THE COOPER UNION
ALBERT NERKEN SCHOOL OF ENGINEERING

Deep Learning Pipeline for Detection of Mild Cognitive Impairment from Unstructured Long Form Clinical Audio

Author:
Theo Jaquenoud

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Engineering

April 2022

Advisors:
Professor Neveen Shlayan
Professor Sam Keene

0.1 Acknowledgments

Firstly, I would like to thank my co-advisors Professor Sam Keene and Professor Neveen Shlayan, without whom this thesis would not have been possible. They have been instrumental not just for this research, but for my growth as an engineer and a young professional. They are also each responsible for introducing me to the worlds of machine learning and biomedical engineering, which together form the backbone of this project.

I would also like to thank my co-advisor Professor Gaurav Pandey for introducing me to this project, giving me the opportunity to participate in research at Mount Sinai, and guiding me throughout my research. He trusted me and gave me a lot of freedom, all the while providing important insight, particularly on how to navigate the biomedical world as an engineer. I am also grateful for Doctor Alex Federman and project manager Chrissy Dayeon Cho's help with integrating me into the team.

Lastly, I would like to thank my family, especially my fiancée and my parents, for supporting me throughout my education and research. I would never have made it this far without your love and care!

Abstract

Mild Cognitive Impairment (MCI) is a condition that negatively effects some older adults' memory, use of language, and judgement. While there are cognitive assessments to diagnose MCI, screening competes with the many other tasks which clinicians have to perform, and is rarely done in a primary care setting. As a result, MCI remains difficult to identify and tends to be underdiagnosed. The research in this thesis was conducted in partnership with the Icahn School of Medicine at Mount Sinai, where researchers assembled a dataset of 800 recordings of primary care visits, paired with MCI diagnoses. Using this dataset, I present a deep learning pipeline for detecting MCI from long form clinical audio recordings. This pipeline uses existing neural network models alongside domain knowledge to correctly isolate the patient's speech in 98% of sampled recordings. Then, it uses short-time Fourier transforms and the Mel scale to convert the resulting audio into a more suitable format for neural network classification. The spectrograms are then split into overlapping regular segments, with each segment inheriting the label of the parent audio. The resulting dataset is used to train convolutional neural networks from scratch, as well as to fine-tune models pretrained on larger labeled datasets. These models convincingly exceed a baseline performance at the segment level, with an AUC ranging from 0.57 to 0.65 based on the model architecture and training regimen. A further exploration of model outputs and patient-level performance is also provided.

Table of Contents

List of Figures

List of Tables

Chapter 1: Introduction

1.1 Problem Statement

A rapidly aging population in the United States is increasing the number of people with mild cognitive impairment (MCI), a condition characterized by a loss in cognitive function in elderly people that is often followed by dementia or even Alzheimer's disease. As screening and diagnostic tools have seen little advances in recent decades, most remain undiagnosed. Moreover, clinicians, researchers, and health systems have inefficient methods to identify cases. [1]

Emerging strategies following recent advances in machine learning (ML), particularly deep learning (DL), have the potential to supplant current methods of identifying MCI. A research team at Mount Sinai is exploring a novel multi-modal analysis of patient's electronic health records (EHR) and audio recorded during clinical visits with their primary care physician in order to screen for or even diagnose MCI. They have collected these recordings from about 800 patients, and are now pursuing four tracks of analysis, that will eventually be merged to produce one ensemble model, see figure 4.1.

The first track is the EHR, analysed by classical ML as well as more modern natural language processing (NLP). The second track is to analyse transcriptions from the recordings of patient-physician conversations with natural language processing.

The third track is to analyse the audio itself, on one hand by extracting standard feature sets and using classical ML methods, and on the other hand using modern DL end-to-end for audio classification. The later is the focus of this paper: to build a deep learning pipeline to detect MCI from within these unstructured long form audio recordings of clinical visits.

1.2 Contribution

The contributions of this research are two-fold. Firstly, a preprocessing pipeline is developed to identify and extract the audio corresponding to the speech of the patient, and separate the result into regular segments to create a labeled dataset suitable for DL. Secondly, a variety of neural networks are tested to identify cognitive impairment from within these processed recordings. These include convolutional neural networks trained from scratch using available data, as well as transfer learning using models pretrained on larger audio datasets. Finally, the outputs of the models are analysed, and methods for aggregating the results on a patient-level are tested.

1.3 Overview

The remainder of this paper is organized as follows. In chapter 2, background is provided on the subjects of MCI, audio processing, and deep learning. In chapter 3, prior works are reviewed, firstly on clinical application of speech analysis, using both traditional feature-based approaches and DL, secondly on more modern examples of audio classification models in non-clinical settings. In chapter 4, the methodology for the research is described. First, an overview of the research project is provided, followed by a description of the dataset. Next, the preprocessing is described in three steps: the speaker diarization and patient identification, the Mel spectrogram generation, and the stratified train-validation-test splits of the dataset. In chapter 5, the results

are presented for both the models and the patient-level aggregation of the scores, and are then discussed within the greater context of the research project. Lastly, chapter 6 concludes the thesis by summarizing the contributions and suggesting future avenues to explore.

Chapter 2: Background

2.1 Mild Cognitive Impairment

Mild Cognitive Impairment (MCI) is a condition that negatively effects some older adults' memory, use of language, and judgement. While some cognitive decline is expected as a result of aging, MCI is characterized as being more serious than normal decline, while not yet being as severe as conditions like dementia [2].

2.1.1 Description and Prevalence of Disease

The brain, much like the rest of the body, changes with age [3]. However, people with MCI experience an accelerated decline in cognitive functions. Symptoms typically include forgetfulness, particularly with names or everyday objects, losing one's train of thought, and being unable to find certain words [3]. MCI may sometimes impair judgement and decision making; it has also been found to be associated with depression, anxiety, and apathy in some patients [4]. Research by the National Institute on Aging estimates that between 10% and 20% of people over the age of 65 with MCI develop dementia over a one-year period [5].

MCI becomes more and more prevalent with age. A recent systematic review [2] reports the prevalence by age group to be 7.6% for ages 55–59, 9.5% for 60–69, 14.6% for 70–79, and 23.6% for 80 and above. The same paper describes how certain groups

appear more likely to develop MCI, these include women, rural inhabitants, those who live alone, and those who have lower levels of education [2].

While there are no known direct causes of MCI, several risk factors beyond just age have been identified, including but not limited to: diabetes, smoking, high blood pressure, high cholesterol, obesity, depression, and lack of physical and mental exercise. Some genetic biomarkers have also been linked with MCI and eventually Alzheimer's disease (AD) [6].

While treatment options are varied and show very little success, if any [7], early diagnosis and careful tracking of the disease's prevention may help slow development, as well as allow patients to adjust their lifestyle and learn to manage their condition [5].

2.1.2 MoCA Test

The Montreal Cognitive Assessment (MoCA) is a brief 30-question test used by physicians to screen patients for MCI. It is designed to test a range of cognitive functions including short term memory, visual and spatial recognition, attention, and language. In about ten minutes, the patient is asked to answer all 30 questions, with 26 points or higher being considered normal. A score below 26 is indicative of MCI with a reported sensitivity of 90% for MCI and 100% for mild AD. [8]

While the MoCA itself has proven successful in identifying MCI, screening competes with the many other tasks which clinicians have to perform, and is rarely done in a primary care setting [9, 10]. As a result, MCI remains difficult to identify and tends to be be underdiagnosed [11, 12], leaving the door open for new screening techniques that can be seamlessly integrated with primary care.

2.2 Audio Processing

2.2.1 Digital Signals Processing

Digital Signals Processing (DSP) refers to any processing that is done on a digital signal, including audio and speech signals. In many cases, a signal may not start off as digital, in which case it must be sampled and quantized.

Digitization

If a signal is not already digital, it must be digitized. The process of transforming a continuous-time signal into a discrete-time signal is called **sampling**. It can be equated to taking measurements of a continuous-time or continuous-space signal at certain intervals. For time-domain signals, as is the case with audio, that time interval will typically be held constant. Thus we can define a **sample rate**, the number of samples recorded in a period of one second, measured in Hertz (Hz).

Since continuous signals can take on an infinite range of values, the measurements must be **quantized**, or mapped to a smaller, finite set of values. Most commonly, these values are represented as words of a fixed length, such as 8-bit, 16-bit, or 24-bit, which are able to represent 256, 65 536, and 16 777 216 discrete levels respectively. The number of bits used for a single sample is also called the bit-depth.

When it comes to recording digital audio, there are a number of encoding formats, but the most straightforward one, which corresponds to the sampling and quantization schema presented above, is pulse-code modulation (PCM). Many lossless audio file storing formats follow some version of PCM, including the .wav file type, which supports a range of sample rates and bit depths, as well as a number of audio channels. Typical high quality audio formats use a sample rate of 44.1 kHz or 48 kHz, and a bit-depth of 16, 24, or 32 bits-per-sample [13].

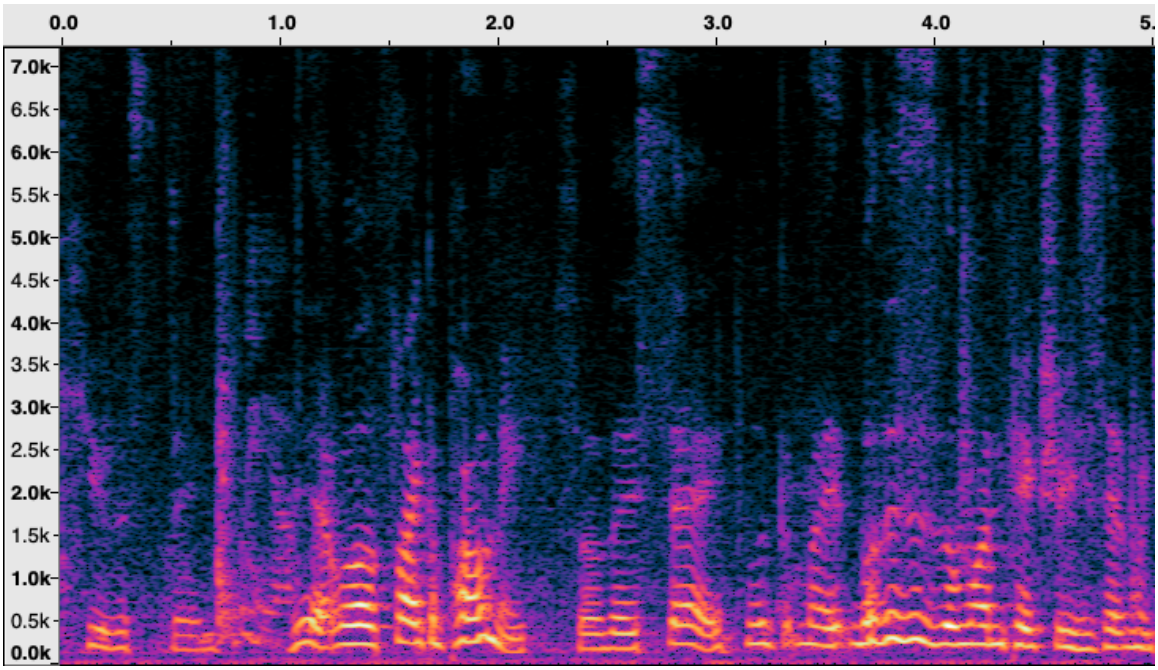

Figure 2.1: Example of a short time Fourier transform of an audio recording of speech. In this spectrogram, frequency is represented linearly in kilohertz on the vertical axis, time is represented in seconds on the horizontal axis, and amplitude or volume is represented in decibels as colors ranging from dark purple to yellow. It is not uncommon for spectrograms to use logarithmic scaling for frequency or amplitude.

Short Time Fourier Transforms and Spectrograms

The Fourier transform is a mathematical transform that decomposes time or spatial domain functions into their constituent frequencies. While the spectrum of many signals, obtained by Fourier transform, can help with analysing the signal, it is insufficient for signals like audio that lack stationarity. The short time Fourier transform (STFT) was introduced as an alternative which includes both time and frequency content in one representation. [14] It works by sliding a window across a signal and performing a discrete Fourier transform at each windowed interval, appending the results in a 2D representation where one axis represents frequency, and the other represents time.

The Short Time Fourier Transform:

$$X[r,k] = \sum_{m=0}^{N-1} x[rR+m]\ w[m]\ \mathrm{e}^{-j\frac{2\pi}{N}kn} \tag{2.1}$$

The STFT is defined in equation 2.1, where x is a signal, w is a window function, R is the hop size, and N is the length of the underlying discrete Fourier transform. Hann and Gaussian windows centered at 0 are common choices for STFTs.

2.2.2 Speech Processing

Speech processing is a broad field of study that analyses human speech signals. Most often, these signals will be represented digitally, and analysed using a variety of techniques including digital signal processing, statistical methods, and of course machine learning.

The Mel Scale

The Mel scale is a logarithmic mapping of frequencies that is meant to replicate human perception of pitches by matching the spacing between pitches as judged by listeners. In particular, the scale is designed such that a pitch of 1000 on the Mel scale is equivalent to 1000 Hz, and all other pitches are defined in relation to this centerpoint. Because the scale is based on subjective perception of spacing between pitches, there is no single Mel scale, and different researchers have found different formulas for their scales. One popular example from O'Shaughnessy is shown in equation 2.2. [15]

$$m = 1127\ ln(1 + \frac{f}{700}) \tag{2.2}$$

While their is no single universally accepted Mel scale, most follow a very similar pattern, with a logarithmic relationship to the Hertz scale. The creation and use of

the Mel scale is not without its critics. [16] Despite the mixed reception in the fields of audiometry and psychoacoustics, the scale has empirically proven itself useful in the domain of deep learning. [17]

The Mel scale can be used to produce Mel spectrograms. They can be computed one of two ways. Either by first obtaining a regular spectrogram as described in section 2.2.1, and then re scaling the frequency axis by multiplying a transformation matrix. [18] Alternatively, they can be computed using Fourier transforms and filter banks. [19]

Diarization

Speaker diarization, or just diarization, is the process of partitioning and labeling segments of audio according to the identity of the speaker. A diarization algorithm must perform two essential tasks, speech detection, and speaker recognition. [20] Implementations of diarization will often start by identifying when changes in speaker occur, and then cluster the speakers using a range of algorithms.

One of the more popular methods of speaker diarization used to be a combination of Gaussian mixture models (GMM) to differentiate the speakers and hidden Markov models (HMM) to segment them within the audio. [21] These methods typically use lower-dimensional representations of the audio, such as Mel-frequency cepstral coefficients (MFCC), derived from the Mel scale described in section 2.2.2. However, with recent advances in deep learning, most methods today use deep neural networks. Neural networks were first used to provide richer latent embeddings of the speech signals using autoencoders, but eventually, full end-to-end neural diarization systems began outperforming other methods. [22]

Because diarization allows you to segment and isolate a speaker, it can be useful for tasks where a single person is targeted for analysis, but they are not the sole speaker in the recording, as is the case in the problem described in this thesis.

2.3 Deep Learning

Deep learning (DL) is a subset of machine learning which uses artificial neural networks (ANN) to perform a variety of tasks, notably in computer vision (CV), natural language processing (NLP), and automatic speech recognition (ASR) [23]. ANNs are named due to the inspiration they draw from their biological equivalent. They are made up of a collection of nodes, each representing a neuron, which take in a collection of inputs, perform calculations on them, and output the results. The computation involved in each unit commonly begins with a weighted sum of the inputs with an additional bias term, the result of which is then passed through some non-linear function.

In recent years, deep learning based algorithms have overtaken many expert-based and classical machine learning algorithms [23]. Examples can be seen in the fields of computer vision, with such models as convolutional neural networks (CNNs) [24], generative adversarial networks (GANs) [25], and vision transformers (ViTs) [26].

2.3.1 Artificial Neural Networks Overview

Neural networks are canonically represented as a series of layers. The model inputs are considered to be the first layer, while the outputs are the last layer. The layers in between come in many forms, and the operations that they perform are used to categorize the type of neural network (e.g. feed forward, convolutional, recurrent, etc...). Most operations are composed of two parts, a linear operation such as a weighted sum or a convolution, followed by a non-linear operation called an activation function. [27]

The performance of the model during training is evaluated using a loss function, which compares model outputs to ground truth labels. The adjustment of the model parameters (i.e. the training) is framed as an optimization problem, with the loss

serving as the objective function. The parameters of the operations are trained by gradient descent, which works by measuring a loss function at the output, representing the model error, and propagating it back from the output layer to all of the hidden layers and their parameters [28, 29].

