self.hidden1.weight, nonlinearity='relu')
        self.act1 = nn.ReLU()
        # Second hidden layer
        self.hidden2 = nn.Linear(1000, 250)
        kaiming_uniform_(self.hidden2.weight, nonlinearity='relu')
        self.act2 = nn.ReLU()
        # Second hidden layer
        self.hidden3 = nn.Linear(250, 75)
        kaiming_uniform_(self.hidden3.weight, nonlinearity='relu')
        self.act3 = nn.ReLU()
        # Third hidden layer
        self.hidden4 = nn.Linear(75,1)


    def forward(self, t1, t2):
        X = torch.cat((t1, t2), dim=1)
        X = torch.reshape(X, (t1.shape[0], 2*t1.shape[1]*t1.shape[2]))
        #Input to the first hidden layer
        X = self.hidden1(X)
        X = self.act1(X)
        # Second hidden layer
```

```
        X = self.hidden2(X)
        X = self.act2(X)
        # Third hidden layer
        X = self.hidden3(X)
        X = self.act3(X)
        # Fourth hidden layer
        X = self.hidden4(X)
        return X
```

Listing 5: Logistic Regression with 12 MFCC and 480 frames using 30 ms window

```
class LR(nn.Module):

    def __init__(self):
        super(LR, self).__init__()
        self.layer1=nn.Linear(4824,100)
        self.layer2=nn.Linear(100,1)

