Edge Device Speaker Verification

Thomas Duffy
Data Science & Engineering
City College of New York

December 19, 2020

Thesis Submitted in Partial Fulfillment of the

Requirements for the Degree of

Master of Science

in the

Program of Data Science and Engineering

Grove School of Engineering

Approved: _____
Dr. Jie Wei
Advisor

Approved: _____
Dr. Michael Grossberg
Co-Director

Approved: _____
Dr. Zhigang Zhu
Co-Director

Abstract

The continued shrinking of processors and other physical hardware in concert with development of embeddable machine learning frameworks has enabled new use cases placing machine learning directly in the "wild". The problem of speaker verification, for a long time, has been deployed to perform inference on systems with significant computations resources. More recently, these systems have been built for smaller, cheaper devices which can be placed in people's homes or other edge locations. Here, we aim to demonstrate that a reasonably accurate, generalizable, text-independent speaker verification system can be built, trained, and, ultimately, deployed onto a microcontroller with as a little as 1MB of flash memory. That is, a system which should be able to *enroll* new speakers onto the device in an online fashion. Previous research has demonstrated that online enrollment through use of a generalizable speaker verification model using speaker-specific embeddings is possible. Recent work has outlined embeddable systems which work on mobile phones and Internet-of-Things (IoT) devices which can be located in user's homes and other disparate locations with limited access to computational hardware. So far, however, the feasibilty of building such a system for a microcontroller has not been established. As mentioned, these systems have been successfully deployed on larger edge devices, here our aim to explore the possibility of doing so on a single microcontroller. We use a concatenation of the publically available LibriSpeech and VoxCeleb datasets to train several small, generalizable, speaker verification models. Models trained on this data include those implementing recurrent and convolutional neural network architectures. In order to deploy our inference system to a microcontroller, we re-produce a log-mel spectrogram framework implemented in Python to our target device supported language: `C++`. We show that it is possible to build a reasonably accurate, $EER \leq 11\%$, generalizable text-independent speaker verification model which will fit on even the smallest microcontroller. In conjuction with our log-mel spectrogram implementation in `C++`, it is possible to deploy this system in its entirety onto an Arduino Nano device with an on-board microphone for online speaker enrollment and inference. The field of edge device machine learning (TinyML) is an active area of research. Our contribution demonstrates the possibility of building systems which can perform inference on a form small microcontroller, accepting the trade-offs inherit in the problem.

Contents

1 Introduction

Speaker verification continues its move from large, complex, hand-crafted systems, to smaller, more accurate, on-device systems. Smaller form factors are becoming more widely used in the "wild", e.g. Google Home devices, to perform the task of speaker verification in low resource environments with incredible accuracy. Here, our aim is to contribute to the growing body of research by investigating the feasability for building a text-independent speaker verification system which can fit onto a microcontroller with an onboard microphone. Microcontrollers have extremely limited computation and space resources, but can be used in a low-power setting. Such a system would open new use cases for meaningful speaker verification on a chip the size of a U.S. quarter dollar coin.

Speaker verification is the process of receiving an audio sample of speech and determining whether the person uttering it is *known* to the system. In order to do this, the system must, itself, contain a representation of that speaker and then be able to discern at a level of confidence that this incoming utterance belongs to that speaker. Here, our formulation uses the concept of a *voiceprint* to "enroll" a speaker, or add them to the set of the system's known representations. We make use of deep neural networks here as they enable us to connect our inference task directly to the training of our system.

The problem of text-independent speaker verification is a decidedly harder one than text-dependent. In text-dependent applications, the system expects, and is trained for, a pre-defined utterance. For example, the wake word phrase on a Google Home device is "Hey, Google". Text-dependent systems are capable of higher accuracy, and require less data, because they can properly represent a speaker in a constrained, utterance-specific space. In text-independent applications, however, the utterance space is unconstrained. This requires the system to learn a *more* generalizable representation of a speaker. That said, text-independent systems have a wider range of applications than text-dependent ones.

The potential for a text-independent systems in an embeddable format would allow for use cases in biometric security and law enforcement. The form factor of such a device would allow it to be placed essentially anywhere, and provide its predictions both synchronously or asynchronously depending on the network capabilities of its location.

In Section 2, we will present an overview of the current state of the field of speaker verification and the various methods which have been used to perform this task in the past. Then, in Section 3, we will present the methodology behind our approach and its implementation. Finally, in Section 4, we will show our results and discuss them.

2 Literature Survey

Speaker verification systems have made significant progress over the last decade by employing deep neural network (DNN) architectures. A large amount of available speech data, both public and proprietary, has enabled researchers to use deeper model architectures which are able to learn from data directly [18]. In order to contextualize our intention of deploying a simliar, smaller, model to an edge device, we will discuss the current state of the speaker verification field.

2.1 Speaker Verification Overview

A speaker verification system aims to take some input utterance, u, e.g. an individual, s, saying "Hello", and, first, correctly verify the system knows the speaker, i.e. if $s \in S$ where S is the set of all speakers known to the system, or if $s \notin S$ then the system should be able to "enroll" them, i.e. $S_{i+1} = s \cup S_i$. This task can be expressed as asking for the probability $p_i = P(s = i; i \in S|u)$ for all speakers i known to the system. Then, we will consider the utterance $\underline{u}$ to be verified as an utterance from speaker i if $p_i > t$ for some threshold t.

In Section 2.2 we will discuss the biological processes used by humans to process audio and connect that to the task ahead. From there, in Section 2.3 and Section 2.4 we will outline how raw audio signals received by microphones are turned into digital signals are converted into features which serve as inputs to models. An overview of state-of-the-art model architectures and the high-dimensional speaker voiceprint representations they utilize will be presented in Section 2.5, and Section 2.6, respectively. Finally, we will briefly discuss *edge* devices, their computational limitations, and current use cases in Section 2.7.

2.2 Sound Theory

In order to properly formalize our speaker verification problem, it will serve us to first provide an understanding of the biological processes used by humans to perform the task. This understanding informs a significant portion of the specialized architectures which have been built to model these systems. Ultimately, our aim is to model the harmonic content of an utterance, as these are directly connected to the formants which contain rich, speaker-specific, information [4].

2.2.1 Speech

Speech begins as air is passed up through the lungs, beginning in the *trachea* and then moving into the *laryngeal* section, where the vocal folds are located. The opening through which air passes in the vocal folds has a triangular shape and its openness controls the amount of resistance placed on the air passing through it, which in turn modulates the sounds generated. Above the vocal folds, air moves through the *pharynx*, which constitutes the beginning of the *vocal tract*. Depending on the amount of air, and its tension, passing through the vocal folds, different parts of the pharynx will be used to produce different sounds. Continuing its journey, air then passes through the *epiglottis* which ends in the back of the tongue. Finally, after passing through the mouth and its palates, air encounters the teeth and lips, which play a critical role in articulation. Below, in Figure 1, is a cross-section of these components.