Nodes, Weights, Activations

As described previously, nodes are the fundamental building blocks of neural networks. They can be thought of as containing single values, sometimes called activations for reasons which will soon become apparent. These nodes are typically organized in layers, which can be represented as matrices or higher-rank tensors. Two consecutive layers of a neural network are related by some non-linear function, which itself is usually a composite function built up of a linear and non-linear component.

A common example of a linear function would be a weighted sum, in which each node from one layer contributes to the value of each node in the subsequent layer. The result of that weighted sum would then pass through a non-linear function, called an activation function. The result of this function is called the activation, and it becomes the values of a node in the subsequent layer. A variety of activation functions are used in deep learning, among the most popular are the sigmoid or logistic function, the rectified linear unit (ReLU), and the softmax function [30]. Figure 2.2 schematizes the propagation of values in a basic neural network layer, called a dense or fully connected layer. If the layers can be represented as a single column vector as in figure 2.2, then the weighted sum can be represented as a matrix multiply.

2.3.2 Convolutional Neural Networks

Convolutional neural networks (CNNs) is a class of deep learning architectures whose fundamental operation is the convolution of an input or hidden layer by a kernel. They were first introduced for two-dimensional classification tasks, such as with hand-

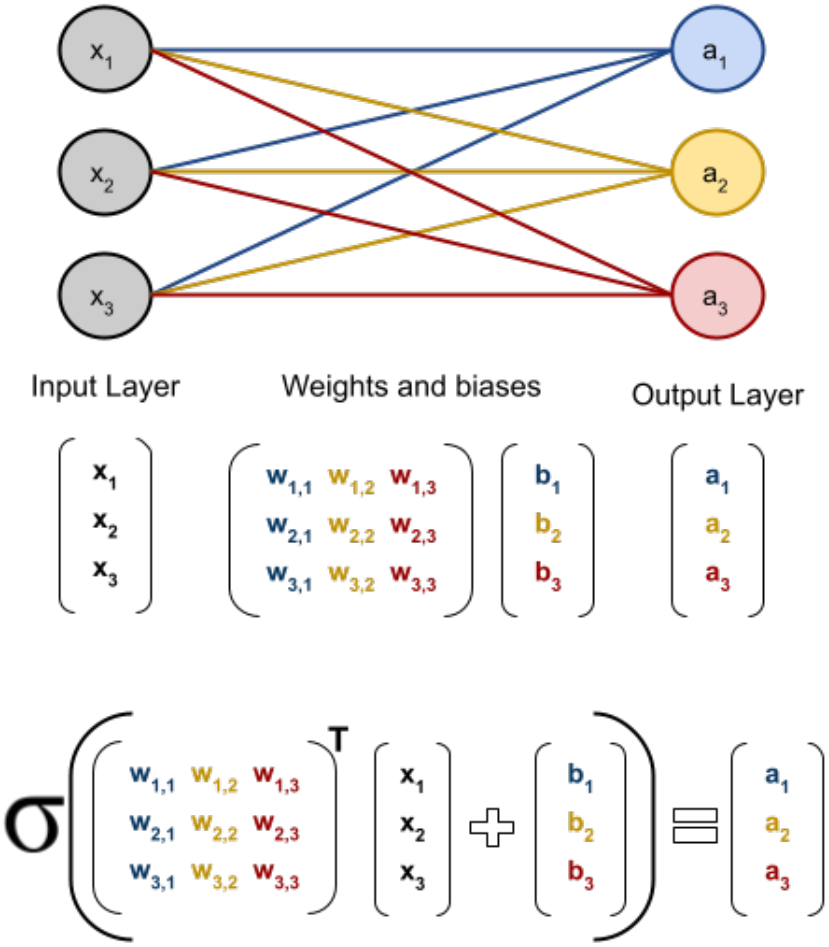

Figure 2.2: Forward propagation in a fully-connected layer of a feed-forward neural network

written digit recognition [31], but have grown in application and popularity in recent years as DL and CV continue to develop. One of the main strengths of the CNN is that the convolution operation is invariant to translations, which serves to make models more generalizable for pixel-based tasks.

Kernels and Convolution

The fundamental operation of a CNN is the convolution, in which one or more matrices called kernels are convolved with an input image. The results from each convolution are stored in a series of new matrices, to which an activation function can be

applied point-wise. The kernel operates identically to a traditional linear filter, and as such the names are often used interchangeably. However the values of the kernel are not fixed like with a linear filter, but are parameters of the model, and they can be trained by back propagation [32].Unlike a feed forward network which has unique weights for each node in a layer, the weights of the kernels are shared by each pixel in an image or intermediate layer.

The number of kernels in a convolutional layer determine the number of channels in the subsequent layer. If for example a grayscale image (one channel) is passed through a CNN layer with 16 filters, the output of the layer, called a feature map, will have 16 channels. While the size of each channel will depend on parameters of the CNN layer, this typically means the overall size of an input is increased by the convolutional operation.

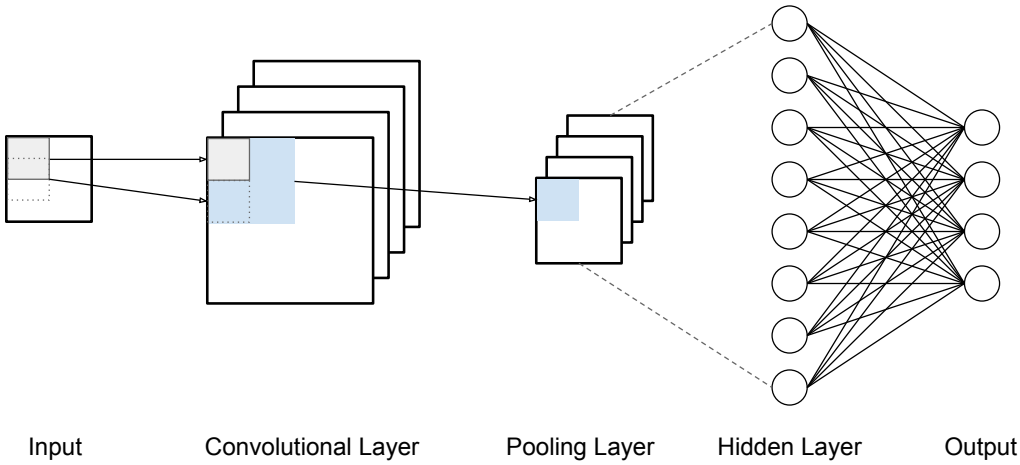

Figure 2.3: Basic architecture of a Convolutional Neural Network (CNN). An input image is passed through a convolutional layer, increasing its size channel-wise. This is followed by a pooling layer which reduces the spatial size of the resulting feature maps. For classification tasks, after a series of convolutional and pooling layers, the resulting feature map is flattened and fed into a series of fully-connected layers, with the number of nodes in the last layer matching the number of possible classes.

Pooling

Pooling is used to counteract this growth by artificially reducing the spatial size of the convolved feature map. Like a convolutional layer, a pooling layer also uses a kernel or filter. However, the output is a reduction of the input, such as a maximum, mean, or average. Max-pooling is often preferred, as it is thought to extract only the most dominant features, while suppressing noise. [27]

As the image passes through more pairs of convolutional and pooling layers, the number of channels in the feature maps is gradually increased, while the height and width is decreased. In deeper layers, the feature maps are said to be more expressive. A common analogy describes filters in earlier layers as detecting simple features like hard edges, while filters in deeper layers detect much more complex shapes or even objects. While the real representations of CNN are much more complicated [33], the analogy illustrates how deeper layers of the model develop a richer representation of the input with respect to the output. An equivalent analogy for audio and spectrogram classification is presented in figure 2.4.

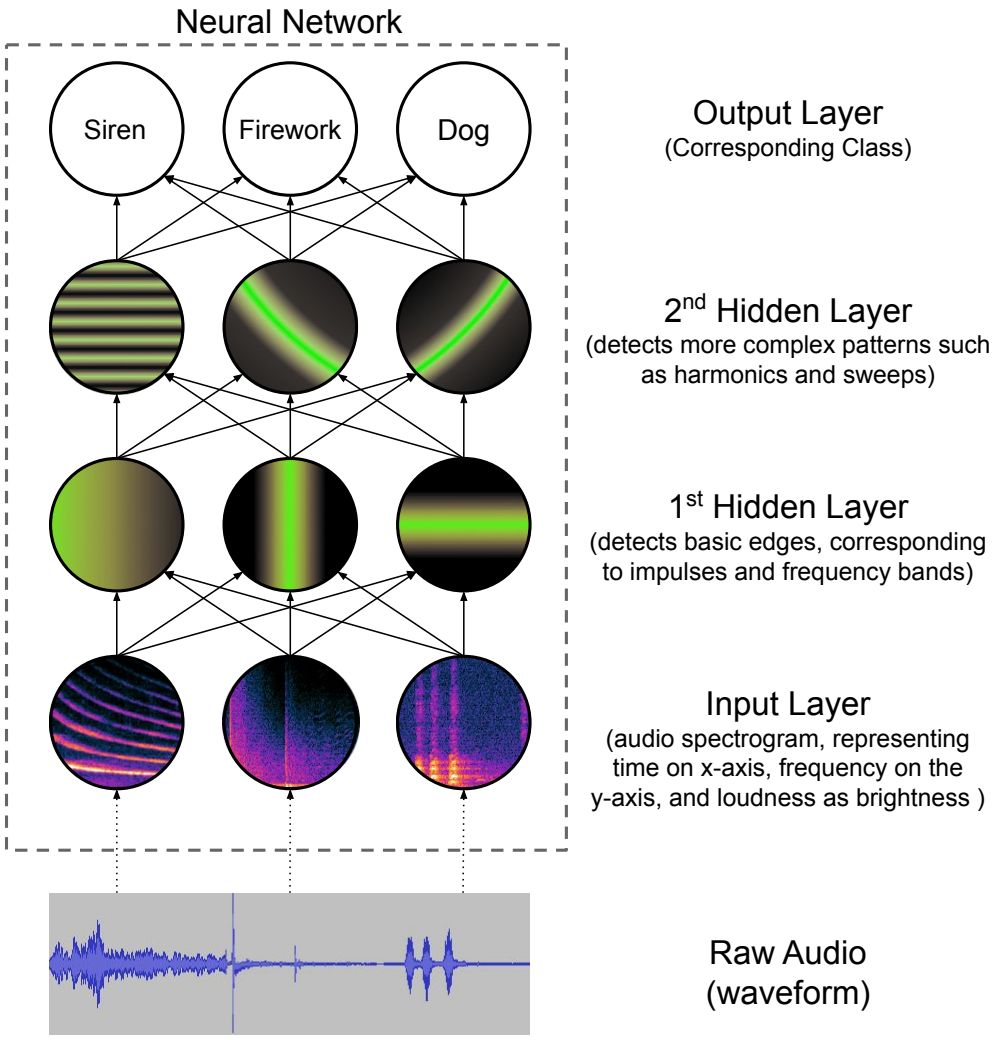

Figure 2.4: An Analogy for Feature Extraction in Convolutional Neural Networks. The working principle of the convolutional neural network is that it develops a richer understanding of the input with respect to the output as deeper layers of the model are reached. While this principle is often described in terms of edges and shapes, this figure illustrates how it can be intuitively understood when working on acoustic inputs. Waveforms are converted into spectrograms. Then they are processed by layers of increasingly complex filters, the first of which can be thought of as simply detecting individual frequencies or impulses, while deeper layers will interpret more sophisticated features like harmonics, echos, or frequency sweeps.

Chapter 3: Prior Works

3.1 Clinical Automated Speech Analysis

Studies on automated speech analysis have been made in a variety of clinical fields, particularly on the subjects of cognitive health, speech pathology, and mental health. While much of this work has been done using statistical analysis, or classical ML (section 3.1.1), there is a noticeable trend demonstrating the increased prevalence and success of DL methods (section 3.1.2). In addition to clinical research, a number of other advances are being made in the domain of audio classification (section 3.2).

Sections 3.1.1 and 3.1.2 are derived from a review paper I co-authored on clinical applications of algorithmic audio processing and the recent emergence of DL in the field. [34] The paper remains in preprint at the time of writing.

3.1.1 Statistical and Feature-based Approaches

Statistical methods, such as hypothesis testing, have been used routinely to analyze biomedical data [35]. Traditional ML methods, e.g., Support Vector Machine (SVM), Random Forest (RF) and k-Nearest Neighbor (kNN), have also been widely applied in biomedicine [36]. These methods are designed to sift through large amounts of data without any particular guiding (biomedical) hypothesis, to discover potentially actionable knowledge. A predictive model encapsulates a mathematical relationship

between the data describing an entity of interest, say a patient, and an outcome or label, say the disease status, of the entity. The purpose of this model is to make predictions of this outcome or label that are not yet known for other entities.

Early applications of these methods aimed to understand which features of clinical audio had explanatory or predictive power. A semi-automated approach assessed speech differences between children with cerebral palsy and controls by analyzing data from speech elicitation tasks [37]. Trained listeners transcribed speech recordings, which were used to determine word counts and what proportion of the words uttered matched the target elicitation (intelligibility). The study found that speech rate (words uttered per minute) and intelligibility classified normally developing children and those with cerebral palsy.

A similar study analyzed speech from picture-describing and sentence-repeating tasks to distinguish between patients with Alzheimer's dementia (AD) and those with mild cognitive impairment (MCI) [38]. The study found that the duration of speech and the increased likelihood of inserting or deleting to words in prompted sentences differentiated AD and MCI patients. Other studies used statistical tests like Mann-Whitney U to evaluate the association of pitch features with neuropsychiatric conditions [38, 39, 40].

Several studies have also used audio features alongside ML methods to classify patients according to neuropsychiatric conditions [38, 41, 42, 43] . Most of these studies use variations of the SVM algorithm, which finds an optimal boundary separating two classes of data points. One study used multiple SVM models to identify a motor speech disorder by determining the severity of unintelligible speech [42]. A sentence-level SVM trained on energy features and a phoneme-level SVM trained on pitch and time features yielded accuracies of 79.8% and 77.3% respectively. This accuracy increased to 84.8% when the SVMs were combined, an approach known as ensemble learning [36].

Other studies have found success in similar tasks using other traditional ML algorithms like RF and kNN [44, 45]. Several approaches also combined audio features with linguistic characteristics derived from textual transcripts of the audio [39, 43, 45, 46]. One study performed 3-way classification of levels of cognitive impairment (control, mild, and early stage Alzhiemer's disease) using an SVM trained on acoustic features derived using Praat26 and an automatic speech recognition (ASR) system. This classifier had an accuracy of 60%, which improved to 66.7% after combining the acoustic features with a variety of linguistic ones [47].

3.1.2 Deep Learning Methods

More recently, deep learning (DL) techniques have been used to characterize clinical conditions from patient audio [48, 49]. These techniques typically utilize much larger, and often less structured, datasets than statistical and traditional ML techniques. DL techniques generally utilize multi-layer neural networks to build implicit representations of datasets and enable various analysis tasks, including predictive modeling [50] . Due to this architecture, DL techniques are capable of building predictive models directly from audio recordings, without the need for the same level of feature extraction as in traditional ML.

Convolutional neural networks (figure 2.3) are among the most prevalent DL technique for audio due to their ability to represent contextual information in data, as discussed in section 2.3. Another established architecture which is particularly suited for time series modeling is the recurrent neural network (RNN) (figure 3.1), where sequential inputs pass through functional transformations at consecutively connected layers of the network [49]. Because information in RNNs tends to be lost as it propagates through time, RNNs are not suitable for processing audio waveforms at the sample level; instead intermediate representations of sound are used [51].

Most clinical audio processing that utilizes DL inputs pre-computed features into

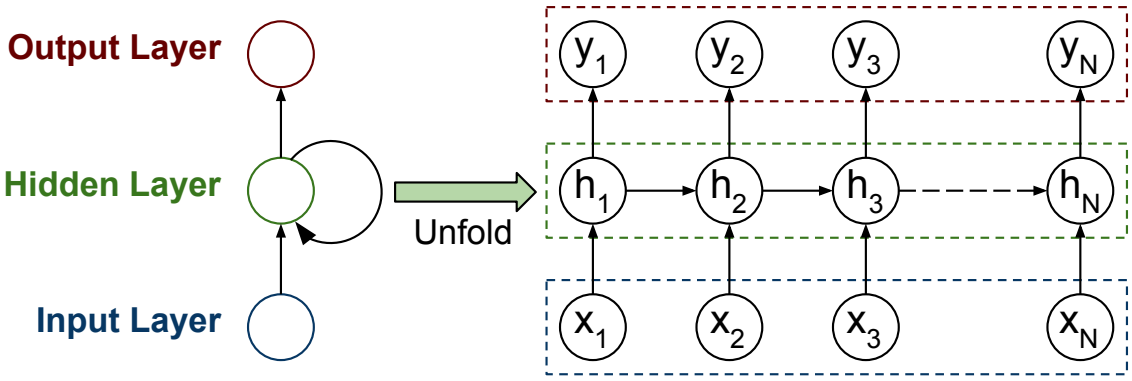

Figure 3.1: Recurrent Neural Network illustrated in two forms. The depiction on the left demonstrates the recurrent nature of the network, in which the output of the hidden layer is fed back as an input to itself in the next time step. The second depiction, called an unrolled RNN, shows more clearly how a time series is input into an RNN, with the output of the hidden layer propagating through time and serving as an input for future time steps.

a CNN, which can then predict the presence or degree of a condition [52, 53]. One investigation built a CNN model using GeMAPS [54] and other feature sets to classify depression severity [55]. A related approach trained parallel CNN models for different categories of audio features [I could add the table from the review paper] [55]. The output layers of each CNN were then concatenated and used to predict depression severity. Multiple investigations found that an ensemble of individual CNNs built from different data modalities (e.g., audio, text, and video) can predict depression severity even more accurately [52, 53, 56].