    def forward(self, t1, t2):
        X = torch.cat((t1, t2), dim=1)
        X = torch.reshape(X, (t1.shape[0], 2*t1.shape[1]*t1.shape[2]))
        X=self.layer1(X)
        X=self.layer2(X)
        return X
```

THIS PAGE INTENTIONALLY LEFT BLANK

List of References

[1] Jaworski, "Numbers and facts you need to know about audio content in 2021," 2022, business 2 Community. Accessed September 26, 2022, https://www.business2community.com/digital-marketing/numbers-and-facts-you-need-to-know-about-audio-content-in-2021-02398919.

[2] D. Tarraf, W. Shelton, E. Parker, B. Alkire, D. Gehlhaus, J. Grana, A. Levedahl, J. Léveillé, J. Mondschein, J. Ryseff, A. Wyne, D. Elinoff, E. Geist, B. Harris, E. Hui, C. Kenney, S. Newberry, C. Sachs, P. Schirmer, D. Schlang, V. Smith, A. Tingstad, P. Vedula, and K. Warren, *The Department of Defense's Posture for Artificial Intelligence: Assessment and Recommendations for Improvement*. Santa Monica, CA: RAND Corporation, 2021.

[3] C. Rudin, C. Chen, Z. Chen, H. Huang, L. Semenova, and C. Zhong, "Interpretable machine learning: Fundamental principles and 10 grand challenges," 2021. Available: https://arxiv.org/abs/2103.11251

[4] A. Nagrani, J. S. Chung, W. Xie, and A. Zisserman, "Voxceleb: Large-scale speaker verification in the wild," *Computer Speech & Language*, vol. 60, p. 101027, 2020. Available: https://www.sciencedirect.com/science/article/pii/S0885230819302712

[5] L. Whitney, "How to train Amazon's Alexa to recognize your voice," 2022, pC Mag website. Accessed 10/14/2022, https://www.pcmag.com/how-to/train-amazons-alexa-to-recognize-your-voice.

[6] B. Kim and F. Doshi-Velez, "Machine learning techniques for accountability," *AI Magazine*, no. Spring, 2021.

[7] C. Rudin, "Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead," *Nature Machine Intelligence*, vol. 1, 2019.

[8] B. Kim, M. Wattenberg, J. Gilmer, C. Cai, J. Wexler, F. Viegas, and R. sayres, "Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (TCAV)," in *Proceedings of the 35th International Conference on Machine Learning* (Proceedings of Machine Learning Research), J. Dy and A. Krause, Eds. PMLR, 10–15 Jul 2018, vol. 80, pp. 2668–2677. Available: https://proceedings.mlr.press/v80/kim18d.html

[9] DARPA, "Explainable artificial intelligence (XAI)," *Defense Advanced Research Projects Agency, Broad Agency Announcement*, 2016.

[10] D. Broniatowski, "Psychological foundations of explainability and interpretability in artificial intelligence," National Institute of Standards and Technology (NIST), Washington, DC, National Institute of Standards and Technology Interagency or Internal Report 8367, 2021.

[11] S. Bhatt, A. Jain, and A. Dev, "Acoustic modeling in speech recognition: A systematic review," *International Journal of Advanced Computer Science and Applications*, vol. 11, no. 4, 2020. Available: http://dx.doi.org/10.14569/IJACSA.2020.0110455

[12] F. Doshi-Velez and B. Kim, "Towards a rigorous science of interpretable machine learning," 2017. Available: https://arxiv.org/abs/1702.08608

[13] E. Babaee, Nor Badrul Anuar, A. Wahab, S. Shamshirband, and A. Chronopoulos, "An overview of audio event detection methods from feature extraction to classification," *Applied Artificial Intelligence*, vol. 31, no. 9-10, pp. 661–714, 2017. Available: https://doi.org/10.1080/08839514.2018.1430469

[14] R. F. Lyon, "Machine hearing: An emerging field [exploratory dsp]," *IEEE Signal Processing Magazine*, vol. 27, no. 5, pp. 131–139, 2010.

[15] F. Alias, J. C. Socoro, and X. Sevillano, "A review of physical and perceptual feature extraction techniques for speech, music and environmental sounds," *Applied Sciences*, vol. 6, no. 5, 2016. Available: https://www.mdpi.com/2076-3417/6/5/143

[16] D. Gerhard, "Audio signal classification: History and current techniques," Department of Computer Science, University of Regina, Regina, SK, Canada, techreport TR-CS 2003-07, 2003.

[17] G. Sharma, K. Umapathy, and S. Krishnan, "Trends in audio signal feature extraction methods," *Applied Acoustics*, vol. 158, p. 107020, 2020. Available: https://www.sciencedirect.com/science/article/pii/S0003682X19308795

[18] E. Shriberg, *Higher-Level Features in Speaker Recognition*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 241–259. Available: https://doi.org/10.1007/978-3-540-74200-5_14

[19] Z. Wu, J. Yamagishi, T. Kinnunen, C. Hanilci, M. Sahidullah, A. Sizov, N. Evans, M. Todisco, and H. Delgado, "ASVspoof: The automatic speaker verification spoofing and countermeasures challenge," *IEEE Journal of Selected Topics in Signal Processing*, vol. 11, no. 4, pp. 588–604, 2017.

[20] S. Stevens, J. Volkmann, and E. Newman, "A scale for the measurement of the psychological magnitude of pitch," *Journal of the Acoustic Society of America*, vol. 8, pp. 185–190, 1937.

[21] R. Warren, *Auditory Perception: An Analysis and Synthesis*, 3rd ed. Cambridge University Press, Cambridge, 2008.

[22] A. Bachem, "Chroma fixation at the ends of the musical frequency scale," *Journal of the Acoustical Society of America*, vol. 20, pp. 704–705, 1948.

[23] S. Davis and P. Mermelstein, "Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 28, no. 4, pp. 357–366, 1980.

[24] C. Fant, "Acoustic description and classification of phonetic units," *Ericsson Technics*, vol. 1, 1959.

[25] W. Ward, "Musical perception," in *Foundations of Modern Auditory Theory*, J. Tobias, Ed. Academic Press, London, 1970.

[26] W. Koenig, "A new frequency scale for acoustic measurements," Bell Telephone Laboratory Record, Tech. Rep., 1949.

[27] J. Deller, J. Hansen, and J. Proakis, *Discrete-Time Processing of Speech Signals*. Wiley-IEEE Press, Piscataway, NJ, 2000.

[28] D. Greenwood, "The mel scale's disqualifying bias and a consistency of pitch-difference equisections in 1956 with equal cochlear distances and equal frequency ratios," *Hearing Research*, vol. 103, pp. 199–224, 1997.

[29] D. Greenwood, "List-archive exchange to abandon mel scale," Accessed on 10/10/2022 at http://www.auditory.org/postings/2009/53.html,, 2009.

[30] S. Sarangi, M. Sahidullah, and G. Saha, "Optimization of data-driven filterbank for automatic speaker verification," *Digital Signal Processing*, vol. 104, p. 102795, 2020. Available: https://www.sciencedirect.com/science/article/pii/S1051200420301408

[31] D. Lewis, "Pitch-scales," *Journal of the Acoustical Society of America*, vol. 14, p. 127, 1942.

[32] S. Stevens, "On the psychophysical law," *Psychological Review*, vol. 64, pp. 153–181, 1957.

[33] F. Moore, "An introduction to the mathematics of digital signal processing: Part ii: Sampling, transforms, and digital filtering," *Computer Music Journal*, vol. 2, pp. 38–60, 1978.

[34] B. Bogert, M. Healy, and J. Tukey, "The quefrency analysis of time series for echoes: Cepstrum, pseudo-autocovariance, cross-cepstrum and saphe cracking," *Proceedings of the Symposium on Time Series Analysis*, vol. 15, pp. 209–243, 1963.

[35] A. Noll, "Cepstrum pitch determination," *The Journal of the Acoustical Society of America*, vol. 41, pp. 293–309, 1967.

[36] S. Molau, M. Pitz, R. Schluter, and H. Ney, "Computing mel-frequency cepstral coefficients on the power spectrum," in *2001 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings (Cat. No.01CH37221)*, 2001, vol. 1, pp. 73–76 vol.1.

[37] A. Sable, "Introduction to audio analysis and precessing," 2021, paperspace Blog. Accessed November 9, 2022, https://blog.paperspace.com/introduction-to-audio-analysis-and-synthesis/.

[38] J. Smith, "Cepstral smoothing," 2020, "Cross Synthesis Using Cepstral Smoothing or Linear Prediction for Spectral Envelopes", (From Lecture Overheads, Music 421). Accessed November 14, 2022, https://ccrma.stanford.edu/~jos/SpecEnv/Cepstral_Smoothing.html.

[39] S. Sigurdsson, K. Petersen, and T. Lehn-Schioler, "Mel frequency cepstral coefficients: An evaluation of robustness of mp3 encoded music," in *Proceedings of the Seventh International Conference on Music Information Retrieval (ISMIR)*, 2006.

[40] K. Paliwal, "Decorrelated and liftered filter-bank energies for robustspeech recognition," in *6th European Conference on Speech Communication and Technology (EUROSPEECH '99*, Budapest, Hungary, September 5-9,, 1999.

[41] B. Alsaify, H. Arja, B. Maayah, and M. Al-Taweel, "A dataset for voice-based human identity recognition," *Data in Brief*, vol. 42, p. 108070, 2022.

[42] H. Lei and E. Lopez-Gonzalo, "Mel, linear, and antimel frequency cepstral coefficients in broad phonetic regions for telephone speaker recognition," in *Proceedings of the 10th International Conference of the International Speech Communication Association (Interspeech 2009)*, Brighton, United Kingdom, 2009, pp. pp. 2323–2326.

[43] VoxCeleb, "Vox celeb: a large scale audio-visual dataset of human speech," 2022, voxCeleb. Accessed October 25, 202, https://mm.kaist.ac.kr/datasets/voxceleb/index.html.

Initial Distribution List

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California