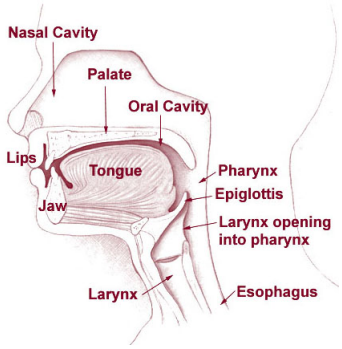


Figure 1: Speech tract

Information regarding the speaker-specific characteristics of these biological components is embedded within utterances [4].

2.2.2 Hearing

The auditory system can be broken down into two parts: the mechanical system responsible for intake of sounds via air pressure and the computational nervous system which processes the information contained within those sounds. The mechanical part is the ear, which we will discuss first. In Figure 2, below, we show a cross-section with relevant parts highlighted.

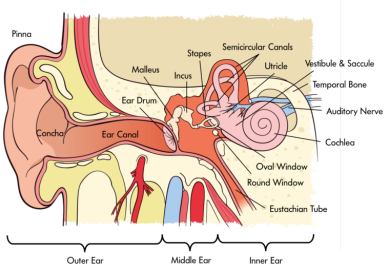


Figure 2: Auditory System (Ear)

First, the external ear is a combination of cartilages in the auricula and the ear canal. Next, the middle ear is where the *tympanic membrane* (ear drum) is located. In addition to the membrane, there are three bones which are responsible for transferring the motion of the ear drum induced by the sound waves amplified by the external ear: the *malleus* (hammer), *incus* (anvil), and *stapes* (stirrup). Those vibrations are then passed along to the inner ear via the *cochlear fenestra ovalis*. The inner ear is made up of the *cochlea*, a spiral cavity, and three semicircular canals: the *superior ampulla*, the *anterior ampulla*, and the *posterior ampulla*. The motion of the *stapes* bone in the middle ear induces pressure waves which excite the *cilia*, which are tiny hairs, inside of the *spiral tympani* within the *cochlea*. The *cilia* are connected to the auditory nerve bundle and are responsible for transmitting the resultant motion signal to the brain for cognition. It is important to note the spiral shape of the *spiral tympani* applies a semi-logarithmic transformation on sound, which will become important when considering feature representations of sound [4].

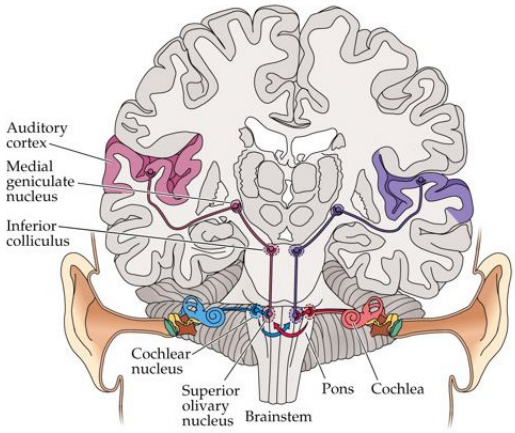


Figure 3: Auditory System (Brain)

The speech processing function of the brain, outlined above in Figure 3, is too complex for a full discussion here, but some relevant points must be mentioned. The information relayed by the cochlea is sent to the *primary auditory cortex*. The neurons in this section are organized *tonotopically* (according to tone), and map to the sensitivity of the cilia within the cochlea. So, different regions of primary auditory cortex become excited by specific ranges of frequencies.

The next processing step occurs in the *secondary auditory cortex*. It operates in both the right and left hemispheres of the brain and is the beginning of the specialized audio processing tasks contained in the different hemispheres. In the right hemisphere, it extracts harmonic, melodic and rhythmic patterns. In the left, it maps specific sounds to their phonetic elements.

Finally, information is passed to the *tertiary audio cortex*. It is responsible for turning the patterns generated by the previous cortex into higher level representations. In the right hemisphere, it responsible for extracting the musical discourse of the output from the secondary auditory cortex. Meanwhile, in the left hemisphere, it maps the phonetic elements deciphered upstream into lexical semantics.

Speaker recognition in humans is mostly performed in the right hemisphere of the brain, which suggests it makes use of information relating to tonality, rhythm, intonation and stress. However, in automatic speaker recognition, we tend to use the same features as those in speech recognition. These features are less prone to impersonation, for example when someone adapts their voice to mimic the pitch of another person [4].

2.3 Audio Processing

Audio in nature can be thought of as a continuous signal, as it varies over time which is itself continuous. In order to represent this physical phenomenon, we must *sample* it as specific intervals. This process is called *discretization*, i.e. the process of mapping a continuous signal to a discrete one. In this process, we must specify how frequently to sample the continuous signal in order to construct an accurate representation of it. The answer is provided by the Sampling Theroem which states that if a function $h(t)$ contains no frequencies greater than f_c cycles per second, it is completely determined by giving its ordinates at a series of points spaced $\frac{1}{2f_c}$ seconds apart. Typically, a sampling frequency of 16kHz is used for audio data [4].

Additionally, computers cannot represent arbitrarily precise measurements, so instead the measurements are *quantized*. For example, a 16-bit signal is restricted to the bounds $[-32,768, 32,767]$. Notice how this

impacts the storage requirements of representing a signal. Storing a 16-bit signal allows for more precision, but at the cost of space.

2.4 Input Features / Frontend

In order to form meaningful *representations* of speech which are directly connected to the task of speaker recognition, it *may* prove useful to perform transformations on the discrete, quantized, raw waveform representing the audio signal. Many widely used techniques attempt to mimic the biological processes described in Section 2.2.2. Recall, in Section 2.3, we stated the first input to any modeling task will be the quanitized raw waveform representation of a continuous sound. Simply, this can be thought of as a representation of the wave's *amplitude* as it varies in time. Below, in Figure 4 a waveplot denoting the normalized amplitude of the audio signal generated by one of the LibriSpeech speakers saying the phrase, "Is Papa Alone Enquired Miss Temple", is shown. The amplitude is normalized to be $[-1, 1]$.