Other approaches have leveraged the sequential or temporal nature of audio. One AD detection effort employed a Time-Delayed CNN [57]. Instead of the entire recording, this approach applied the convolutional filter to utterances (segments of speech separated by silence) over all preceding time frames (hence the "delay"). This allowed them to extract local features from different temporal segments of the recording [57]. Another study used a Long-Short Term Memory architecture (LSTM, a sophisticated implementation of an RNN) [50] to screen for depression [58]. Mel frequency coefficients, discussed in section 2.2.2, were extracted from different temporal segments of

the audio and input to the LSTM. The outputs of the recurrent layers were then fed to the fully connected layers of the network to predict depression scores [58].

DL methods have also been used in situations of insufficient audio data. One study [58] employed transfer learning [50] to classify eight types of emotions from the relatively small RAVDESS dataset [59]. This approach repurposed an RNN trained on a data-rich task (depression score classification), and fine-tuned it with RAVDESS for a related task (emotion recognition). The approach achieved a validation accuracy of 76.3%, an increase of 8.7% from a baseline RNN trained solely on the RAVDESS [58]. Another useful DL architecture for data augmentation is a Generative Adversarial Network (GAN), which consists of two competing neural networks, a generator and a discriminator, to generate new data samples [50]. In an approach to diagnose childhood autism, the generator of a GAN was trained using GeMAPS and other feature sets extracted from the Child Pathological Speech Database, and a discriminator was trained to help the model generate more realistic data points [60, 54, 41]. Learned representations of the data were then extracted from the intermediate layers of the discriminator, and used to train an SVM model to classify four levels of pathology related to developmental disorders and autism [60].

Some approaches use x-vectors, which are DL-extracted representations trained for speaker identification in a conversation. These x-vectors can then be used with ML or DL methods to classify speakers with and without a pathology. Several studies reported better performance of this approach for Alzheimer's and Parkinson's disease diagnosis compared to feature-based methods.

Finally, recent DL methods learn which characteristics of a native audio signal are useful for classification. Zhao and colleagues used a hierarchical attention transfer network that reconstructed segments of patient speech using an autoencoder, and integrated them with LSTM representations from a speech recognition model to screen for depression. Another study compared using the raw audio signal, and its various

filtered versions, with a CNN. Filtering the signal boosted the CNN's ability to accurately classify levels of depression from patient speech, illustrating the advantages of effective pre-processing.

3.2 State-of-the-art Audio Classification

While clinical studies have pushed new applications of audio processing neural networks, a number of advances have been made with non-clinical application.

Driven by popular technologies like smart home assistants, a large amount of audio DL research has been conducted on the subject of end-to-end automatic speech recognition, also called speech-to-text. Because this is not directly relevant to the research presented this dissertation, I will differ to a review on the subject by Kim et al. [61].

The task that is most interesting for our purposes is that of audio classification. Hershey et al. have tested a variety of common CNN architectures designed for CV on multi-class audio classification tasks [62]. Among the models tested are variations of AlexNet [63], VGG [64], Inception [65], and ResNet [66].

Other studies have compared CNNs to RNNs [67] for the task of emotion classification from recorded speech, finding more success with the CNNs. Li et al. also developed specialized models for speech emotion recognition using asymmetric convolutional filters for frequency-based and time-based feature maps, as well as an attention-based pooling layer [68].

More recently, the popular attention mechanisms [69] and vision transformers [26] are making an appearance in the audio space, with the introduction of the Audio Spectrogram Transformer [70]. This architecture shows a lot of promise, and pushes the boundaries of state-of-the-art audio classification, but it requires a large amount of data to train.

Chapter 4: Methodology

4.1 Overview

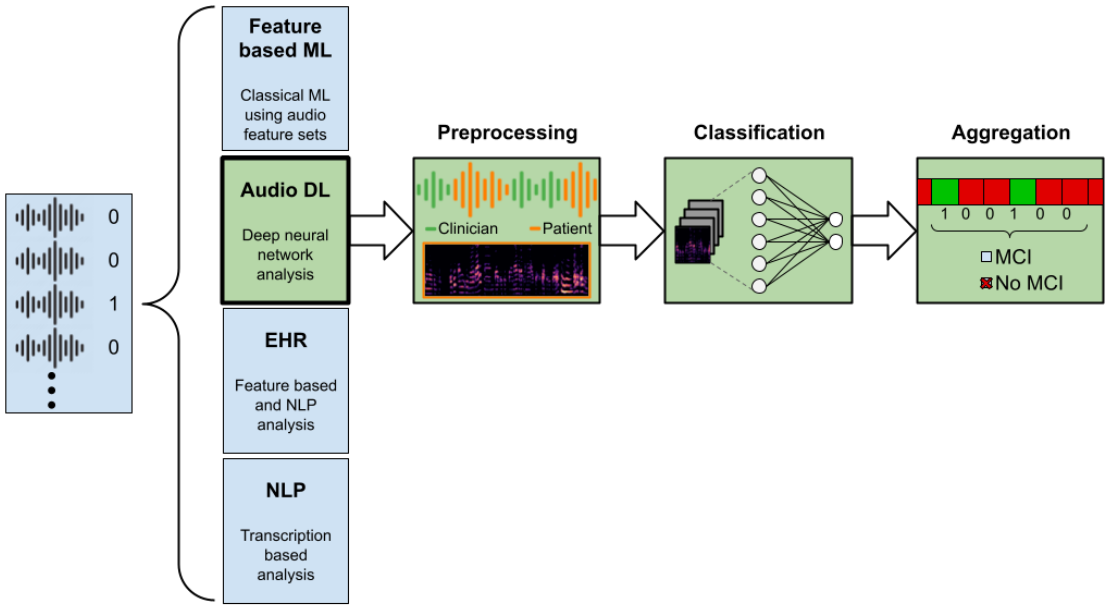

Figure 4.1: The research project aims to use different modalities to automatically identify and diagnose MCI. The modalities include electronic health records, transcriptions of primary care visits, and audio recordings of primary care visits, which can be analyzed either by classical ML using standard feature sets or by deep learning using only the raw audio. My contribution to the project is with the audio-only deep learning, for which I propose a pipeline to preprocess the audio, train neural networks, and make predictions.

4.2 Dataset

This research was conducted as part of an ongoing project at the Icahn School of Medicine at Mount Sinai. The aim of the project is "to develop and validate state-of-the-art machine learning (ML) algorithms to identify patients with mild cognitive impairment (MCI) in primary care using structured and unstructured data from the electronic health record (EHR) and automated speech analysis (ASA) of audio recorded patient-physician encounter" [1].

To accomplish this goal, the researchers assembled a dataset of 800 patients representative of the Mount Sinai patient population in the New York City area, who consented to having one of their primary care visits recorded via one to two microphones. The microphones were lavalier (clip-on) microphones connected to a digital audio recorder. Separately, they were screened for cognitive impairment via the widely used Montreal Cognitive Assessment (MoCA), and the scores were thresholded to provided a binary label for MCI. The result is a labeled dataset consisting of audio recordings of one-on-one conversations between patients and their primary care physicians, paired with a cognitive health outcome measured separately by the MoCA. An important distinction is that the cognitive assessment was not conducted as part of the primary care visit, and therefore was not recorded. The aim is to eventually use only the primary care visit to asses cognitive health instead of the MoCA.

4.3 Preprocessing

4.3.1 Diarization and Patient Identification

Because each audio recordings is unstructured and contains speech from both the patient and physician, in order to perform any analysis exclusively on the patient's speech, it is necessary to isolate it from the rest of the audio. This process of separat-

ing audio signals based on the identity of the speakers is called speaker diarization, or simply diarization [71]. In recent years, deep learning algorithms have become the most successful and are gaining in prevalence [72]. The task can be divided into two steps: speech detection and speaker recognition. In our case, it is also then necessary to identify and isolate the patient.

Speaker Diarization

For diarization, I chose to use pyannote-audio, an open-source speaker diarization toolkit built on the PyTorch framework [73]. Alongside the toolkit, pretrained models are also made available for download. These models achieve a near state-of-the-art diarization error rate (DER, see equation 4.1) ranging from 12.8% to 22.2% on a range of standard diarization test sets.

$$DER = \frac{false\ alarm + missed\ detection + speaker\ confusion}{total\ duration\ of\ time}) \tag{4.1}$$

The pyannote-audio diarization models perform the task in five steps, as pictured in figure 4.2. First, the audio is resampled, and standardized features are extracted for regular overlapping windows of the waveform. These features are then used to first perform speech activity detection, and further to detect changes in speakers. Each segment between a speaker change is then represented in a lower dimensional space as a speaker embedding. Finally, the speaker embeddings are clustered, and the clusters are used to label the various speech segments. Unfortunately, the algorithm as provided does not allow the user to input a known number of clusters. This will prove to be an issue in some cases when the algorithm separates a single speaker into multiple clusters and therefore labels.

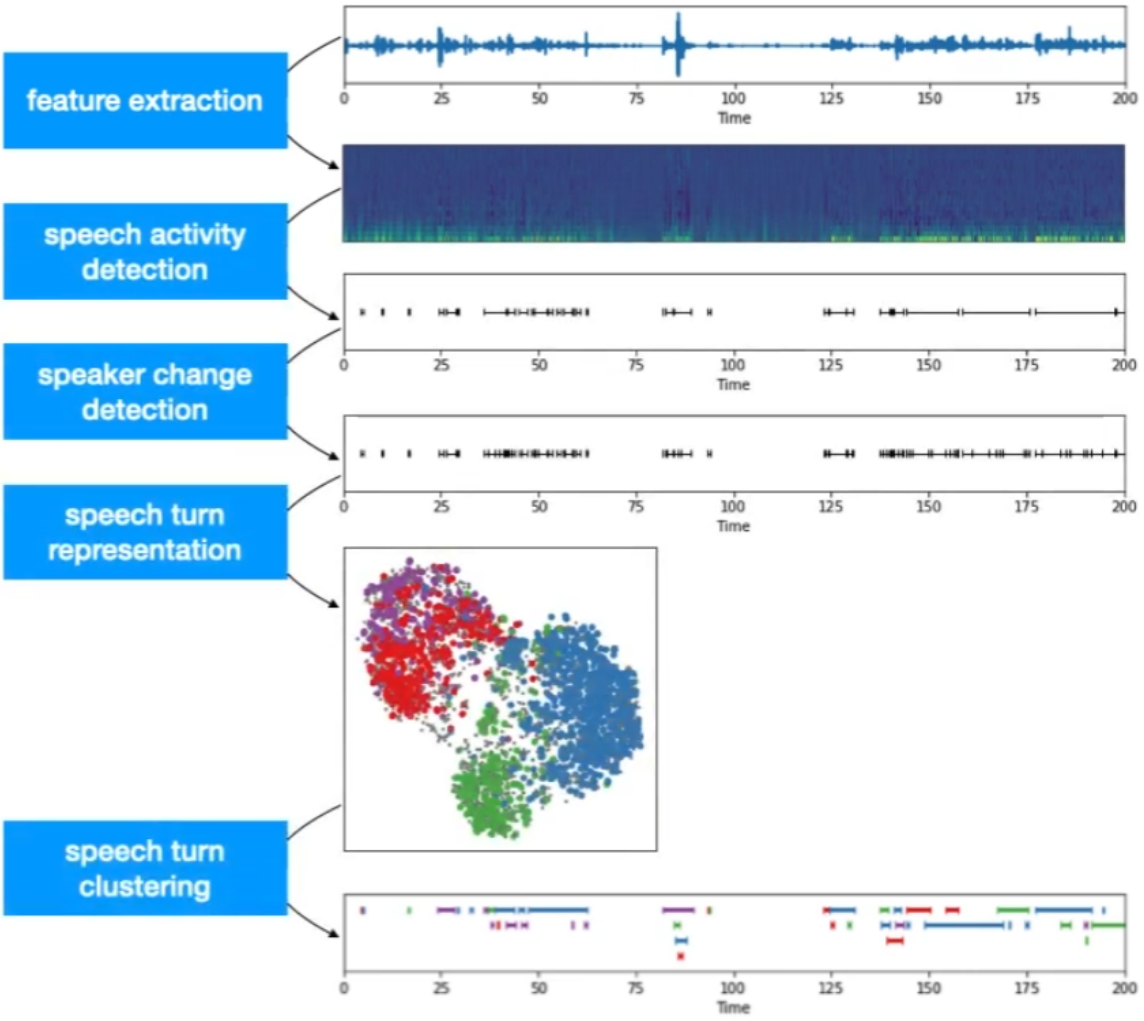

Figure 4.2: pyannote-audio is an open-source speaker diarization toolkit written in Python. Based on PyTorch machine learning framework, it provides a set of trainable end-to-end neural building blocks that can be combined and jointly optimized to build speaker diarization pipelines. The algorithm consists of a series of steps which serve to detect speakers, extract embeddings, and cluster them resulting in a labeled timeline. *This image is taken from the pyannote-audio documentation, and reused with explicit permission from the original authors* [73].

To process the patient-physician conversations with pyannote-audio, a script was used to loop over the audio files, convert them to the appropriate sample rate and bit depth, and input them through the pyannote-audio pretrained diarization model. While this algorithm can run nearly at real time on available hardware, with 691 audio files averaging 30 minutes each, I determined that processing all of the audio files one

at a time would take just over fourteen days to compute. Using cloud computing resources and the *joblib* library to facilitate the parallelization of python code [74], this time was reduced to under 12 hours. The results were saved as json files to avoid having to recompute any of this in the future. The first few segments of a sample output are listed below. Each segment contains a pair of start and end times, a unique track identifier, and a label corresponding to the speaker.

Listing 4.1: Example diarized segments using pyannote-audio

{'segment': {'start': 0.01015625, 'end': 0.85390625}, 'track': 'A', 'label': 'A'}
{'segment': {'start': 1.3517187499999999, 'end': 2.12121875}, 'track': 'B', 'label': 'A'}
{'segment': {'start': 2.51946875, 'end': 3.72603125}, 'track': 'C', 'label': 'A'}
{'segment': {'start': 3.99265625, 'end': 4.051718750000001}, 'track': 'D', 'label': 'A'}
{'segment': {'start': 5.23634375, 'end': 6.19146875}, 'track': 'E', 'label': 'A'}

These labeled sequences resulting from the diarization can be visualized on a color-coded timeline. Figure 4.3 shows three examples of labeled sequences corresponding to three different audio recordings. The first two show examples of successful diarization, where two speakers, the patient and clinician, were identified and their speech isolated. While there are a few segments labeled as a non-present third speaker, speaker C, they represent very little of the overall labeled speech. For contrast, the third example shows a recording for which multiple speakers were identified. Certain factors like speakers moving around the room or significantly changing intonation are identified as additional speakers instead of one, leading to a high error rate.