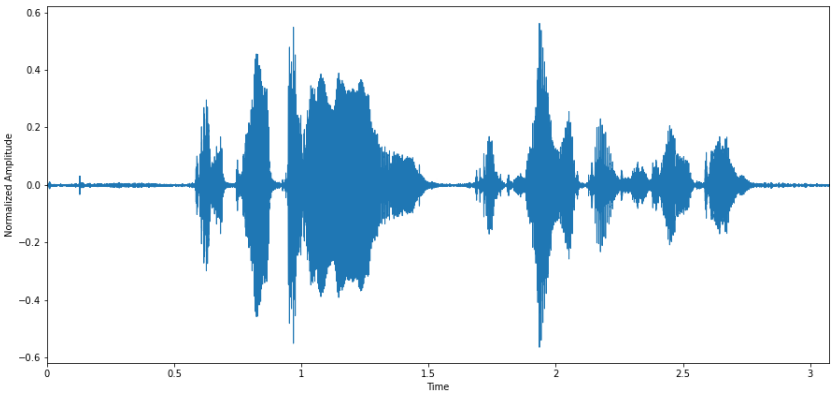


Figure 4: Example Raw Waveform from LibriSpeech

For a long time in speech research, hand-crafted features like Mel-frequency cepstral coefficients (MFCC) [10, 21, 32, 38, 41], Section 2.4.3, and Spectrograms [13, 19, 24, 30, 45, 47, 49], Section 2.4.2, were used as they closely mimic the process of human perception of audio information [4]. More recently, researchers have explored techniques to learn these filters directly from raw waveforms, Section 2.4.4, as part of the frontend of a neural network, in order to connect them *more directly* to the prediction task at hand [15, 20, 21, 30, 33, 41, 53, 54]. We will discuss each of these in turn.

2.4.1 Mel

Before introducing the typical features, we will introduce the mel scale, which is often used in conversion. The mel (abbrevation of *melody*) is a unit of pitch which is equal to one thousandth of the pitch ($\wp$) of a simple tone with frequency of 1000 Hz with an amplitude of 40dB above the auditory threshold. It is defined,

$$\wp = 2595 \log_{10} \left(1 + \frac{f}{700}\right) \tag{2.1}$$

This pitch scale more accurately represents the way humans internally represent audio signals, as discussed in Section 2.2.2. Below, in Figure 5, is a sample mel filter which is applied to audio in a process similar to that performed by the *spiral tympani*.

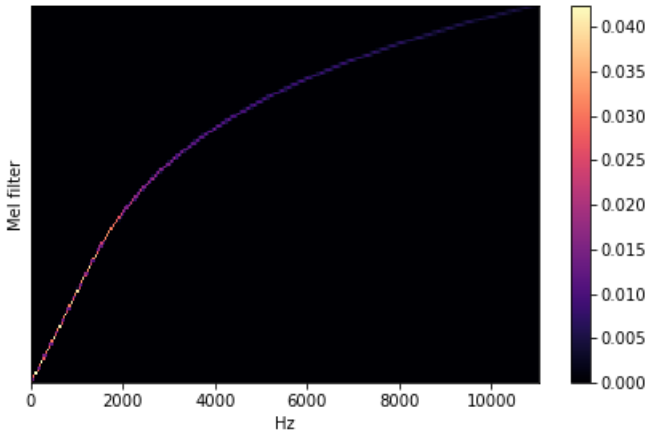


Figure 5: Mel filter bank

Notice its logarithmic nature. Additionally, it places a focus on the lower part of the spectrum, which tends to be where speaker specific characteristics are located [4].

2.4.2 Mel Spectrogram

A spectrogram is a two-dimensional representation of the spectral content of an audio sample. For each time step, the energy or power level of a particular frequency band is represented. Figure 6, shows this more clearly, as the highlighted sections over time represent the *formants*.

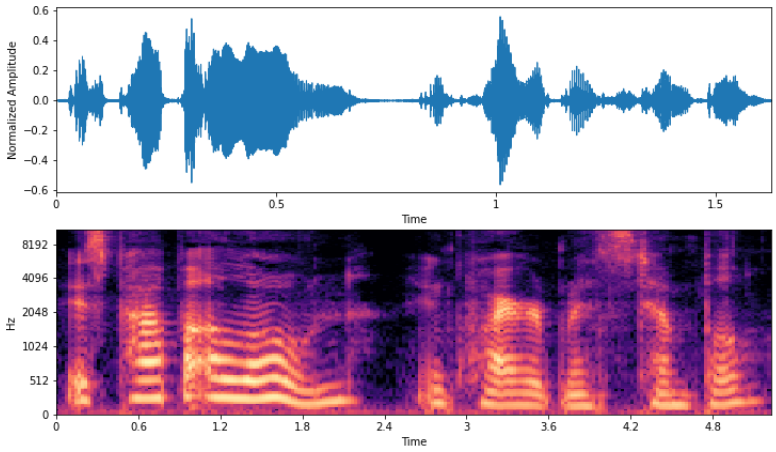


Figure 6: Example Mel Spectrogram

The formants, recall their importance outlined in Section 2.2.1, can be thought of as the resonant regions within a spectrogram (the more brightly colored areas above). These are useful in the domain of speaker recognition for speaker separation. Generally, the vocal tract length is inversely proportional to the height of the frequency range of a particular speaker [4]. As an example, the voice of an older man will "sound" lower than that of a younger boy and the formants of the two represented by spectrograms would show the younger boys formants present much higher on the frequency scale. The ease of interpretability along with its similarity to an image, means the spectrogram still enjoys widespread usage in state-of-the-art-systems [13, 19, 24, 30, 47, 49, 54].

2.4.3 Mel-frequency cepstral coefficients (MFCC)

The mel-frequency cepstrum is a representation of the short-term power spectrum of a sound which is derived from a linear cosine transformation of a log power spectrum on a nonlinear mel scale of frequency. It is related to the power Spectrogram of Section 2.4.2, in that it can be derived by applying a discrete cosine transformation to a periodogram. In the mel-frequency cepstrum, as previously stated in Section 2.4.1, the frequency bands are equally spaced on the mel scale, which better approximates the way human beings process audio information. Below, in Figure 7, are the MFCCs for the audio sample shown previously in Figure 4,

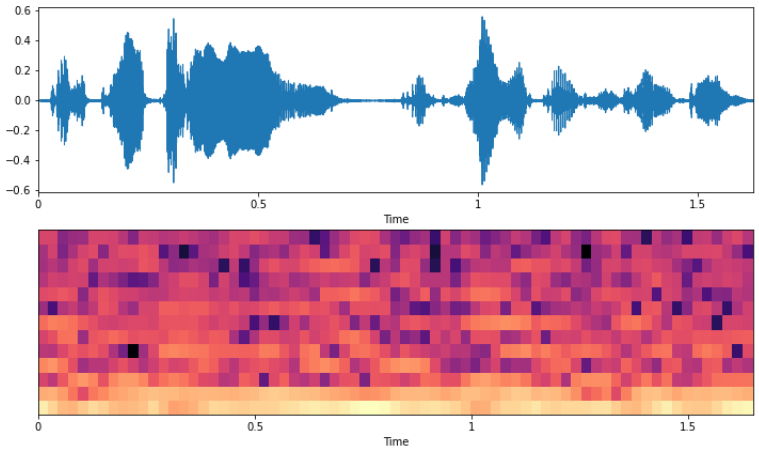


Figure 7: Example Mel-frequency cepstral coefficients

It continues to be used widely as an input feature for DNN models [10, 21, 32, 38, 41].

2.4.4 Raw Waveform

More recent research has explored the use of *learnable* convolutional filters at the frontend of a neural network directly on raw waveforms [5, 9, 15, 33, 35, 52, 53, 54]. By relying on the convolutional layers to provide a meaningful representation, researchers have posited it shoule be possible to *learn* filters that retain a high amount of signal information which directly relates to the prediction task at hand as they are simultaneously trained with the rest of the network [33].

Zegidour *et al* (2016), performed a comprehensive study on replacing the speech feature frontend with different architectures attempting to *learn* the modeling of the raw waveform. Their research was primarily focused on methods that would serve as a direct replacement of mel-filterbanks, discussed in Section 2.4.2. They present results for gammatone-based and scattering-based filters [53]. Additionally, some have argued the features discussed in Sections 2.4.2 and 2.4.3 smooth the speech spectrum which might limit the amount of information extracted from speaker specific attributes like *pitch* and *formants* [33].

Sainath *et al* (2015) similarly attempted to replace log-mel filterbank energies by using a time-delay convolution layer at the frontend. They showed that the time convolution layer succeeded in reducing temporal variations and preserved signal locality [37].

Conversely, some have criticized current raw waveform implementations as being over-parametrized and, thus, difficult to make use of in embedded environments [9].

2.5 Model Architectures

Once the incoming audio has been represented using the techniques discussed in Section 2.4, we seek to learn a function f connecting a particular utterance u to a specific speaker s. Various deep neural network

architectures have been tested in performing this task, and will be presented here.

2.5.1 Time-Delay Neural Networks (TDNN)

As discussed in Section 2.2.2, humans recognize speaker-specific information as a signal varies over time. With this understanding, Waibel *et al* (1989) proposed using a time-delay as an input layer to their network architecture, which introduces N delays $D_{0...N}$ such that every input feature is applied the weights of every D_i along with not applying a delay [48]. This allows the neural network to connect events in time (e.g. how the formant of a particular speaker varies over time). Recent research continues to make use of this technique [21, 30, 40, 45].

2.5.2 Convolutional Long Short-Term Memory Deep Neural Network (CLDNN)

As mentioned in Section 2.4.4, some have explored feeding raw waveforms directly into the DNN in the hopes of learning the feature representation alongside the classification task. Sainath *et al* (2015) proposed using convolutional layers as a time-delay to represent local signal dependencies by passing in the raw waveform and building features which approximate log-mel filterbanks [37].

2.5.3 Generalized End-to-End Loss

We can formulate the text-independent speaker recognition problem as an attempt to learn a *voiceprint* representation, which will be discussed at length in Section 2.6. With this formulation, the main concern is finding a latent embedding representation of a particular speaker and then ascribing a given utterance's embedding to a known speaker's embedding. Wan *et al* (2019) developed a generalizable system that directly connects the separation of speakers in that latent space to the task at hand using the cosine similarity of given speaker centroids to maximize their distance in the latent space [49]. Similarly, Ren, Chen and Xu (2019) attempted to learn representations by using PLDA [35].

2.5.4 ResNet

The networks implemented based on the methodology described in Sections 2.5.1 and 2.5.2 contain numerous (sometimes many) layers. He, Zhang, Ren, and Sun (2015) found that very deep networks in the image domain suffered from a *degradation* problem, where a deeper network's training error actually exceeds the error of a more shallow network of a similar architecture. This would appear to be unlikely given that a deeper model *should* have a training error at least lower than a shallower one assuming that we could construct it by concatenating the shallower network with an arbitray number of identity mappings. Instead, they proposed fitting a mapping $\mathcal{F}(x) := \mathcal{H}(x) - x$ which includes these identity mappings, essentially skipping layers [11]. Researchers have attempted to apply these learnings to the audio and speech domain [15, 19].

2.6 Embeddings / Backend

In Section 2.5.3 we presented a particular formalization of the speaker recognition problem which aims to learn an accurate *latent* representation of speaker specific information. Generally, these *embedding vectors* represent the *voiceprint* of a particular speaker. For a long time, Gaussian mixture models (GMM) constituted the state of the art of speaker identification systems, as they were capable of representing speaker-specific spectral features. However, the use of DNNs allows researchers to optimize their models directly in relation to the classification task at hand [34].

2.6.1 *i*-Vector

Dehak et al (2010) attempted to find a more robust speaker-specific representation by leveraging the ability of Support Vector Machines (SVM) in their work where they introduced the "identity" vector, or *i*-vector. Their aim was to find a low dimensional space (total variability space) which reduced the dimensionality of the speaker identification problem from the high dimensional supervector spaces used in previous GMMs [8].

More recent research into the domain of DNNs have argued their improvements directly connect the vectorized representation of an utterance to the classification task at hand, that of speaker identification [38]. Additionally, their ability to handle noisy environments and focus on the important information within a signal has also been called into question [40].

2.6.2 *d*-Vector

The advent of deep neural networks led researchers to explore systems that could potentially replace the *i*-vector systems of old and learn speaker embedding information. Variani *et al* (2014) developed a model which considered the final *hidden* layer the voiceprint of the speaker given a certain utterance. They called these "deep" vectors, or *d*-vectors. This development allowed the embedding to be directly related to the classification (separation) task at hand. The final, output, layer in training was a simple softmax layer. In order to perform enrollment, an average of several embedding vectors from input utterances was taken [47].

Wan, Wang, Papir, and Moreno (2019) extended this with their work at Google, using the enrollment phrase "Ok Google". They developed a cosine similarity matrix loss that attempted to learn *centroids* of a given speaker's utterances. This loss function would, then, learn the representations which best separated speakers within the embedding layer, which is exactly the task at hand during inference [49]. This architecture was the motivating one for our work here.

2.6.3 *X*-Vector

Snyder, Garcia-Romero, Povey, and Khudanpur (2017) also attempted to improve upon the *i*-vector systems by way of deep neural networks. Their approach bears similarities to the *d*-vector discussed in Section 2.6.2, however, they made use of probabilistic linear discriminant analysis (PLDA) to separate the embedding layer at the backend of the network. They make use of statistical pooling which allows the network to be trained, and make predictions, on variable length input sequences as it does not learn *frame*-level embeddings, but rather *segment*-level ones. They argue that this makes the model more generalizable to utterances not seen in training [43, 44]. Further research has been done to improve this method, e.g. by utilizing a Gaussian noise constrained network or multi-level statistical pooling from both the TDNN and LSTM layers [10, 45].

2.6.4 *H*−Vector

Shi, Huang, and Hain (2019) explored an improvement on the *d*-vector discussed in Section 2.6.2 and *X*-vector discussed in Section 2.6.3 by using a hierarchical attention model which is applied at both the *frame*-level and *segment*-level parts of the network. This approach treats the incoming utterance as a document, meaning that we should only concern ourselves with the relevant parts which aid in separating our speakers [41]. The hope is the model will concern itself only with the relevant speaker information embedded in an utterance, as opposed to all the information available in an utterance, as discussed at length in Section 2.2.2.