Patient Identification

Diarization alone can only tell us who is speaking using generic labels, but in order to analyze the patient's voice, we must be able to identify and isolate it. To accomplish this, I utilize special characteristics that I know *a priori* about the recordings. In particular, the patient identification will rely on the assumptions that there are only

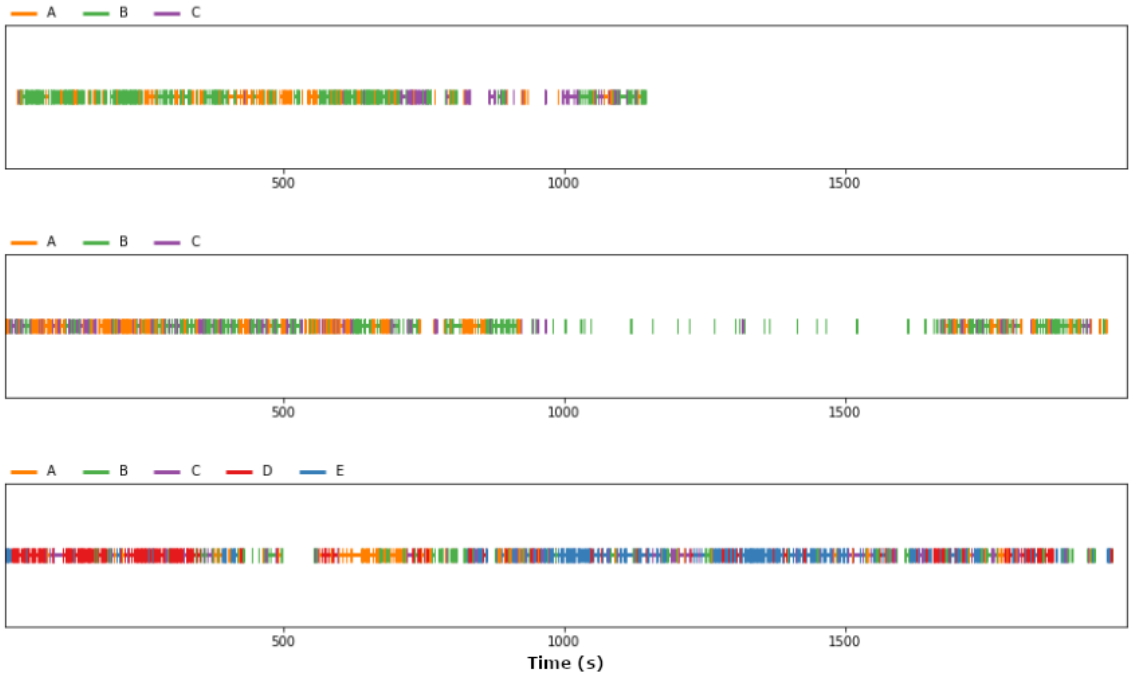

Figure 4.3: Three examples of labeled timelines for diarized recording: two typical good examples with both speakers correctly identified and minimal misclassification of speech attributed to a third speaker, and one fairly poor example where a larger portion of speech is misclassified as a number of other speakers.

two speakers present in the conversation, and that the patient is always the louder speaker, due to the lavalier microphone being placed directly on the patient.

There are three strategies that were considered for isolating the patient's speech. The first was to remove all non-patient speech, preserving any unlabeled white space. A second strategy was to consider only the two most prominent speakers, assuming they are the patient and physician, and keep only the loudest one. The final strategy is to keep only the audio from the loudest speaker, regardless of length or prominence. In each case, the average sum of squares, or average power of the signal, is used as a metric for loudness.

The first strategy strategy was quickly discarded, as most of the recordings contain a large amount of silence, when neither the patient nor the physician speak for a long period of time. In particular, there is usually a significant length of time at the end of the recording where the primary care visit has ended, but the microphone has not yet

	Strategy 2	**Strategy 3**
Accuracy	76.9%	98.3%
Mean Duration	4:45	5:14

Table 4.1: Accuracy and mean duration of resulting audio when applying two different patient identification strategies. In Strategy 2, only the two most prominent speakers are considered, and the loudest of the two is identified as the patient. In Strategy 3, the loudest speaker is identified as the speaker regardless of prominence. Accuracy is measured simply as the percentage of files in which the patient was correctly isolated, and was calculated based on a sample of 60 audio files out of 691.

been turned off by a technician. This is undesirable because it leads to a significant increase in noise in the data.

The second and third strategy should yield the same results if the patient is indeed the loudest and most prominent speaker, but in cases were the diarization has identified too many speaker, like the third example of figure 4.3, they could differ.

These strategies were difficult to compare. Ideally, a metric similar to DER would be used, but hand labeling the input data and diarized results would be too time consuming. Instead I opted for a binary approach; a diarization would be considered successful if the majority of the captured speech corresponds to the patient's, otherwise it would be considered unsuccessful. After running two scripts to execute each of the patient identification strategies, 60 audio recordings were randomly sampled, and I manually labeled the outcome of the patient identification on each one. The results are summarized in table 4.1.

Within the random samples, the second strategy of choosing the louder of the two most prominent speakers had an accuracy of 76.9% while the third strategy of choosing the loudest speaker regardless of prominence had an accuracy of 98.3% . The average duration of the resulting audio is also longer for strategy 3, which means we are capturing more training data overall.

Note that in cases were the diarization has mistakenly identified multiple speakers instead of just two, it is possible that the patient's speech has been given multiple

labels. Our strategy however only isolates one. This means our resulting files may not capture the entirety of the patient's speech. Nevertheless, it is most important for our purposes that the patient be properly isolated, even if some of their speech is not captured. While the average length of a recording using strategy 3 is longer than that using strategy 2, there were some examples (8) where strategy 3 produced shorter results. Both strategies reach the exact same result in 62% of cases.

Results

After applying the pyannote-audio diarization algorithm to each patient recording in the dataset, and applying strategy 3 to isolate the patients' speech, a new dataset of audio files was assembled. Because the diarization removes not only non-patient speech, but also lengthy silences, the average file was now less than one fifth as long, meaning a significant amount of noise was suppressed. This removes problems caused by outlier files that were too long, especially if the microphone was mistakenly left on. For example, the maximum audio length was now fifteen minutes instead of over two hours.

Strategy 3 outperforming Strategy 2 suggests that there are a number of cases in which neither of the most prominent speakers were the patient (e.g. if the clinician's speech was mistakenly classified as two different speakers). The remaining errors in Strategy 3 occur if another speaker appears louder, possibly due to non-speech noises like doors slamming or the patient coughing.

Summary statistics of the original audio files and their resulting diarized patient files are presented in table 4.2. While outlier files on the lengthier side are now condensed into shorter files, never exceeding 15 minutes and 8 seconds, there are still some outliers on the shorter end. The minimum file length being only a second long suggests a breakdown of the fundamental assumption that there are only two speakers in the room and the patient is the loudest. However, I note that only 5 files out of

	Duration Statistics (h:m:s)	
	Raw Audio	After Diarization
Mean	30:04	5:16
Median	28:23	5:59
St. Dev.	13:36	2:04
Min.	00:04	0:01
Max.	2:13:02	15:08

Table 4.2: Comparison of duration summary statistics between the initial raw audio and the final diarized and patient-isolated audio. The diarization has the effect of greatly reducing the size of the dataset, while eliminating noise.

691 are shorter than a minute, suggesting that this is a rare problem,

4.3.2 Mel Spectrograms

Audio as a waveform is fairly sparse with information, and the format is difficult for typical deep learning models to process. Instead, it is common practice to use spectrograms of the audio instead. As discussed in 2.2.1, short time Fourier transforms using the Mel scale can be used to produce Mel spectrograms. In addition to condensing the information, this also lets us use common neural network architectures like CNNs, originally intended for images.

Furthermore, because the majority of neural network architectures do not support variable-sized inputs, particularly when it comes to convolutional networks, the spectrograms have to be separated into regular segments. It is also good practice to have some overlap between adjacent segments to prevent the model from overfitting to the arbitrary boundaries. In addition to providing fixed sized inputs, these segments prevent model inputs from being too large. A typical audio processing CNN will have an input corresponding to a few seconds of audio at most, but not several minutes.

There are a number of parameters that can be tuned to produce adequate Mel spectrogram segments. These parameters, summarized in table 4.3, fall under three categories, the STFT parameters, Mel parameters, and segment parameters. Most of these parameters affect either the resolution or the length in the frequency and time

	Parameter Name	Example Value
STFT Parameters	Sample Rate	16000 Hertz
	# of FFT bins	2048 samples
	Window Length †	2048 samples
	Window Type	Hann (Raised Cosine)
	Hop Length †	512 samples
Mel Parameters	# of Mel Bins †	128 mel bins
	Mel Scale Type	Slaney
	Min. Mel Frequency	0 Hertz
	Max. Mel Frequency †	8000 Hertz
Segment Parameters	Segment Length †	256 frames
	Segment Overlap †	128 frames

Table 4.3: List of parameters that can be tuned for producing a dataset of mel spectrogram segments. Most of these parameters affect either the resolution or the length in the frequency and time directions. Parameters marked with a † are parameters that I would like to rigorously test in the future, as I believe they might have an effect on model performance.

directions. For example using a shorter FFT window length and hop length would increase the resolution of the spectrogram, but also make it lengthier. This trade off is significant with regards to deep learning because a higher resolution preserves more of the original signal, but the larger inputs are more difficult to get models to converge with during training.

The *librosa* library for python [18] was used to read audio files and generate Mel spectrograms. The same parameters as stated in table 4.3 were used, leading to spectrogram segments 8.22 seconds in length, with a 4.11 second overlap between adjacent segments.

In addition to generating the spectrograms, the power of the signal was converted to the decibel scale. Because the majority of frequencies are not present at the majority of times, the Mel spectrogram on its own is extremely skewed towards near-zero values. For example, 95 percent of pixels have a value less than 1, but the maximum is over 8000. Due to the logarithmic transformation, converting to the decibel scale leads to a smoother distribution, as portrayed in figure 4.4. The maximum volume of the signal is used as the reference in the decibel conversion, meaning this unit

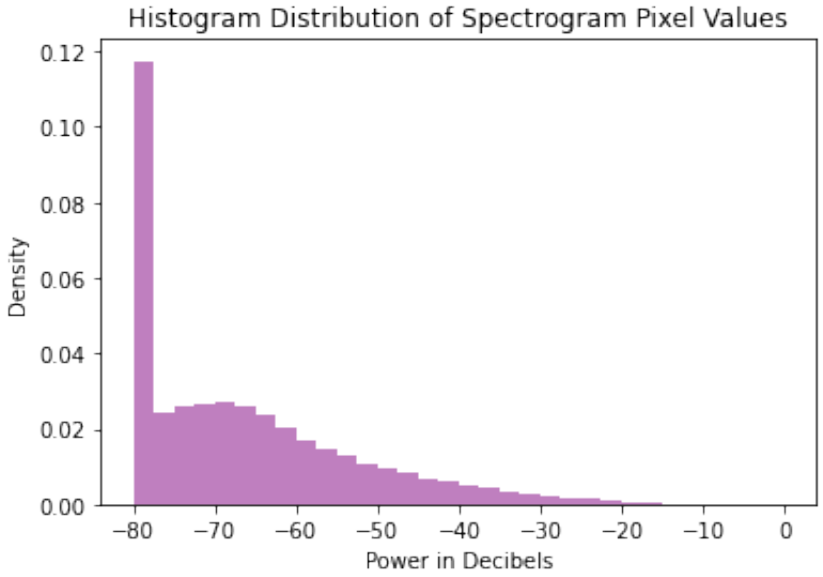

Figure 4.4: Three examples of labeled timelines for diarized recording: two typical good examples with both speakers correctly identified and minimal misclassification of speech attributed to a third speaker, and one fairly poor example where a larger portion of speech is misclassified as a number of other speakers.

conversion also serves as a form of normalization.

Once all the Mel spectrograms are generated and converted to a decibel scale, they are separated into regular overlapping segments and stored in a new directory. An example of such a segment is presented in figure 4.5

The *joblib* package was once again used to parallelize this task and speed up computation. As this is being done, a dataframe is being populated with the spectrogram filename, patient code, and the MoCA outcome label. This dataframe will serve as the labeled dataset, matching log-Mel spectrogram inputs to the MCI status outputs.

4.3.3 Train-Validate-Test Split

As is common practice in ML, the dataset was split into a training, a validation, and a test set. The distribution for each set was 80%, 10%, and 10% respectively. These sets are used to accurately asses the performance of the model at various stages of training using only data which it has not yet seen. In particular, the training data

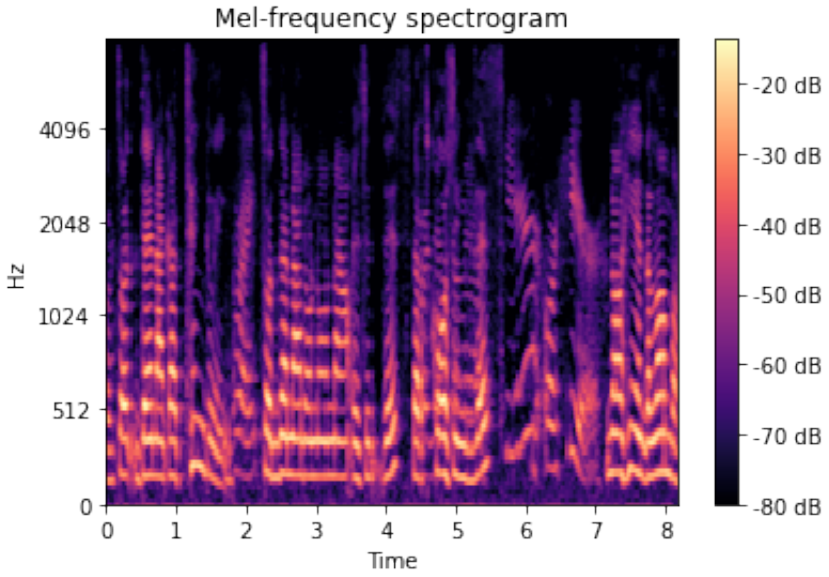

Figure 4.5: Example Mel spectrogram in the decibel scale. The reference for the dB conversion is the loudest sample in entire recording, so most spectrogram segments have a maximum value slightly below 0 dB.

will be used to train the deep learning models, which will be validated using the validation set. Models will be selected based on best performance on the validation set. Then, the methods of segment aggregation will be tuned using the validation test, and tested using the final test set. Because this audio classification model is meant to be a part of a larger multi-modal ensemble, a final external test set is also set aside to evaluate the performance of the finished product on completely new data. I was deliberately not given access to this data.

As an additional step, the train-validate-test splits were stratified, meaning that the distribution of the class labels was kept as constant as possible across the different sets. This ensures that each set consists of samples whose distribution closely resembles that of the represented population. The prevalence of the outcome was about 21.5% in each of the three sets.

4.4 Models

To learn to predict the MoCA outcome of a patient using the spectrogram segments, convolutional neural networks were trained. While not all the parameters stated in table 4.3 could be tested, one of the primary interests was to the different lengths of time at the input. This was due in part to members of the research group intuiting that most indicators of CI would happen over time rather than be captured by frequency. Two convolutional networks were tested for different input lengths corresponding to 4 seconds and 8 seconds. The results were then compared against available pretrained models, notably VGGish which is trained on AudioSet, a large audio dataset and ontology.

4.4.1 Convolutional Networks

The first set of models that were tested were CNNs which use single spectrogram segments as inputs to predict the outcome.

Model Architectures

The model architecture was inspired by simple CNNs such as LeNet [75] or AlexNet [63] which consist exclusively of convolutional layers, pooling layers, and fully connected layers. The convolutional and pooling layers gradually reduce the width and height of the input while increasing the depth of the so-called feature maps. Once the feature maps get sufficiently small, they are flattened, and the resulting vectors are fed through fully connected layers, which are essentially identical to a multi-layer perceptron. The final output is passed through a sigmoid activation to produce a probability of the outcome being positive.