2.7 Edge Devices

Edge, and internet-of-things (IoT), devices continue to profilerate, such that by the year 2025 it is estimated more than 75 billion devices will be connected to the internet. Many of these devices contain sensors to continuously measure physical phenomena (temperature, noise, humidity, etc.) and have numerous applications. At the same time, these devices have limited compute resources while internet connections are difficult to maintain, hence the aim to perform as much of the computing on the device as possible [25]. This hardware proliferation has percepitated an increase in machine learning software frameworks targeting these devices [3, 7].

Improvements in speaker verification systems has coincided with their use in edge devices. Many of the previously discussed model architectures have allowed industry to deploy useful models on mobile phones, IoT devices, and microcontrollers. For instance, Google has embedded speaker verification into the Google Assistant application it ships with its phone along with placing it on their Google Home device [49]. Similarly, Amazon's Alexa home speaker performs on-device speaker verification [16, 39]. Apple's Siri voice assistant performs this function using an on-device model which ships with all iPhones [42]. Demos have recently been shared, e.g. at Tensorflow Dev Summit, demonstrating the ability of these models to perform useful functions on microcontrollers with as little as 32kB of on-board dynamic RAM [7].

2.7.1 Mobile Phone

McGraw *et al* (2016) describe the process of iteratively shrinking a baseline server-side model based on an LSTM architecture similar to that described in Section 2.5.3 such that it fits on a Nexus Android smartphone with 2 GB of RAM. The baseline LSTM model contains 3 layers each with 850 cells, totaling 20.1 million parameters. In order to shrink the model, LSTM projection layers are used. Further, the model is quantized such that during inference, 8-bit integers are used in place of 32-bit floating point numbers except in the activation functions and the network's output. The smallest version of the model is reduced to 3MB and represented by 3M parameters [23]. The usage of parameter quantization from 32-bit floating point to 8-bit integers has been confirmed in further research [12].

2.7.2 Microcontroller (Arduino)

The constraints of microcontrollers, in comparison, are significantly more stringent. These devices can come in many form factors, all with limited computational resources. The Tensorflow team built support for 130 of the 1,400 operations supported by the framework by extending the TensorflowLite project into Tensorflow Micro. This framework makes no assumptions of being able to dynamically allocate memory, instead using an arena model for temporary computational needs. It supports many devices using the ARM Cortex-M processor architecture, which includes our target Arduino Nano device, and has also been tested on the ESP32 architecture. They were able to demonstrate $\sim 4\%$ interpreter overhead deploying a keyword detection model [7]. Though the field is still relatively young, there has already been some successful research in using microcontrollers for audio-based applications.

Ahmed *et al* (2020) built a noise pollution monitoring system using an Arduino Nano which perform the Fast Fourier transform on-device. The processed signal was then passed to a Wi-Fi connected NodeMCU (EPS 8266) chip which was responsible for passing it along to a Google Firebase database in the cloud, where stream processing was performed [1].

Esling *et al* (2020) extended the *lottery ticket hypothesis*, which claims that deep models are often overparametrized and can be extensively pruned, to the realm of deep generative audio models. They used a process of structured trimming to train much smaller models than the parent counterparts which

performed at a similar, or higher, accuracy. They discussed the ability for these models to be embedded on Raspberry Pi and Arduino devices, noting that they would require even further pruning which would result in at least a 2.5x increase in the error rate [9].

Wong, Famouri, Pavlova and, Surana (2020) approached the problem from a slightly different angle. They introduced the concept of *attention condensers*, a self-attention mechanism which outputs a minimal embedding describing joint local and cross-channel activation relationships. This allowed them to implement several models, called TinySpeech, which had fewer than 11,000 parameters in total and were smaller than 50 kB in size. They reported favorable result using the Google Speech Commands dataset, a dataset used for low-vocabulary speech recognition [50].

3 Methodology

Here, we will detail the design and implementation of our model along with its deployment onto our edge device. First, we will describe the input datasets and features the model is trained on in Section 3.1. Then, we will outline the various model architectures in Section 3.2. Finally, we will discuss how to serialize and prepare it for deployment on a microcontroller in Section 3.3.

3.1 Dataset

We utilize a concatenated dataset derived from LibriSpeech, discussed in Section 3.1.1, VoxCeleb1, discussed in Section 3.1.2, and CommonVoice, discussed in Section 3.1.3, with the aim of including a sufficient number of *unique* speakers such that a generalizable speaker voiceprint is learnable. Overall, our dataset contains 3,217,792 utterance samples which are 1.2 seconds in length from 9,580 unique speakers.

In order to form our dataset we must define the number of speakers per batch, N, and the number of utterances per speaker within a batch, M. Then, a batch is of size NM. In our process, we set $N = M = 8$, such that a batch contains 64 samples. This process allows us to train more efficiently than the widely used triplet loss by considering NM utterances within a single batch [35, 49].

3.1.1 LibriSpeech

The LibriSpeech corpus contains approximately 1,000 hours of speech sampled at 16kHz. The dataset contains recordings of speakers reading from public domain texts, primarily from Project Gutenberg. Each audio segment is no longer than 35 seconds in length. The samples were split from larger segments by partitioning on any silence which exceeded 0.3 seconds [29]. Summary statistics for the 3 subsets of LibriSpeech used in training are presented below in Table 1.

Table 1: LibriSpeech Dataset Summary (Training)

Subset	**Length (hrs)**	**Length / speaker (min)**	n **speakers**
train-clean-100	100.6	25	251
train-clean-360	363.6	25	921
train-other-500	496.7	30	1,166
total	**960.9**	**27**	**2,368**

LibriSpeech provides testing datasets, with the prefix `test`, which were used in evaluating model performance. In Table 2, below, we show the summary statistics for these datasets.

Table 2: LibriSpeech Dataset Summary (Out-of-Sample)

Subset	**Length (hrs)**	**Length / speaker (min)**	n **speakers**
test-clean	5.4	8	40
test-other	5.1	10	33
total	**10.5**	**8.4**	**77**

3.1.2 VoxCeleb1

The VoxCeleb datasets are ingested from recorded interviews with celebrities posted on YouTube. This dataset focuses on real world speaking conditions in order to aid in building more robust, generalizable, speech and speaker models. To that end, videos within the dataset were recorded in many different environments, e.g. outdoor stadiums, red carpet interviews, studio interviews, and speeches [26, 27]. Below, in Table 3, we provide summary statistics for the VoxCeleb1 dataset.

Table 3: VoxCeleb1 Dataset Summary

Subset	**Length (hrs)**	**Length / speaker (min)**	n **speakers**
VoxCeleb1	350.5	16.8	1,251

3.1.3 CommonVoice

The CommonVoice dataset is an open source, community driven, dataset managed by the non-profit Mozilla Corp. It collates user submitted audio samples, recorded via the browser, which are then validated by volunteers [2]. Below, in Table 4 we present summary statistics for the train split of this dataset.

Table 4: CommonVoice Dataset Summary (Training)

Subset	**Length (hrs)**	**Length / speaker (min)**	n **speakers**
train	142.5	1.43	5,958

3.1.4 VCTK

The VCTK dataset was collected by researchers at the Centre for Speech Technology Research at the University of Edinburgh. It includes utterances from 109 English speakers with varying accents who, each, read aloud 400 sentences, selected from the *Herald Glasgow* newspaper [51]. Importantly, the accents in this dataset are mostly British as opposed to the others which are American-focused. Here, we have used the entire dataset as a validation set. Its summary statistics can be found in Table 5.

Table 5: VCTK Dataset Summary (Out-of-Sample)

Subset	**Length (hrs)**	**Length / speaker (min)**	n **speakers**
test	44.0	24.2	109

3.1.5 Features

Though some researchers have begun to explore feeding raw waveform samples *directly* into models, as discussed in Section 2.4.4, those approaches typically require even more data than using more typical, spectral-based, features [15, 33, 37, 54]. Here, instead, we use log filter banks, as presented in Section 2.4.2.

First, we split each utterance of variable length into 1.2 second long samples, using a voice activity detector to discard samples which contain no speech. In order to perform further transformations, we'll assume the signal is *stationary* over small steps. So, we frame the original 1.2 second sample into 25ms frames, sliding the window 10ms at every step. The 25ms frames are then passed through a Hann window function, defined below, to avoid spectral leakage.

$$w[n] = \sin^2\left(\frac{\pi n}{N}\right) \tag{3.1}$$

where N is the length of the window in samples (in this case 25ms * 16kHz = 400). We then perform an N-point FFT on each frame to convert our frames into the frequency domain. Here, we've used 512 $NFFT$. Then, we take the power of each frame f by applying $|f|^2$, yielding a periodogram. Note this leaves us with only the real part of the Fourier transform. Given that for real-valued signals, the Fourier transform is Hermitian symmetric, we end up with a frame length of 257, or $NFFT/2 + 1$.

In parallel, we generate 40 triangular mel filters on a mel-scale, which recall as discussed in Section 2.4.1 mimics the human hearing process. Multiplying the periodogram with these filter banks results in our final feature: the spectrogram of the signal's power spectrum. Examples from a particular training set speaker are shown below in Figure 8.

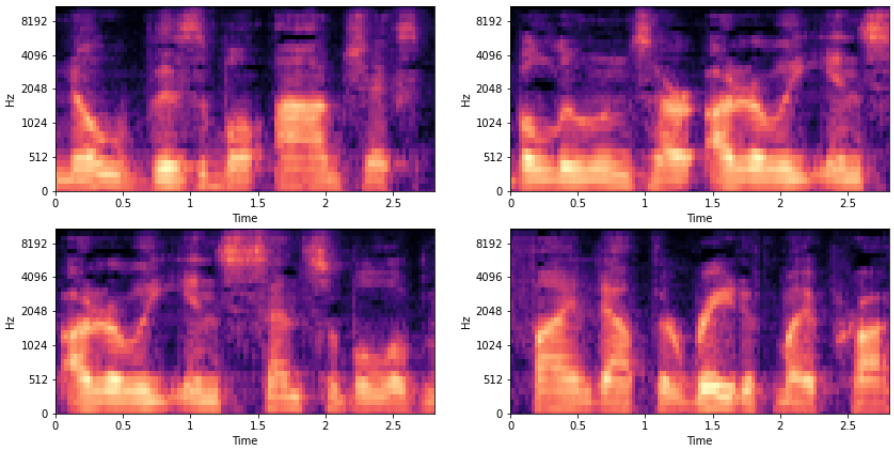


Figure 8: Spectrogram Features

3.2 Model

In order to represent the *signature* of a particular speaker, we want to generate a fixed-length, speaker-specific, embedding in some high dimensional space, motivated by the approaches discussed in Section 2.6. More formally, our aim is to learn a function f to map some input utterance $\mathbf{u}$ to an embedding $\mathbf{e}$.

$$f : \mathbf{u} \mapsto \mathbf{e}$$

Then given $\mathbf{e}$ we aim to known which *enrolled* (known) speaker $s \in S$, where all speakers are identified by a centroid embedding $\mathbf{c}_i$, the utterance should be assigned to. Or,

$$\begin{aligned} &\arg_i \max_P P\left(s = s_i \in S|\mathbf{u}\right) \\ &\arg_i \max\left(\cos\left(\mathbf{c}_i, \mathbf{e}\right)\right) \end{aligned} \tag{3.2}$$

for all enrolled speakers s_i. We want to assign the utterance $\mathbf{u}$ to the speaker which maximizes the probability that s_i said $\mathbf{u}$. In our formalization, motivated by the framework proposed in Section 2.6.2, we use cosine similarity to denote the probability that $\mathbf{u}$ belongs to a particular speaker s, namely their centroid $\mathbf{c}$. Where we define cosine similarity for two vectors $\mathbf{v}_1$ and $\mathbf{v}_2$,

$$\cos(\mathbf{v}_1, \mathbf{v}_2) = \frac{\mathbf{v}_1 \cdot \mathbf{v}_2}{\|\mathbf{v}_1\|\|\mathbf{v}_2\|}$$

Our inference task is thusly, given an utterance $\mathbf{u}$ which we have mapped to an embedding $\mathbf{e}$ find the known speaker centroid $\mathbf{c}_s$ embedding vector which has the highest cosine similiarity with $\mathbf{e}$. The benefit of this formalization is that we can express our training task directly in terms of our inference task. As discussed at length in Section 2.5, this is one major benefit of using DNNs for the task of speaker recognition.

3.2.1 Loss

As stated, we can connect our training task to our inference task directly. Specifically, we achieve this by selecting M utterances from N speakers within a batch, leaving us with a batch size of NM. Every one of the NM utterances is passed through our model and a resultant embedding $\mathbf{e}_i$ is generated. Then the centroid $\mathbf{c}_s$ of embeddings $\mathbf{e}_{s1}...\mathbf{e}_{sM}$ represents the voiceprint of speaker s for the M utterances in a batch, and is defined

$$\begin{aligned} \mathbf{c}_s &= \mathbb{E}_m[\mathbf{e}_{s,m}] \\ &= \frac{1}{M}\sum_{m=1}^{M} \mathbf{e}_{s,m} \end{aligned} \tag{3.3}$$

Our goal is to maximize the similarity of each $\mathbf{e}_{s,m}$ to its speaker's centroid $\mathbf{c}_s$ and minimize its similarity to all other centroids. Thus we draw a cosine similarity matrix where

$$\mathbf{S}_{ji,s} = w \cdot \cos(\mathbf{e}_{ji},\, \mathbf{c}_s) + b \tag{3.4}$$

where w and b are both learnable parameters. Visually,

Table 6: Batch Cosine Similarity Matrix

	$\mathbf{c}_1$	$\mathbf{c}_2$	$\mathbf{c}_3$	$\mathbf{c}_4$
$\mathbf{e}_{1,1}$				
$\mathbf{e}_{1,2}$				
$\mathbf{e}_{2,1}$				
$\mathbf{e}_{2,2}$				
$\mathbf{e}_{3,1}$				
$\mathbf{e}_{3,2}$				
$\mathbf{e}_{4,1}$				
$\mathbf{e}_{4,2}$				

Again, our goal here is to maximize similarity from a speaker's embeddings to its centroids and conversely minimize the similarity of an embedding to the centroids of the other speakers.