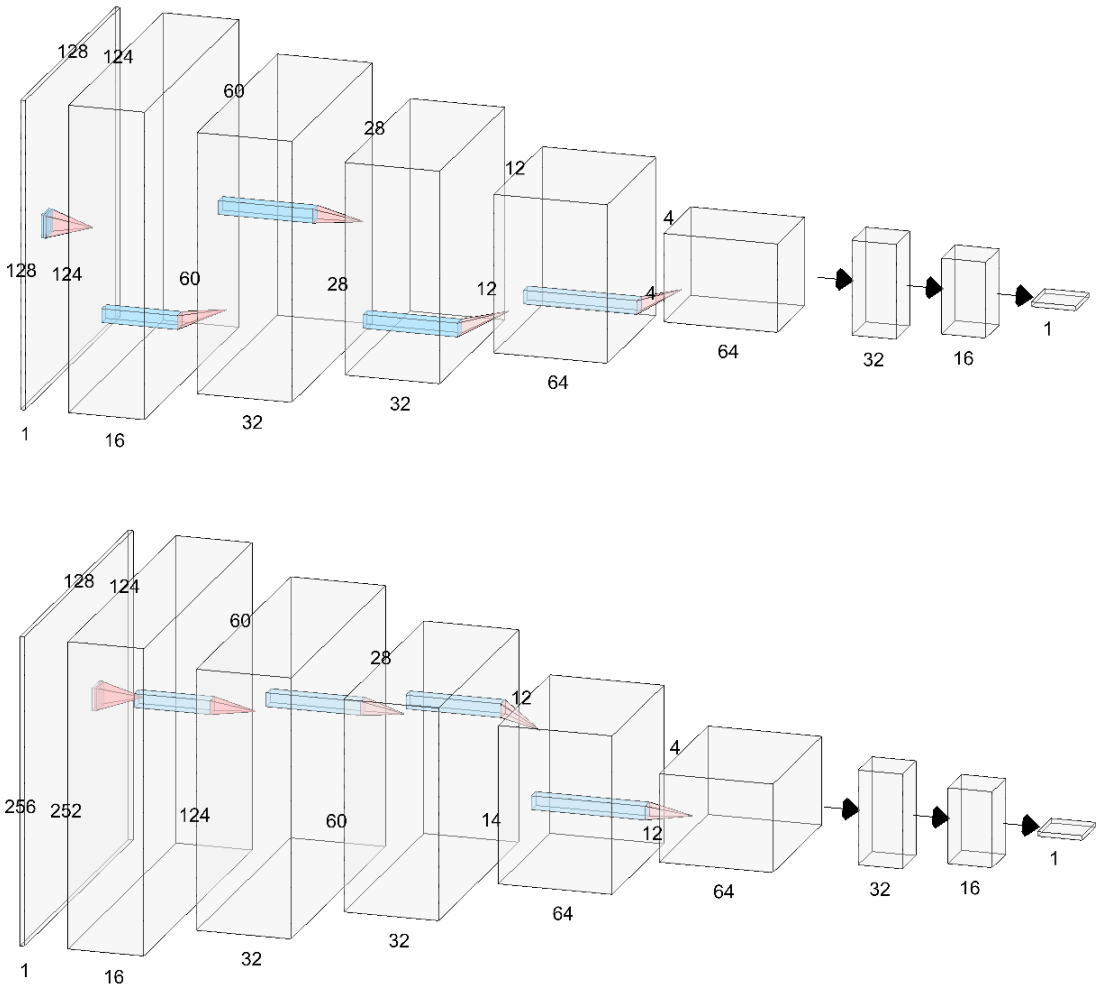

Figure 4.6: CNN model architectures for 4 second (128x128) and 8 second (128x256) inputs. The first convolutional layer has a 5x5 filter and all subsequent convolutions have 3x3 filters. All convolutions use a stride of 1x1 and are followed by a Max-Pool layer of filter size 2x2. The fully connected layers at the end of the model have 32, 16, and one node for the final output. All layers use ReLU activation, except for the output which uses sigmoid. The models have 78501 and 94885 parameters respectively.

Class Weights

In order to deal with the class imbalance, neural networks use what are called class weights. They allow models to account for the distribution of classes during training. They are implemented by modifying the loss function depending on the class, such that the loss is increased for less common classes. This will lead to more significant

gradient updates when classifying the rare outcome, thereby incentivizes predictions.

In Tensorflow, Keras has a built-in implementation of class weights, which can be fed in as a dictionary matching class keys to weight values. However, when using the dataset object *tf.data.Dataset* which improves the efficiency of input pipelines, the built-in class weights are not supported. Instead, I had to manually create a function wrapper for the Keras binary cross-entropy function which scales the loss by the appropriate weights. This function is available in the Appendix.

Additional Parameters

In addition to the architecture and class weights, deep neural networks contain a variety of hyperparameters that can selected and tuned. The adaptive moment estimation optimizer ADAM was chosen, with a default learning rate of 0.001. The models were trained for 100 epochs, but the best models based on validation performance were selected. Fully connected layers featured a dropout rate of 30% during training. Finally, a batch size of 64 was selected based on available GPU memory.

4.4.2 Transfer Learning with VGGish

Transfer learning is the practice of taking ML models that have already been trained on one task, and reusing them to solve a similar problem with different data. It is particularly useful for tasks like NLP or CV for which there exist very large training sets. For example image classification algorithms can be built on top of models pretrained on ImageNet, a labeled collection of over 14 Million images. Transfer learning allows smaller scale research projects to leverage information gained from much larger datasets using greater computational resources.

When using a pretrained neural network on a task for which it hasn't been trained, the majority of the layers are "frozen" meaning their weights are no longer updated by gradient descent and back-propagation. The last layer, or sometimes last few layers

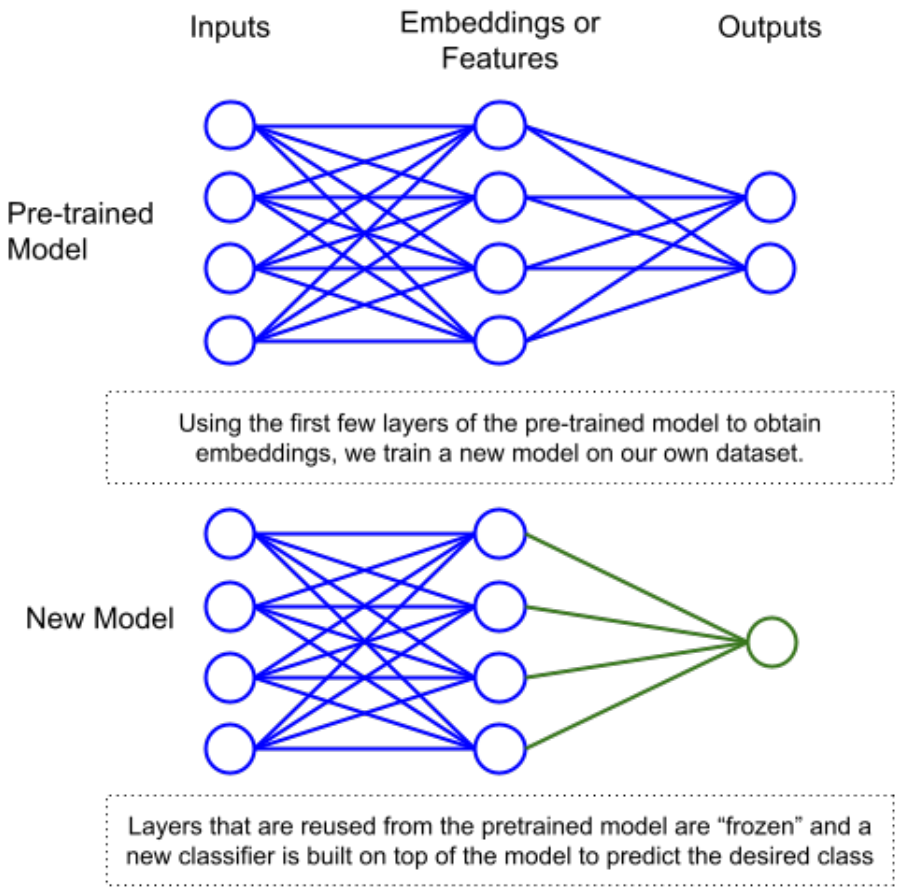

Figure 4.7: Transfer learning is the practice of using models trained on one task, and applying them to a different but similar task. In deep learning, this is typically implemented by reusing a pretrained neural network, but training an alternative classifier at the last few layers to match the desired number of classes. The process can be accomplished in three steps: selecting a source model that has been trained on an appropriate dataset, modifying the model architecture to match desired task, and finally tune the model with available data.

of the model, are removed, and replaced by layers in accordance with the desired task, as demonstrated in figure 4.7. For example, if the goal is classification, the last layer is replaced by one whose length matches the desired number of classes. The model is then trained again using the new dataset, a process commonly called fine tuning.

AudioSet

The chosen dataset for implementing transfer learning into this project is AudioSet [76]. It is a collection of just over 2 Million sound clips, each ten seconds long, hand-labeled into a hierarchical ontology of 527 classes. The audio is taken from YouTube videos, and covers a wide range of sounds including human speech, but also music, animals, ambient sounds, and a variety of miscellaneous noises. A partial list showing the class distribution is provided in figure 4.8.

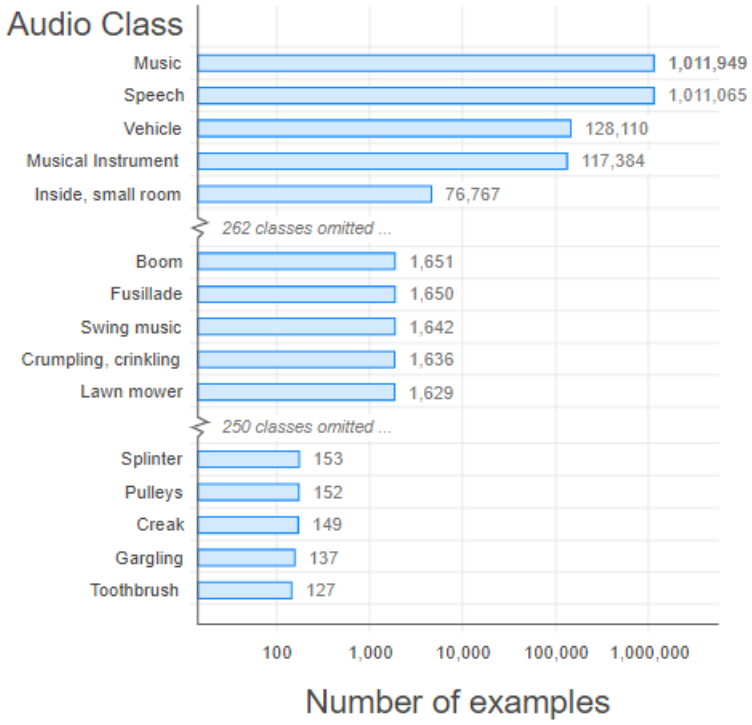

Figure 4.8: AudioSet is a labeled dataset containing 5.8 thousand hours of audio clips. Each 10 second audio file is hand labeled into one of 527 classes. This figure shows the distribution of certain classes, including speech which is the second most prominent class.

VGGish

VGGish [77] is a convolutional neural network for audio classification based on the VGG "very deep" convolutional image classification network [64]. The model was

developed alongside the AudioSet ontology to study the performance of 2D convolutional networks for large scale audio classification, and pretrained models are readily available via Tensorflow and Google APIs.

Similar to the input pipeline I used, it first converts audio to 16 kHz mono, then computes a STFT with a Hann window of size of 25ms and a hop length of 10ms. The spectrogram is then converted to the mel scale using 64 mel bins between 125 and 7500 Hz. Rather than convert to dB, it simply computes the logarithm of the Mel spectrogram using a small offset to avoid taking the log of 0. Finally, the resulting log Mel spectrograms are framed into non-overlapping segments 0.96 seconds in length.

Much like my own models described above, VGGish itself consists of a series of convolutional and pooling layers followed by a flattening of the feature maps, and a series of fully connected layers. The last layer of the model trained on AudioSet has 527 nodes to produce outputs according to the AudioSet classes, but an alternative "feature extraction" model is available which uses PCA to output audio embedding vectors of length 128.

Fine Tuning

Because VGGish is trained using 0.96 second segments, the dataset had to be regenerated to match the expected input format. With some help from one of the authors of the paper that introduced VGGish, it was confirmed that the pretrained architecture could not be modified without losing the benefits of pretraining. Once this was completed, to use VGGish with our output labels, the last few layers of the model were removed, all other layers were frozen, and four new fully-connected layers were added to reduce the feature vector from its 128 dimensional space to a single output class. These last layers of the model were trained over 50 epochs using the same hyperparameters as the CNNs described in section 4.4.1.

Chapter 5: Results and Discussion

5.1 Model Results

Once the models were trained using the training set, a prediction was made for each segment in the validation set. Because the last layer of each model was a single node with sigmoid activation, the output of the model is always between 0 and 1, and can be interpreted as the probability of a segment corresponding to a patient with MCI. Table 5.1 shows a few rows from a dataframe containing the validation data and predictions using the VGGish-based model pretrained on AudioSet.

patient_code	**audio_index**	**embedding**	**label**	**predictions**
patient_1	469	469	1	0.28602687
patient_1	470	470	1	0.28150588
patient_1	471	471	1	0.2954426
patient_2	0	472	0	0.073486425
patient_2	1	473	0	0.1042492
patient_2	2	474	0	0.16976058

Table 5.1: Sample outputs of VGGish based model on validation data from two patients with different MoCA outcomes. The audio_index is used to keep track of which 0.96 second segment the row corresponds to. The embedding number is used to find the matching embedding in the matrix containing all of the validation data. While the only identifying information in the patient code is the patient's initials, they were omitted from this table.

5.1.1 Receiver Operating Characteristics

Due to the imbalance in the class distributions, simple metrics like accuracy aren't particularly useful. A classifier could easily predict all cases as negative, and be correct about 78.5% of the time. Instead, we are interested in how it performs when identifying the less common positive cases. Two of the main metrics for this are the true positive rates (TPR) (also called sensitivity, recall, or detection probability) and false positive rates (FPR).

The receiver operating characteristic (ROC) curve is a graph that captures both of these metrics, and illustrates the diagnostic ability of a binary classifier. It tests a range of thresholds on the continuous predictions of the classifier, and plots the resulting TPR as a function of the FPR. For very low threshold, nearly all predictions will be classified as positive, so the TPR and FPR will both be near 1, tracing out the top-right portion of the curve. As the threshold increases, positive cases will become less common, and both the TPR and FPR will decrease, eventually tracing out the bottom-left of the curve.

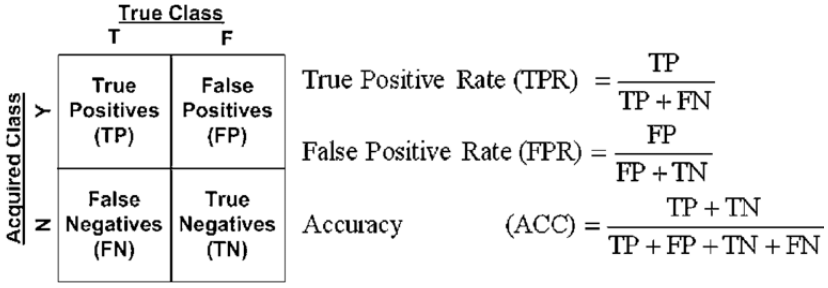

Figure 5.1: Visual representation of how to calculate the True Positive Rate (TPR) and False Positive Rate (FPR) using a confusion matrix. These metrics are used to plot the Receiver Operating Characteristic (ROC) curve and obtain the Area Under the Curve (AUC) metric.

The area under the curve (AUC) is a useful metric for comparing model performances. A baseline model's ROC curve will be a straight line on the diagonal from (0,0) to (1,1), and will therefore have an AUC of 0.5. If a model is making predic-

tions more accurately than random guessing, the ROC curve will be pushed above the diagonal, and the AUC will increase. A perfect classifier would have an AUC of 1.

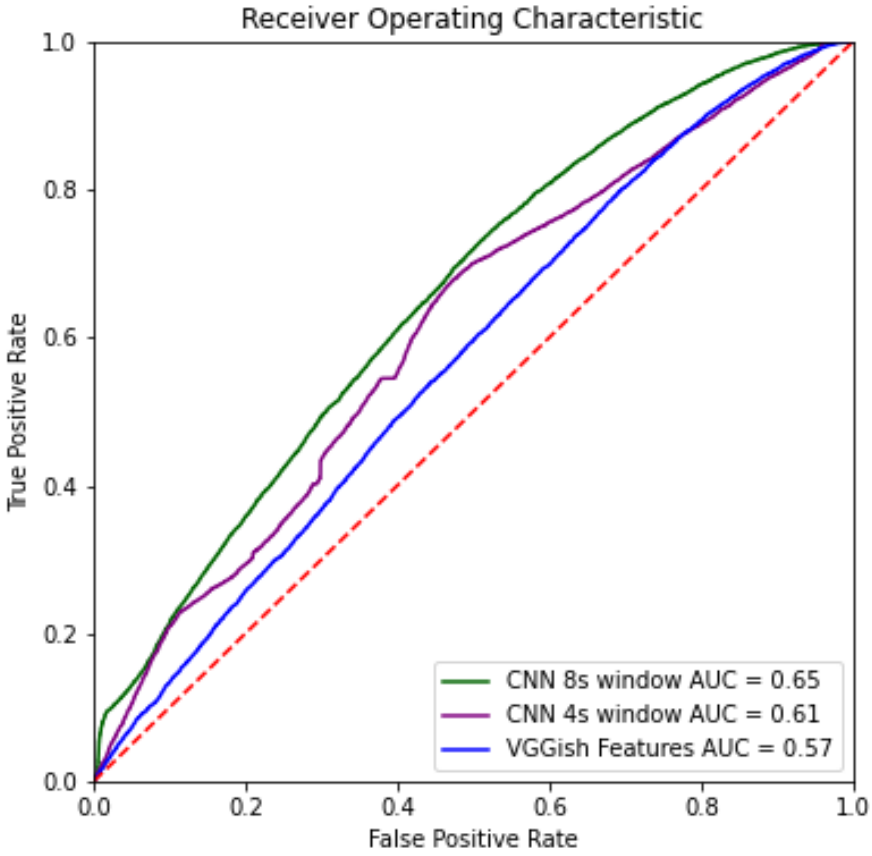

Figure 5.2: ROC curves and AUC metrics for three different models. Two models correspond to the convolutional networks described in section 4.4.1 and the remaining model corresponds to the network using VGGish embeddings with 0.96 second inputs. The dashed red line corresponds to a baseline model with an AUC of 0.5.