For text-independent speaker identification, soft max loss has been found to perform well [49]. Recall soft max loss is defined, for some function f,

$$L_i = -f_{y_i} + \log \sum_j \exp(f_j)$$

here, our model f outputs a cosine similarity matrix $\mathbf{S}$, as defined by Equation 3.4, so we can re-write the loss for each embedding $\mathbf{e}_{ji}$ as,

$$L(\mathbf{e}_{ji}) = -\mathbf{S}_{ji,j} + \log \sum_{k=1}^{N} \exp(\mathbf{S}_{ji,k}) \tag{3.5}$$

And the total loss over $\mathbf{S}$ for each batch,

$$\begin{aligned} L_G(\mathbf{x}; \mathbf{w}) &= L_G(\mathbf{S}) \\ &= \sum_{j,i} L(\mathbf{e}_{ji}) \end{aligned} \tag{3.6}$$

3.2.2 Architecture

Previous researchers have achieved successful results using time-delay networks, so here we include both Long short- term memory (LSTM) and gated recurrent units (GRU) [20, 21, 30, 40, 45, 48]. Samples time-delay architectures, which we will detail below, are show here in Figure 9,

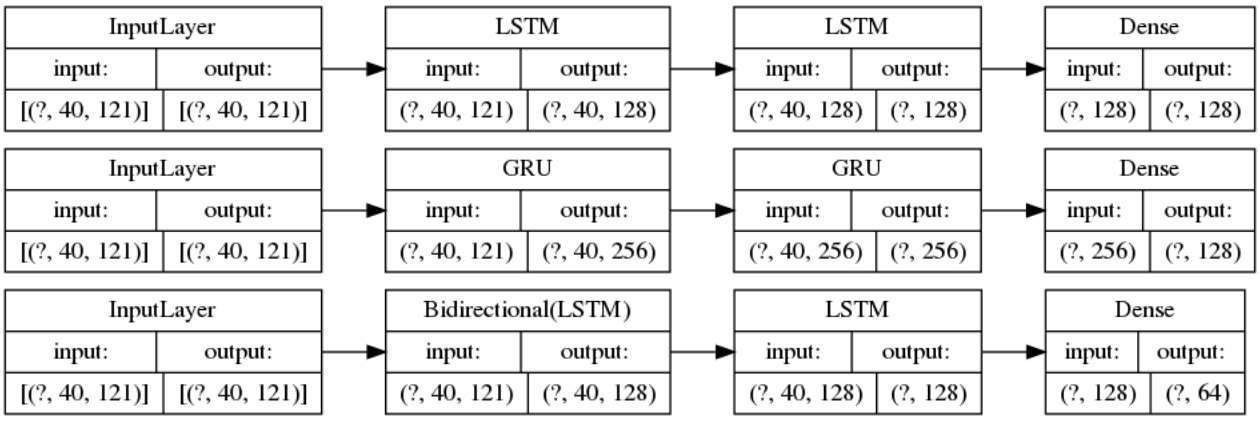


Figure 9: RNN Model Architectures

Others have attempted using Convolutional filters, which we detail as well [5, 33, 38]. These models have the added benefit that more space optimizations exist for serializing them into TensorflowLite models.

The physical constraints of our target device significantly limit the size of model that we can use. The Arduino Nano 33 BLE Sense has 1MB of flash memory where the FlatBuffer of our model is stored, along with the remainder of the program. So, here we present multiple architectures, listed by increasing number of parameters. The larger ones represent target models and potentially would not fit on the device. More details regarding their storage requirements is outlined in Section 4.1.1.

Table 7: Average Pooling Convolutional Model Architecture [1]

Layer	**Input Shape**	**Output Shape**	**# Params**
Convolution 1D	(40, 121)	(31, 8)	9,688
Convolution 1D	(31, 8)	(29, 8)	200
Average Pooling	(29, 8)	(29, 2)	-
Fully Connected	(58,)	(32,)	1,888
total			**11,776**

Table 8: Convolutional Model Architecture [2]

Layer	**Input Shape**	**Output Shape**	**# Params**
Convolution 1D	(40, 121)	(31, 8)	9,688
Convolution 1D	(31, 8)	(29, 8)	200
Fully Connected	(232,)	(64,)	14,912
total			**24,800**

Table 9: Convolutional Model Architecture [3]

Layer	**Input Shape**	**Output Shape**	**# Params**
Convolution 1D	(40, 121)	(31, 16)	19,376
Convolution 1D	(31, 16)	(29, 16)	784
Fully Connected	(464,)	(64,)	29,760
total			**49,920**

Table 10: Bidirectional LSTM Model Architecture [3]

Layer	**Input Shape**	**Output Shape**	**# Params**
Bidirectional LSTM	(40, 121)	(40, 128)	95,232
LSTM	(40, 128)	(128,)	131,584
Fully Connected	(128,)	(64,)	8,256
total			**235,072**

Table 11: LSTM Model Architecture [4]

Layer	**Input Shape**	**Output Shape**	**# Params**
LSTM	(40, 121)	(40, 128)	128,000
LSTM	(40, 128)	(128,)	131,584
Fully Connected	(128,)	(128,)	16,512
total			**276,096**

Table 12: GRU Model Architecture [5]

Layer	**Input Shape**	**Output Shape**	**# Params**
GRU	(40, 121)	(40, 256)	291,072
GRU	(40, 256)	(256,)	394,752
Fully Connected	(256,)	(128,)	32,896
total			**718,720**

3.2.3 Training

We train all the above models over 50 epochs using stochastic gradient descent with an initial learning rate of 0.01. The learning rate decays exponentially after the 25th epoch. We apply dropout of 10% at every layer of the network. The gradients are norm clipped at 3.0.

3.3 Edge Device

Ultimately, our trained model should be deployable onto a small form factor edge device. Here, the target device for our Tensorflow Micro models is an Arduino Nano 33 BLE Sense, which was specifically designed for TinyML applications. It has a number of onboard sensors, including a microphone.

3.3.1 Hardware

The board is based on the nRF52840 microcontroller which has 1MB of flash memory, where the program is stored, along with 256 kB of sRAM, where the variables created by the program are stored [28]. It has an ARM-Cortex M4 32-bit processor which is a supported processor for Tensorflow Micro [7].