ROC curves along with their AUC are plotted for each of the three models described in section 4.4 in figure 5.2. They show a performance significantly above baseline for each of the three models. Additionally, it appears that longer receptive fields at the input lead to an increase in model performance. However, it is important to note that the models being compared are not perfectly equivalent apart from the

sizes of their inputs. The VGGish based model has only a 0.96 second receptive field, but it is significantly larger (in terms of number of layers and parameters) and is also pretrained using a larger dataset.

While there is room for improvement, these results clearly show the trained models outperform a baseline, and demonstrate evidence of the models extracting meaningful features and signal from the audio data.

5.1.2 Output Distributions

One way of analysing how the model is discriminating between the two classes is to treat the outputs based on the validation set as distributions. In figure 5.3, histograms of the outputs of two different models, with and without class weights, are shown for each of the two output classes. The frequency of the outputs is shown as well as the density. While it is clear from these graphs that the two classes are not easily discriminated by the models, the densities show that the distributions of the outputs are different based on the ground-truth label of each input.

Figure 5.3 demonstrates how the inclusion of class weights incentivizes the model to make more positive predictions. The figure also shows the skewness for each class. Whether the models were trained with or without class weights, the skewness is significantly higher for the outputs that should be positive, meaning that, while the classes are no fully separable, there is a clear difference between the classes being detected.

The fact that the outputs for ground-truth positive and negative CI status segments are different can be further supported by a Kolmogorov-Smirnov (KS) test. The KS test is a non-parametric statistical test that compares the cumulative distributions of two different samples in order to determine their likeness [78]. It can be used to test the null hypothesis: that two samples are drawn from the same population. Running this test on the outputs displayed in figure 5.3 yields p values of

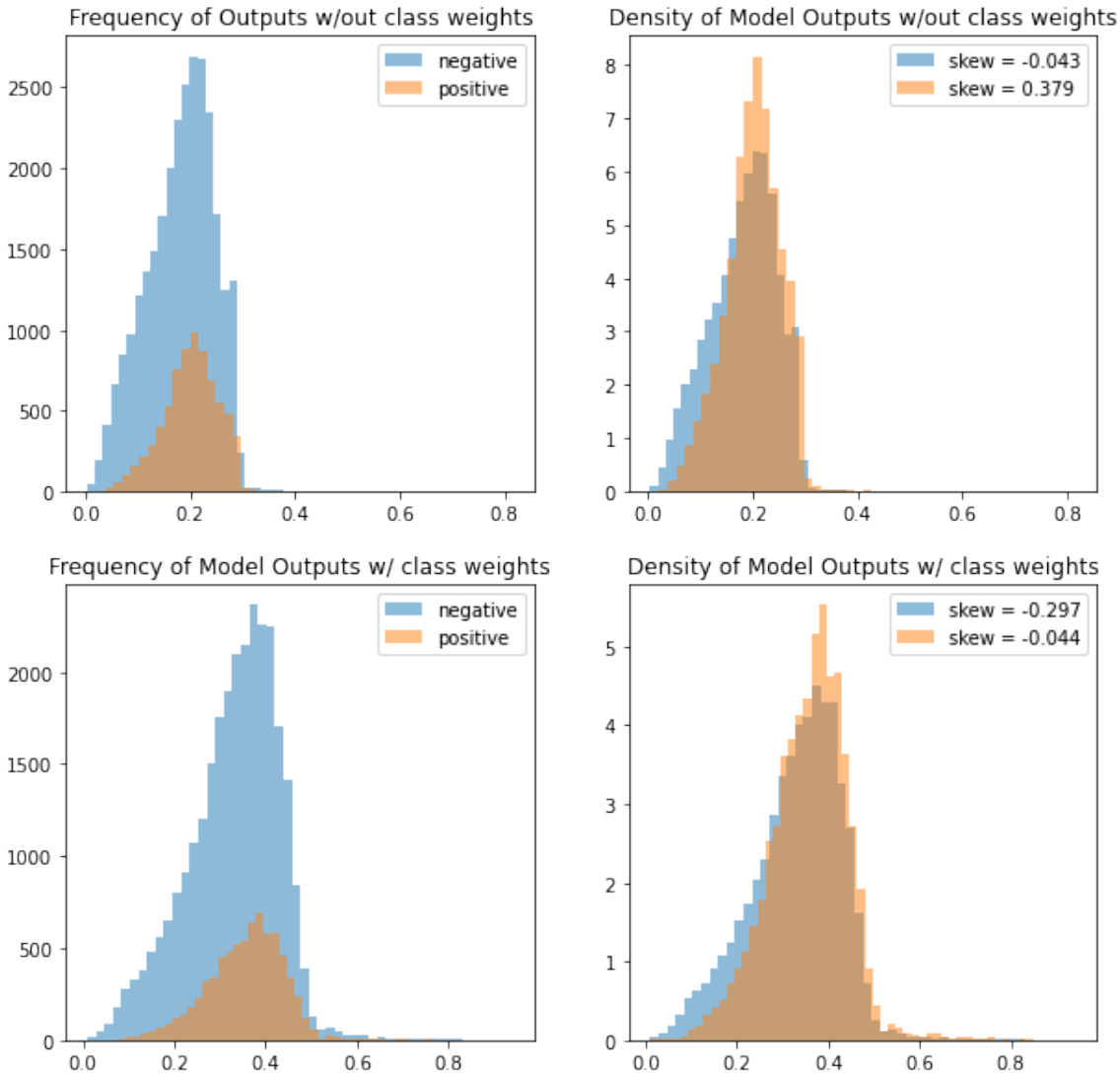

Figure 5.3: Distributions of model outputs for VGGish-based models with and without class weights, separated by CI status. The distributions show that the model weights are effective in incentivizing the model to make more positive predictions. While there is significant overlap in the outputs, showing that the two classes are not being discriminated effectively, there is a clear difference in the shapes of the distributions, particularly in the skewness. In both cases, the distributions of the positive CI statuses are significantly more positive skewed than their negative counterparts.

1.4^{-99} and 3.9^{-57} respectively, meaning that the probability that the positive and negative samples as represented by the model are drawn from the same distribution is extremely low. The null hypothesis is rejected.

5.2 Aggregation

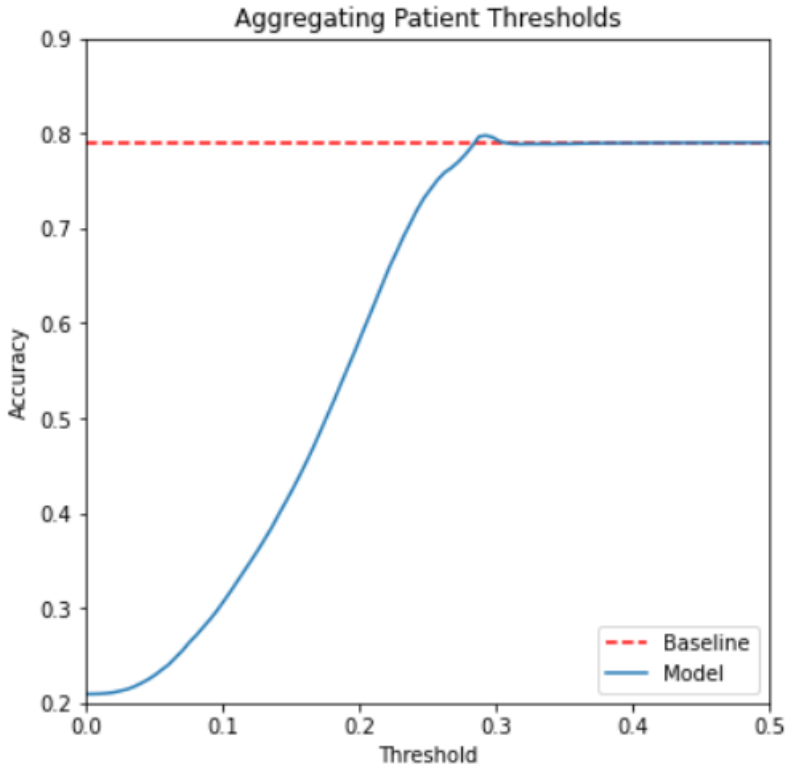

Figure 5.4: This graph plots patient-level diagnostic accuracy as a function of the threshold used to aggregate the segment-level scores. Despite the models being reasonably discriminatory at the segment level, this graph shows that they hardly outperform a baseline accuracy when combining the segment-level predictions.

While the models may be able to output statistically significant results on the segment level, it may be interesting to investigate whether these models on their own could provide useful predictions at the patient-level. To do this, the predictions must be aggregated somehow. It is important to note that this aggregation is not necessary in order for the model to perform within the ensemble described in figure 4.1. By design, the aggregation destroys much of the information that is contained within the sequences of individual predictions, information that may be useful to generate predictions when combined with features from other modalities (i.e. audio features, transcriptions, and EHR). Nevertheless, it is insightful to investigate how

these models would perform as the sole predictors of MCI.

One basic technique to test is a simple threshold. It can be implemented one of two ways. The first is by thresholding all the predictions for a single patient and using the resulting binary predictions to vote on the outcome. The second is to average all of the predictions for one patient and find a suitable threshold on the average for the final binary prediction. Both cases end up performing very similarly in practice.

In either case, a range of thresholds have to be tested to find which produces the best outcome. Because the models is biased towards the training data, its performance would not be representative of actual use. This is why we use the validation data, which at this point has only been used to validate the model's performance, to determine which thresholds would perform best. All results in the section are therefore measured using the test set.

As displayed in figure 5.4 despite the models being reasonably discriminatory at the segment level, testing a range of aggregation thresholds shows that the model is unable to reliably predict CI status at a patient level. There exists only a very small range of threshold values for which the model outperforms a baseline all-negative prediction, and the confidence in such predictions is very low, bordering on insignificant.

5.3 Discussion

Overview

To bring this back into context, let us consider each step in the pipeline described in figure 4.1. The contribution of this project represents just one of the modalities in a multi-modal analysis of a complex and highly unstructured dataset. Furthermore, the CI status of the patients being predicted by the models was determined by a MoCA test which was administered completely separately from the clinical visit which is recorded as the input audio. The pipeline consists of processing the data to isolate

the patient's speech and make the audio more suitable for deep neural networks, training and testing a range of models to make predictions based on segments of inputs, and attempting to aggregate segment scores to see if models based on audio alone can make reasonable predictions at the patient-level.

Diarization

The first step was the diarization and patient speech isolation. The pyannote-audio toolbox and pretrained model was tested along with a few different algorithms for determining which speaker was the patient. Selecting the loudest speaker regardless of prominence was successful in isolating the patient in 98.3% of sampled recordings, and reduced the average file length from 30 minutes to 5 minutes. However, it must be noted that a rigorous analysis of the diarization using a metric like DER was not performed due to the tedious and time consuming nature of hand-labeling audio. I know for a fact that some of the patients' speech is omitted from the final audio file, but I do not know how much. Nevertheless, this strategy intentionally prioritizes correctly identifying the patient over capturing all of their speech because the CI status label only corresponds to them, and not other speakers that might leak through.

Segmentation and Labeling

Once the patients' speech was isolated, the audio files were converted into Mel spectrograms which were then framed into regular overlapping segments. Each segment, regardless of length, inherited the same label as the overall file. This naive strategy of passing the labels from the overall file to each segment is problematic for a few reasons.

Take for example a patient who has tested positive for MCI according to the MoCA test. The initial label applied to the entire recording indicates that there should be evidence of MCI somewhere in the patient's speech. If we divide the audio

into 4 second segments and label each of them with a 1, we are essentially telling our models that there is evidence of MCI in every single segment. In practice, however, this is very unlikely to be true; most segments might capture the patient speaking indistinguishably from someone without MCI. In other words, we are training the model based on a lot of false positives labels, making the two output classes very difficult to separate.

We see some evidence of this in the output distributions of figure 5.3. The distributions for ground-truth positive and negative outcomes overlap substantially, and it is only the strongest outliers that differentiate themselves from the center cluster. The difference in skewness offers some evidence that the models are able to discriminate between the classes to some degree, but the distributions remain mostly inseparable overall. It may be the case that these models can not improve significantly unless this naive labeling of the segments is revisited. This is discussed further in the conclusion.

Model Performance and Aggregation

The performance of the models on a segment level based on ROC and AUC has shown that the neural networks are capable of extracting meaningful signal from the processed audio. Figure 5.2 appears to show a correlation between input length (in seconds) and model performance. However it is important to note that these models are not identical apart from the input length. The first and most obvious difference is that the VGGish-based model is pretrained on AudioSet, giving it access to a lot more training data, even if most of it is unrelated to this clinical field of study. The fact that it performs nearly as well as a CNN trained only on our dataset, but with an input over 4 times as long suggests there are very strong benefits to pretraining. This motivates future exploration of pretraining custom architectures more tailored to our long form audio.

While the models show significant improvements on baseline performances at a

segment level, the attempts at aggregation have demonstrated that these models alone cannot serve as adequate discriminators for MCI at a patient level. However, the goal was not for these models to perform the entire task on their own. Eventually, they will be combined with other models based on extracted audio features, transcriptions, and EHR. The models as presented could serve as feature extractors for an ensemble model which incorporates the other modalities. Furthermore, the other modalities can add a layer of self-supervision to the pure audio models, and help improve the aforementioned naive segment labeling strategy. This is discussed further in the future work section.

Chapter 6: Conclusion

6.1 Summary

In this thesis, I presented a deep learning pipeline for detecting MCI from long form clinical audio. First, I introduced a processing methodology for recordings of conversations between patients and physicians during a primary care visit. This pipeline uses existing neural network models alongside domain knowledge to correctly isolates the patient's speech in 98% of sampled recordings, reducing noise in the data by suppressing extended silences and speech from other speakers. Then, to convert the result into a more suitable format for audio classification neural networks, the audio is further processed into decibel-scaled Mel spectrograms and split into overlapping regular segments, with each segment inheriting the label of the parent audio. The resulting dataset is used to train convolutional neural networks from scratch, as well as to fine-tune models pretrained on larger labeled datasets. These models convincingly exceed a baseline performance at the segment level, with an AUC ranging from 0.57 to 0.65 based on the model architecture and training regimen. There is an apparent correlation between models having a larger input window and better performance, but this remains to be further explored. While the models perform reasonably well at the segment level, aggregating the scores to obtain a single prediction at the patient level remains a challenge. Nevertheless, this research demonstrates an ability to pro-

cess the unstructured recordings and extract a signal with some predictive power, and which can be informative within the context of the greater ensemble being developed.

6.2 Future Work

There remains a substantial amount of work before the entire multi-modal research project is fully realized, but for this discussion on future work, I would like to focus on the audio-only deep learning methods.

Diarization

While the speaker diarization ultimately proved successful in isolating and identifying the patient, there is evidence that not all of their speech was captured. Without a rigorous and time consuming labeling of the speakers, which would require annotating the speakers in the audio second by second, it is impossible to know the exact error rate. Such annotations would allow us not only to measure in more detail the accuracy of the diarization, but also to fine tune the performance of the neural diarization model provided by pyannote-audio, and obtain a more accurate diarization.

Spectrogram and Model Hyperparameters

As with most deep learning project, there are always many hyperparameters left to test and tune.

Among the ones I am most interested in exploring are the spectrogram parameters noted in table 4.3. These parameters determine the resolution of the model inputs in both the frequency and time dimensions, as well as overall size of the input. There is a trade-off between using too large inputs that become hard to converge a model with, and spectrograms that have been condensed beyond recognition, and I suspect that by performing a grid-search through these input parameters, an optimal set can

be found.

Furthermore, the model architectures and training regimen can still be optimized. Using models pretrained on AudioSet proved to be useful, but we are then limited by available models and their constraints. For example the VGGish model used in section 4.4.2 only considers inputs 960 ms in length, and would fail to notice any temporal dependencies outside of that range. It may prove useful to pretrain models with larger receptive fields like my own on datasets like AudioSet.

Self-Supervision

While the overall ensemble classifier doesn't necessitate that each sub-model output patient-level predictions (i.e. the final ensemble may benefit from using extracted feature vectors or embeddings), improving patient-level predictions to a level similar to those at a segment-level would likely be greatly beneficial to the model. Part of this problem may stem from the labeling of the data. Not only are the original labels fairly weak, since they were obtained by a test which was conducted independently of the audio recording, but they are further diluted by the segmentation of the spectrograms. By naively assigning each segment the label of parent audio, we are conditioning the models to find evidence of CI where there may be none. This leads to sub-optimal training, but also to difficulty in validating the segment-level performance. One potential solution is to treat this more like a self-supervised learning problem than an outright supervised learning one. Finding some way to increase the granularity of the labels could greatly improve model performance both on a segment level and on a patient level.

Bibliography

[1] A. Federman, J. Wisnivesky, and G. Pandey, "Natural language processing and automated speech recognition to identify older adults with cognitive impairment," Apr 2020. [Online]. Available: https://reporter.nih.gov/search/YC04oG6bIECPlQ_vf_LmGg/project-details/10144364

[2] Y. Lu, C. Liu, D. Yu, S. Fawkes, J. Ma, M. Zhang, and C. Li, "Prevalence of mild cognitive impairment in community-dwelling Chinese populations aged over 55 years: a meta-analysis and systematic review," *BMC Geriatr*, vol. 21, no. 1, p. 10, 01 2021.

[3] M. F. for Medical Education and Research, "Mild cognitive impairment (mci)," Sep 2020. [Online]. Available: https://www.mayoclinic.org/diseases-conditions/mild-cognitive-impairment/symptoms-causes/syc-20354578

[4] L. Ma, "Depression, anxiety, and apathy in mild cognitive impairment: Current perspectives," *Frontiers in Aging Neuroscience*, vol. 12, 2020.

[5] N. I. on Aging and N. I. of Health, "What is mild cognitive impairment?" 2021. [Online]. Available: https://www.nia.nih.gov/health/what-mild-cognitive-impairment

[6] A. J. Saykin, L. Shen, X. Yao, S. Kim, K. Nho, S. L. Risacher, V. K. Ramanan, T. M. Foroud, K. M. Faber, N. Sarwar, and et al., "Genetic studies of quantitative

mci and ad phenotypes in adni: Progress, opportunities, and plans," *Alzheimer's amp; Dementia*, vol. 11, no. 7, p. 792–814, 2015.

[7] C. Cooper, R. Li, C. Lyketsos, and G. Livingston, "Treatment for mild cognitive impairment: Systematic review," *British Journal of Psychiatry*, vol. 203, no. 4, p. 255–264, 2013.

[8] Z. S. Nasreddine, N. A. Phillips, V. BÃ©dirian, S. Charbonneau, V. Whitehead, I. Collin, J. L. Cummings, and H. Chertkow, "The montreal cognitive assessment, moca: A brief screening tool for mild cognitive impairment," *Journal of the American Geriatrics Society*, vol. 53, no. 4, p. 695–699, 2005.

[9] D. E. Barnes, K. E. Covinsky, R. A. Whitmer, L. H. Kuller, O. L. Lopez, and K. Yaffe, "Predicting risk of dementia in older adults: The late-life dementia risk index," *Neurology*, vol. 73, no. 3, p. 173–179, 2009.

[10] D. E. Barnes, A. S. Beiser, A. Lee, K. M. Langa, A. Koyama, S. R. Preis, J. Neuhaus, R. J. McCammon, K. Yaffe, S. Seshadri, and et al., "Development and validation of a brief dementia screening indicator fornbsp;primary care," *Alzheimer's amp; Dementia*, vol. 10, no. 6, p. 656, 2014.

[11] C. O'Driscoll and M. Shaikh, "Cross-cultural applicability of the montreal cognitive assessment (moca): A systematic review," *Journal of Alzheimer's Disease*, vol. 58, no. 3, p. 789–801, 2017.

[12] L. Jennings, A. Hackbarth, N. Wenger, Z. Tan, and D. Reuben, "An automated approach to identifying patients with dementia using electronic medical records," *Innovation in Aging*, vol. 1, no. $\text{suppl}_1, p.1381–1382, 2017$.

[13] Microsoft Corporation, "RFC 2361 - WAVE and AVI Codec Registries," IETF, Jun. 1998.

[14] E. Sejdić, I. Djurović, and J. Jiang, "Time–frequency feature representation using energy concentration: An overview of recent advances," *Digital Signal Processing*, vol. 19, no. 1, pp. 153–183, 2009. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S105120040800002X

[15] D. O'Shaughnessy, *Speech Communications: Human and machine*. Wiley-IEEE Press, 2000.

[16] D. 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, no. 1-2, p. 199–224, 1997.

[17] M. Huzaifah, "Comparison of time-frequency representations for environmental sound classification using convolutional neural networks," 2017.

[18] B. McFee, C. Raffel, D. Liang, D. P. Ellis, M. McVicar, E. Battenberg, and O. Nieto, "librosa: Audio and music signal analysis in python," in *Proceedings of the 14th python in science conference*, vol. 8, 2015.

[19] L. R. Rabiner and R. W. Schafer, *Theory and applications of Digital Speech Processing*. Pearson Higher Education, 2011.

[20] B. Gold, N. Morgan, and D. Ellis, *Speech and Audio Signal Processing*, second edition ed. John Wiley amp; Sons, 2011.

[21] X. Anguera, S. Bozonnet, N. Evans, C. Fredouille, G. Friedland, and O. Vinyals, "Speaker diarization: A review of recent research," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 20, no. 2, pp. 356–370, 2012.

[22] T. J. Park, N. Kanda, D. Dimitriadis, K. J. Han, S. Watanabe, and S. Narayanan, "A review of speaker diarization: Recent advances with deep learning," 2021.

[23] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, p. 436–444, 2015.

[24] J. Gu, Z. Wang, J. Kuen, L. Ma, A. Shahroudy, B. Shuai, T. Liu, X. Wang, G. Wang, J. Cai, and T. Chen, "Recent advances in convolutional neural networks," *Pattern Recognition*, vol. 77, pp. 354–377, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0031320317304120

[25] A. Creswell, T. White, V. Dumoulin, K. Arulkumaran, B. Sengupta, and A. A. Bharath, "Generative adversarial networks: An overview," *IEEE Signal Processing Magazine*, vol. 35, no. 1, pp. 53–65, 2018.

[26] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," *CoRR*, vol. abs/2010.11929, 2020.

[27] C. C. Aggarwal, *Neural networks and deep learning.* Springer, 2019.

[28] A. G. Baydin, B. A. Pearlmutter, and A. A. Radul, "Automatic differentiation in machine learning: a survey," *CoRR*, vol. abs/1502.05767, 2015.

[29] R. Hecht-Nielsen, "Theory of the backpropagation neural network," *International Joint Conference on Neural Networks*, p. 593–611, Jun 1989.

[30] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, "Activation functions: Comparison of trends in practice and research for deep learning," *CoRR*, vol. abs/1811.03378, 2018.

[31] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, vol. 1, no. 4, p. 541–551, 1989.

[32] R. Szeliski, *Computer vision: Algorithms and applications*, 2nd ed. Springer, 2022.

[33] N. Baker, H. Lu, G. Erlikhman, and P. J. Kellman, "Deep convolutional networks do not classify based on global object shape," *PLOS Computational Biology*, vol. 14, no. 12, 2018.

[34] A. Kumar, T. Jaquenoud, J. H. Becker, D. Cho, M. R. Mindt, A. Federman, and G. Pandey, "Can you hear me now? clinical applications of audio recordings," *medRxiv*, 2022. [Online]. Available: https://www.medrxiv.org/content/early/2022/02/08/2022.02.07.22270598

[35] J. H. Zar, *Biostatistical Analysis*. Pearson, 2019.

[36] E. Alpaydin, *Introduction to machine learning*. The MIT Press, 2020.

[37] K. C. Hustad, A. Sakash, A. T. Broman, and P. J. Rathouz, "Differentiating typical from atypical speech production in 5-year-old children with cerebral palsy: A comparative analysis," *American Journal of Speech-Language Pathology*, vol. 28, no. 2S, p. 807–817, 2019.

[38] A. König, A. Satt, A. Sorin, R. Hoory, O. Toledo-Ronen, A. Derreumaux, V. Manera, F. Verhey, P. Aalten, P. H. Robert, and et al., "Automatic speech analysis for the assessment of patients with predementia and alzheimer's disease," *Alzheimer's amp; Dementia: Diagnosis, Assessment amp; Disease Monitoring*, vol. 1, no. 1, p. 112–124, 2015.

[39] D. Beltrami, G. Gagliardi, R. Rossini Favretti, E. Ghidoni, F. Tamburini, and L. Calzà, "Speech analysis by natural language processing techniques: A possible tool for very early detection of cognitive decline?" *Frontiers in Aging Neuroscience*, vol. 10, 2018.

[40] J. Zhang, Z. Pan, C. Gui, T. Xue, Y. Lin, J. Zhu, and D. Cui, "Analysis on speech signal features of manic patients," *Journal of Psychiatric Research*, vol. 98, p. 59–63, 2018.

[41] B. Schuller, S. Steidl, A. Batliner, A. Vinciarelli, K. Scherer, F. Ringeval, M. Chetouani, F. Weninger, F. Eyben, E. Marchi, and et al., "The interspeech 2013 computational paralinguistics challenge: Social signals, conflict, emotion, autism," *Interspeech 2013*, 2013.

[42] V. Berisha, R. Utianski, and J. Liss, "Towards a clinical tool for automatic intelligibility assessment," *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, 2013.

[43] A. C. Arevian, D. Bone, N. Malandrakis, V. R. Martinez, K. B. Wells, D. J. Miklowitz, and S. Narayanan, "Clinical state tracking in serious mental illness through computational analysis of speech," *PLOS ONE*, vol. 15, no. 1, 2020.

[44] N. Barroso, K. Lopez-de Ipiña, H. Eguiraun Martinez, J. Solé-Casals, M. Ecay, A. Ezeiza, P. Martinez-Lage, and U. Lizardui, "Alzheimer disease diagnosis based on automatic spontaneous speech analysis," 06 2013.

[45] W. Jarrold, B. Peintner, D. Wilkins, D. Vergryi, C. Richey, M. L. Gorno-Tempini, and J. Ogar, "Aided diagnosis of dementia type through computer-based analysis of spontaneous speech," in *Proceedings of the Workshop on Computational Linguistics and Clinical Psychology: From Linguistic Signal to Clinical Reality*. Baltimore, Maryland, USA: Association for Computational Linguistics, Jun. 2014, pp. 27–37. [Online]. Available: https://aclanthology.org/W14-3204

[46] G. Stegmann, S. Hahn, J. Liss, J. Shefner, S. Rutkove, K. Kawabata, S. Bhandari, K. Shelton, C. Duncan, V. Berisha, and et al., "Repeatability of commonly used

speech and language features for clinical applications," *Digital Biomarkers*, vol. 4, no. 3, p. 109–122, 2020.

[47] P. Boersma, *Praat, a system for doing phonetics by computer*, Amsterdam, Netherlands, 2001. [Online]. Available: https://www.fon.hum.uva.nl/praat/

[48] N. Cummins, A. Baird, and B. W. Schuller, "Speech analysis for health: Current state-of-the-art and the increasing impact of deep learning," *Methods*, vol. 151, p. 41–54, 2018.

[49] T. Ching, D. S. Himmelstein, B. K. Beaulieu-Jones, A. A. Kalinin, B. T. Do, G. P. Way, E. Ferrero, P.-M. Agapow, M. Zietz, M. M. Hoffman, and et al., "Opportunities and obstacles for deep learning in biology and medicine," *Journal of The Royal Society Interface*, vol. 15, no. 141, p. 20170387, 2018.

[50] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. The MIT Press, 2017.

[51] Y. Wang, L. Neves, and F. Metze, "Audio-based multimedia event detection using deep recurrent neural networks," *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2016.

[52] L. Yang, D. Jiang, X. Xia, E. Pei, M. C. Oveneke, and H. Sahli, "Multimodal measurement of depression using deep learning models," in *Proceedings of the 7th Annual Workshop on Audio/Visual Emotion Challenge*, ser. AVEC '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 53–59. [Online]. Available: https://doi.org/10.1145/3133944.3133948

[53] L. Yang, H. Sahli, X. Xia, E. Pei, M. C. Oveneke, and D. Jiang, "Hybrid depression classification and estimation from audio video and text information," in *Proceedings of the 7th Annual Workshop on Audio/Visual Emotion Challenge*, ser. AVEC '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 45–51. [Online]. Available: https://doi.org/10.1145/3133944.3133950

[54] F. Eyben, K. R. Scherer, B. W. Schuller, J. Sundberg, E. André, C. Busso, L. Y. Devillers, J. Epps, P. Laukka, S. S. Narayanan, and K. P. Truong, "The geneva minimalistic acoustic parameter set (gemaps) for voice research and affective computing," *IEEE Transactions on Affective Computing*, vol. 7, no. 2, pp. 190–202, 2016.

[55] L. He and C. Cao, "Automated depression analysis using convolutional neural networks from speech," *Journal of Biomedical Informatics*, vol. 83, pp. 103–111, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S153204641830090X

[56] P. Yang, Y. Hwa Yang, B. B. Zhou, and A. Y. Zomaya, "A review of ensemble methods in bioinformatics," *Current Bioinformatics*, vol. 5, no. 4, p. 296–308, 2010.

[57] T. Warnita, N. Inoue, and K. Shinoda, "Detecting alzheimer's disease using gated convolutional neural network from audio data," *Interspeech 2018*, 2018.

[58] E. Rejaibi, A. Komaty, F. Meriaudeau, S. Agrebi, and A. Othmani, "Mfcc-based recurrent neural network for automatic clinical depression recognition and assessment from speech," *Biomedical Signal Processing and Control*, vol. 71, p. 103107, 2022.

[59] S. R. Livingstone and F. A. Russo, "The ryerson audio-visual database of emotional speech and song (ravdess): A dynamic, multimodal set of facial and vocal expressions in north american english," *PLOS ONE*, vol. 13, no. 5, 2018.

[60] J. Deng, N. Cummins, M. Schmitt, K. Qian, F. Ringeval, and B. Schuller, "Speech-based diagnosis of autism spectrum condition by generative adversarial network representations," *Proceedings of the 2017 International Conference on Digital Health*, 2017.

[61] C. Kim, D. Gowda, D. Lee, J. Kim, A. Kumar, S. Kim, A. Garg, and C. Han, "A review of on-device fully neural end-to-end automatic speech recognition algorithms," *2020 54th Asilomar Conference on Signals, Systems, and Computers*, 2020.

[62] S. Hershey, S. Chaudhuri, D. P. Ellis, J. F. Gemmeke, A. Jansen, R. C. Moore, M. Plakal, D. Platt, R. A. Saurous, B. Seybold, and et al., "Cnn architectures for large-scale audio classification," *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2017.

[63] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds., vol. 25. Curran Associates, Inc., 2012. [Online]. Available: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

[64] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014. [Online]. Available: https://arxiv.org/abs/1409.1556

[65] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," *CoRR*, vol. abs/1409.4842, 2014.

[66] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015.

[67] W. Lim, D. Jang, and T. Lee, "Speech emotion recognition using convolutional neural networks and recurrent neural networks with attention model," *Proceedings of 2019 the 9th International Workshop on Computer Science and Engineering*, 2019.

[68] P. Li, Y. Song, I. McLoughlin, W. Guo, and L. Dai, "An attention pooling based representation learning method for speech emotion recognition," *Interspeech 2018*, 2018.

[69] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *CoRR*, vol. abs/1706.03762, 2017.

[70] Y. Gong, Y.-A. Chung, and J. Glass, "Ast: Audio spectrogram transformer," *Interspeech 2021*, 2021.

[71] S. Tranter and D. Reynolds, "An overview of automatic speaker diarization systems," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 14, no. 5, pp. 1557–1565, 2006.

[72] T. J. Park, N. Kanda, D. Dimitriadis, K. J. Han, S. Watanabe, and S. Narayanan, "A review of speaker diarization: Recent advances with deep learning," 2021. [Online]. Available: https://arxiv.org/abs/2101.09624

[73] H. Bredin, R. Yin, J. M. Coria, G. Gelly, P. Korshunov, M. Lavechin, D. Fustes, H. Titeux, W. Bouaziz, and M.-P. Gill, "pyannote.audio: neural building blocks for speaker diarization," in *ICASSP 2020, IEEE International Conference on Acoustics, Speech, and Signal Processing*, Barcelona, Spain, May 2020.

[74] Joblib Development Team, "Joblib: running python functions as pipeline jobs." [Online]. Available: https://joblib.readthedocs.io/

[75] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[76] J. F. Gemmeke, D. P. W. Ellis, D. Freedman, A. Jansen, W. Lawrence, R. C. Moore, M. Plakal, and M. Ritter, "Audio set: An ontology and human-labeled dataset for audio events," in *Proc. IEEE ICASSP 2017*, New Orleans, LA, 2017.

[77] S. Hershey, S. Chaudhuri, D. P. W. Ellis, J. F. Gemmeke, A. Jansen, R. C. Moore, M. Plakal, D. Platt, R. A. Saurous, B. Seybold, M. Slaney, R. J. Weiss, and K. W. Wilson, "CNN architectures for large-scale audio classification," *CoRR*, vol. abs/1609.09430, 2016. [Online]. Available: http://arxiv.org/abs/1609.09430

[78] E. L. Lehmann and D. H. J. M., *Nonparametrics: Statistical methods based on ranks*. Springer, 2006.

Code

Listing A.1: diarization_from_json.py

```
import os
import json
import numpy as np
import librosa

from scipy.io.wavfile import write

from joblib import Parallel, delayed

SAMPLE_RATE = 16000

def crop_from_json(json_filename):
  json_directory = "/home/opc/files/diarization_jsons"
  patient_code = json_filename[:-5]

  with open(os.path.join(json_directory,json_filename)) as json_file:
    label_dict = json.load(json_file)


  labels_list = []
  for segment in label_dict['content']:
    label = segment['label']
    if label not in labels_list:
      labels_list.append(label)

  template_dict = {}
  for label in labels_list:
    template_dict[label] = []

  recording_path = "/home/opc/files/patient_files/" + patient_code + ".wav"

  ### load downsampled audio into memory, retrieve length of file
  waveform, _ = librosa.load(recording_path, sr=SAMPLE_RATE)
  num_samples = waveform.shape[0]

  ### save the start and stop times into a list for each of 2 speakers
      (WARNING, THIS ASSUMES 2 SPEAKERS)
  seg_times_dict = template_dict
```

```
  for obj in label_dict['content']:
    arr = (np.array([obj['segment']['start'],
           obj['segment']['end']])*SAMPLE_RATE).astype('uint')
    label = obj['label']
    seg_times_dict[label].append(arr)

# good up to here

  ### calculate number of samples attributed to each speaker
  num_samples_dict = {}

  for label in labels_list:
    num_samps = 0
    for time in seg_times_dict[label]:
      num_samps += int(time[1]-time[0])
    num_samples_dict[label] = num_samps

  #print('num_samples_dict \n',num_samples_dict)

  ### calculate the average power of each speaker, using above
  average_dict = {}

  for label in labels_list:
    sum = 0
    for time in seg_times_dict[label]:
      sum += np.sum(waveform[time[0]:time[1]]**2)
    average = sum/num_samples_dict[label]
    average_dict[label] = average

  #print('average dict \n', average_dict)

  cur_max = 0
  for label in average_dict:
    average = average_dict[label]
    if average > cur_max:
      max_label = label
      cur_max = average

  #print(max_label, cur_max)

  ### create array to become audio file
  new_audio = np.zeros(num_samples_dict[max_label])

  # indexes to use as reference to the audio that gets copied
  last_in_new = int(0)

  for time in seg_times_dict[max_label]:
    n_samples = int(time[1]-time[0])
    new_audio[int(last_in_new):int(last_in_new+n_samples)] =
        waveform[int(time[0]):int(time[1])]
    last_in_new += n_samples

  output_directory = "/home/opc/files/diarized_patient_files"
  output_filename = "Diarized " + patient_code + ".wav"

  write(os.path.join(output_directory,output_filename) , SAMPLE_RATE,
      new_audio)
```

```
  print('finished file:',patient_code,'found label:', max_label, 'with
      \#samps:',num_samples_dict[max_label])
  return

directory = "/home/opc/files/diarization_jsons"

json_file_list = []

for filename in os.listdir(directory):
  if filename.endswith(".json"):
    json_file_list.append(filename)

print(json_file_list)

Parallel(n_jobs=32)(delayed(crop_from_json)(json_file) for json_file in
    json_file_list)
```

Listing A.2: create_spectrogram.py

```
import os
import random
import numpy as np
import pandas as pd

import librosa
import librosa.display

from joblib import Parallel, delayed

SAMPLE_RATE = 16000
NUM_MELS = 224

df_train = pd.read_csv('/home/opc/files/trainset/train_labels.csv')
df_val = pd.read_csv('/home/opc/files/trainset/val_labels.csv')

def min_max_norm(arr):
  min = arr.min()
  max = arr.max()
  return (arr-min)/(max-min)

def proc_audio(wav,sr=SAMPLE_RATE,n_mels=224):
  S = librosa.feature.melspectrogram(wav,sr=sr,n_mels=n_mels)
  S_dB = librosa.power_to_db(S, ref=np.max)
  S_dB = min_max_norm(S_dB)
  return S_dB


# Create list of audio filenames for train and val set
train_list = []
val_list = []

for ind, row in df_train.iterrows():
  s = row['Image']
  s = s.split('_')[0]
  s = 'Diarized Patient ' + s + '.wav'
  pair = (s,row['Label'])
  if pair not in train_list:
    train_list.append(pair)

for ind, row in df_val.iterrows():
  s = row['Image']
  s = s.split('_')[0]
  s = 'Diarized Patient ' + s + '.wav'
  pair = (s, row['Label'])
  if pair not in val_list:
    val_list.append(pair)

audio_dir = '/home/opc/files/diarized_patient_files'
spectrogram_dir = '/home/opc/files/trainset/simple_spectrograms'


# Prepare for dataframe: patient_code, spect_index, filename, label
GLOBAL_ROWS_LIST = []

def para_cut(filename, label):
  patient_code = filename[17:-4]
```

```
  wav, sr = librosa.load(os.path.join(audio_dir,filename), sr = 16000)
  spect = proc_audio(wav)
  num_specs = spect.shape[1]//(NUM_MELS/2) - 1
  num_specs = int(num_specs)
  print('Generating',num_specs,'spectrograms from shape',spect.shape)

  for i in range(num_specs):
    new_filename = patient_code + '_' + str(i) + '.npy'
    start = 112*i
    end = start + 224
    segment = spect[:,start:end]

    np.save(os.path.join(spectrogram_dir,new_filename),segment)

    row_dict = {'patient_code': patient_code,\
                'spect_index': i,\
                'filename': new_filename,\
                'label': label}

    GLOBAL_ROWS_LIST.append(row_dict)

  return


Parallel(n_jobs=32,require='sharedmem') \
    (delayed(para_cut)(filename,label) for filename, label in train_list)

df = pd.DataFrame(GLOBAL_ROWS_LIST)
df.to_csv(os.path.join(spectrogram_dir,'trainset.csv'))
```

Listing A.3: train_cnn.py

```
import os
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras import backend as K

print(tf.__version__)

#tf.config.list_physical_devices('GPU')

df_train = pd.read_csv("/home/opc/files/trainset/train_labels.csv")
df_val = pd.read_csv("/home/opc/files/trainset/val_labels.csv")

train_samples = list()
val_samples = list()

dir_path = "/home/opc/files/trainset/short_spectrograms"

for index, row in df_train.iterrows():
    image_file = os.path.join(dir_path, row['Image'])
    target = str(row['Label'])
    train_samples.append((image_file, target))

for index, row in df_val.iterrows():
    image_file = os.path.join(dir_path, row['Image'])
    target = str(row['Label'])
    val_samples.append((image_file, target))


# load numpy arrays from filenames
def read_sample(data_path: str) -> tuple:
    path = data_path.numpy()
    image_path, target = path[0].decode('utf-8'), path[1]

    img = np.load(image_path)
    img = np.expand_dims(img,axis=2)
    tgt = int(target)

    # can add processing here if desired

    return (img, tgt)


# wrap read_sample into tensorflow function and set shape
@tf.function
def tf_read_sample(data_path: str) -> dict:
    print("INSIDE TF_READ_SAMPLE")
    # wrap custom dataloader in tensorflow
    [image, target] = tf.py_function(read_sample,
                    [data_path],
                    [tf.float32, tf.uint8] )

    image.set_shape((128,256,1))
    print(image.shape)
```

```
    return {'image': image, 'target': target}


# convert arrays to tensors with appropriate data types
@tf.function
def load_sample(sample: dict) -> tuple:
    # convert to tf image
    image = sample['image']
    tgt = sample['target']

    # cast to proper data types
    image = tf.cast(image, tf.float32)
    tgt = tf.cast(tgt, tf.uint8)

    return image, tgt


# create tf training dataset from file links
ds_train = tf.data.Dataset.from_tensor_slices(train_samples)
# read in image/target pairs
ds_train = ds_train.map(tf_read_sample,
    num_parallel_calls=tf.data.experimental.AUTOTUNE)
# read in as tensors
ds_train = ds_train.map(load_sample,
    num_parallel_calls=tf.data.experimental.AUTOTUNE)

# repeat for validation set
ds_val = tf.data.Dataset.from_tensor_slices(val_samples)
ds_val = ds_val.map(tf_read_sample,
    num_parallel_calls=tf.data.experimental.AUTOTUNE)
ds_val = ds_val.map(load_sample,
    num_parallel_calls=tf.data.experimental.AUTOTUNE)


# test if it works!
for image, target in ds_train.take(5):
    print("Image of type: ", type(image), "and shape", image.shape,
        tf.math.reduce_mean(image),
        tf.math.reduce_min(image),tf.math.reduce_max(image))
    print("Target of type: ", type(target), "and value", target)


# CREATE MODEL
inputs = layers.Input((128,256,1))
norm = layers.BatchNormalization()(inputs)
c1 = layers.Conv2D(16, (3,3) , activation='relu')(norm)
s1 = layers.MaxPooling2D()(c1)
c2 = layers.Conv2D(32, (3,3), activation='relu')(s1)
s2 = layers.MaxPooling2D()(c2)
c3 = layers.Conv2D(32, (3,3), activation='relu')(s2)
s3 = layers.MaxPooling2D()(c3)
c4 = layers.Conv2D(64, (3,3), activation='relu')(s3)
s4 = layers.MaxPooling2D()(c4)
c5 = layers.Conv2D(64, (3,3), activation='relu')(s4)
s5 = layers.MaxPooling2D()(c5)
fl = layers.Flatten()(s5)
d1 = layers.Dense(32, activation='relu')(fl)
d2 = layers.Dense(16, activation='relu')(d1)
out = layers.Dense(1, activation='sigmoid')(d2)
```

```
model = keras.Model(inputs=[inputs],outputs=[out])

print(model.summary())

def weightedLoss(originalLossFunc, weightsList):

    def lossFunc(true, pred):

        axis = -1 #if channels last
        #axis= 1 #if channels first


        classSelectors = True
        classSelectors = tf.cast(classSelectors, tf.int32)
        classSelectors = [K.equal(i, classSelectors) for i in
            range(len(weightsList))]
        classSelectors = [K.cast(x, K.floatx()) for x in classSelectors]

        #for each of the selections above, multiply their respective weight
        weights = [sel * w for sel,w in zip(classSelectors, weightsList)]

        #sums all the selections
        weightMultiplier = weights[0]
        for i in range(1, len(weights)):
            weightMultiplier = weightMultiplier + weights[i]


        #make sure your originalLossFunc only collapses the class axis
        #you need the other axes intact to multiply the weights tensor
        loss = originalLossFunc(true,pred)
        loss = loss * weightMultiplier

        return loss
    return lossFunc

optimizer = keras.optimizers.Adam()
list_metrics = [ 'acc', keras.metrics.Recall(), keras.metrics.AUC() ]

save_path = "/home/opc/files/models/baseline_cnn/log.csv"
history_logger=tf.keras.callbacks.CSVLogger(save_path, separator=",",
    append=True)

model.compile(optimizer=optimizer,
    loss=weightedLoss(keras.losses.BinaryCrossentropy(from_logits=False),{0:1,1:4}),metri
    run_eagerly=True)

results = model.fit(x=ds_train.batch(64), batch_size=64, epochs=100,
    callbacks=[history_logger], validation_data=ds_val.batch(64))

model.save("/home/opc/files/models/baseline_cnn/model")
```

Listing A.4: embeds_from_vggish.py

```
from vggish_smoke_test import *

import vggish_slim
import vggish_params
import vggish_input

import os
import json
import numpy as np
import librosa
import pandas as pd
from tqdm import tqdm

def CreateVGGishNetwork(hop_size=0.96): # Hop size is in seconds.
  """Define VGGish model, load the checkpoint, and return a dictionary
      that points
  to the different tensors defined by the model.
  """
  vggish_slim.define_vggish_slim()
  checkpoint_path = 'vggish_model.ckpt'
  vggish_params.EXAMPLE_HOP_SECONDS = hop_size
  vggish_slim.load_vggish_slim_checkpoint(sess, checkpoint_path)
  features_tensor = sess.graph.get_tensor_by_name(
      vggish_params.INPUT_TENSOR_NAME)
  embedding_tensor = sess.graph.get_tensor_by_name(
      vggish_params.OUTPUT_TENSOR_NAME)
  layers = {'conv1': 'vggish/conv1/Relu',
            'pool1': 'vggish/pool1/MaxPool',
            'conv2': 'vggish/conv2/Relu',
            'pool2': 'vggish/pool2/MaxPool',
            'conv3': 'vggish/conv3/conv3_2/Relu',
            'pool3': 'vggish/pool3/MaxPool',
            'conv4': 'vggish/conv4/conv4_2/Relu',
            'pool4': 'vggish/pool4/MaxPool',
            'fc1': 'vggish/fc1/fc1_2/Relu',
            #'fc2': 'vggish/fc2/Relu',
            'embedding': 'vggish/embedding',
            'features': 'vggish/input_features',
          }
  g = tf.get_default_graph()
  for k in layers:
    layers[k] = g.get_tensor_by_name( layers[k] + ':0')
  return {'features': features_tensor,
          'embedding': embedding_tensor,
          'layers': layers,
         }


def ProcessWithVGGish(vgg, x, sr):
  '''Run the VGGish model, starting with a sound (x) at sample rate
  (sr). Return a whitened version of the embeddings. Sound must be scaled
      to be
  floats between -1 and +1.'''
  # Produce a batch of log mel spectrogram examples.
  input_batch = vggish_input.waveform_to_examples(x, sr)
  # print('Log Mel Spectrogram example: ', input_batch[0])
  [embedding_batch] = sess.run([vgg['embedding']],
```

```
                              feed_dict={vgg['features']: input_batch})
  # Postprocess the results to produce whitened quantized embeddings.
  pca_params_path = 'vggish_pca_params.npz'
  pproc = vggish_postprocess.Postprocessor(pca_params_path)
  postprocessed_batch = pproc.postprocess(embedding_batch)
  # print('Postprocessed VGGish embedding: ', postprocessed_batch[0])
  return postprocessed_batch[0]


import tensorflow.compat.v1 as tf


tf.compat.v1.disable_eager_execution()
tf.reset_default_graph()
sess = tf.Session()

vgg = CreateVGGishNetwork()


def EmbeddingsFromVGGish(vgg, x, sr):
  '''Run the VGGish model, starting with a sound (x) at sample rate
  (sr). Return a dictionary of embeddings from the different layers
  of the model.'''
  # Produce a batch of log mel spectrogram examples.
  input_batch = vggish_input.waveform_to_examples(x, sr)
  # print('Log Mel Spectrogram example: ', input_batch[0])
  layer_names = vgg['layers'].keys()
  tensors = [vgg['layers'][k] for k in layer_names]
  results = sess.run(tensors,
                     feed_dict={vgg['features']: input_batch})
  resdict = {}
  for i, k in enumerate(layer_names):
    resdict[k] = results[i]
  return resdict

def MinMaxNormalize(array):
  min = array.min()
  max = array.max()
  return 2*(array-min)/(max-min) - 1

def embedding_from_file(filename, offset):
  num_samples = 16000
  SAMPLE_RATE = 16000
  audio_dir = '/home/opc/files/diarized_patient_files_v2'

  wave, _ = librosa.load(os.path.join(audio_dir,filename), sr=SAMPLE_RATE)
  wave = MinMaxNormalize(wave)
  segment = wave[offset*num_samples:(offset+1)*num_samples]
  #print('segment is from:',offset*num_samples,'to',(offset+1)*num_samples)

  return EmbeddingsFromVGGish(vgg,segment,num_samples)['embedding']

df = pd.read_csv('/home/opc/files/trainset/val_segments.csv')

big_array = np.zeros([len(df),128])

for ind, row in tqdm(df.iterrows()):
  filename = 'Diarized Patient ' + row['patient_code'] + '.wav'
  embedding = embedding_from_file(filename,row['audio_index'])
```

```
  df.loc[ind,'embedding'] = ind
  big_array[ind] = embedding

df.to_csv('/home/opc/files/trainset/val_segments.csv')
print('Shape of embed matrix: ', big_array.shape)
np.save('/home/opc/files/trainset/val_embeddings.npy',big_array)
```