3.3.2 TensorflowLite Micro

The Tensorflow Lite library was previously extended to target microcontroller devices [7]. It supports a subset of all available Tensorflow Operations, which are the abstraction used in Tensorflow to represent a node in the computational graph it draws. Here, we have only made use of supported ones, hence the serialization of our model into a TFLite micro model is possible.

4 Results

As stated in Section 3.2, our aim is learn a function f which can map an utterance $\mathbf{u}$ to a particular, enrolled, speaker s via their centroid $\mathbf{c}_s$. We expressed the assignment of utterance $\mathbf{u}$ to speaker s via the cosine similarity between $\mathbf{u}$ and $\mathbf{c}_s$. Here, we will assess the claim that such a function is learnable, and how well it achieves the stated task.

4.1 Model Performance

Given implementations of the models discussed in Section 3.2.2, we now present their respective performance. Performance was evaulated on the `test` sets provided by LibriSpeech, discussed in Section 3.1.1 and the entirety of the VCTK dataset, discussed in Section 3.1.4. We investigate performance using the *equal error rate* metric, in Section 4.1.1, the F_1 measure and its components in Section 4.1.2, and then consider the embedding space directly, in Sections 4.1.3, 4.1.4, and 4.1.5.

4.1.1 Equal Error Rate

In the domain of biometrics, the *equal error rate*, is used to measure the performance of a verification system [17, 46]. It is the level at which, for some confidence threshold t, the false positive (FP) and false negative (FN) rates are roughly equal.

$$EER = \frac{FP + FN}{P + N}$$

That is, we want to balance the trade off between falsely verifying impostor speakers (and their utterances) and accidentally rejecting legitimate speakers. This tuning, of course, depends on the application and assumes that the cost of a false positive and false negative are equal. If one is more costly than the other, we can change our verification threshold level t in order to reflect that.

Below, in Table 13, we present the EER for each model (using their references from Section 3.2.2) on the LibriSpeech and VCTK test datasets, discussed previously in Sections 3.1.1 and 3.1.4.

Table 13: Equal Error Rate

Model	Architecture	Embedding Size	LibriSpeech EER	VCTK EER
[1]	Convolutional w/ Avg. Pooling	32	10.7%	11.7%
[2]	Convolutional	64	11.1%	11.0%
[3]	Convolutional	64	10.9%	11.1%
[4]	Bidirectional LSTM	64	10.9%	11.8%
[5]	LSTM	128	10.9%	11.3%
[6]	GRU	128	11.1%	11.4%

It is, perhaps, unsurprising that the EER for the VCTK should be categorically higher. Recall, as stated in Section 3.1.4, its samples are taken from speakers with British accents, which do not constitute a large portion of the training datasets.

4.1.2 F_1-measure

Our inference task, as formulated in Section 3.2, can be interpreted as a classification task. That is, given some threshold t, determine whether to assign utterance $\mathbf{u}$ to a particular speaker s. In this framing, precision and recall tell us how our model performs as its classification task. They are defined,

$$Precision = \frac{TP}{TP + FP}$$
$$Recall = \frac{TP}{TP + FN}$$

where TP is the number of true positives, FP the number of false positives, and FN the number of false negatives. We can combine these metric in terms of the F_1-measure which is simply the harmonic mean of *precision* and *recall* defined above

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{4.1}$$

In Tables 14 and 15, below, we show all three of these statistics for the models listed in Section 3.2.2. Note that we, additionally, report the acceptance / rejection *threshold* for each model which determines the statistics, i.e. the one associated with the *equal error rate* of Section 4.1.1.

Table 14: LibriSpeech F_1-Measure

Model	Architecture	Precision	Recall	F_1-measure	Threshold
[1]	Convolutional w/ Avg. Pooling	0.997	0.879	0.936	0.177
[2]	Convolutional	0.998	0.879	0.936	0.207
[3]	Convolutional	0.997	0.878	0.935	0.203
[4]	Bidirectional LSTM	0.998	0.879	0.936	0.236
[5]	LSTM	0.995	0.888	0.941	0.246
[6]	GRU	0.999	0.887	0.940	0.268

Table 15: VCTK F_1-Measure

Model	Architecture	Precision	Recall	F_1-measure	Threshold
[1]	Convolutional w/ Avg. Pooling	0.995	0.953	0.976	0.155
[2]	Convolutional	0.993	0.892	0.943	0.177
[3]	Convolutional	0.995	0.892	0.943	0.176
[4]	Bidirectional LSTM	0.997	0.920	0.958	0.182
[5]	LSTM	0.998	0.896	0.945	0.200
[6]	GRU	0.998	0.898	0.947	0.205

4.1.3 Cosine Similarity

As presented in Section 3.2.1, each iteration of our training generates a cosine similarity matrix of size $[NM, NM]$ from the embedding vectors to their respective centroids. Here, we'll visualize the pairwse cosine

similarity of embedding vectors $\mathbf{e}_{sm}$ of an unseen batch from the test set. Again, here $N = M = 8$ and we hope to see a block diagonal matrix where the diagonal blocks have high similarity, e.g. they are predicted to have come from the same speaker, and the off-diagonal entries to have a low similarity. In Figure 10 and 11, below, we show results from the LibriSpeech and VCTK test sets.

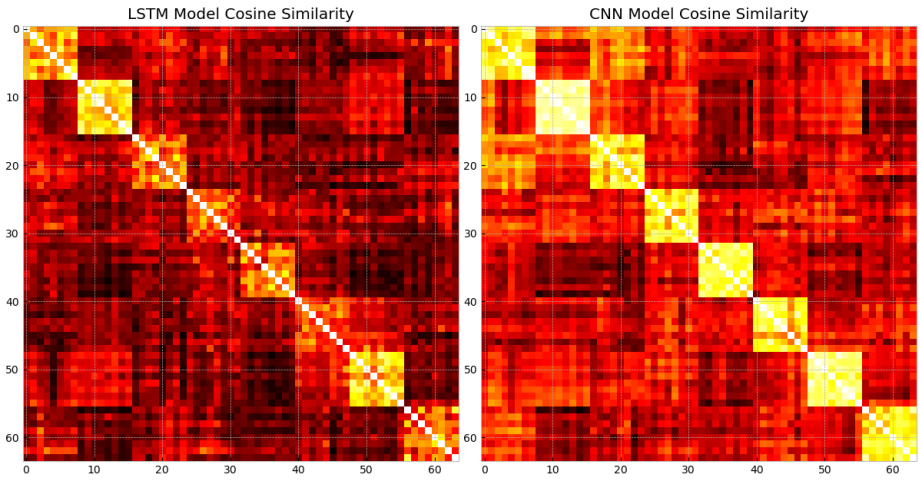


Figure 10: LibriSpeech Single Batch Embedding Pairwise Cosine Similarity

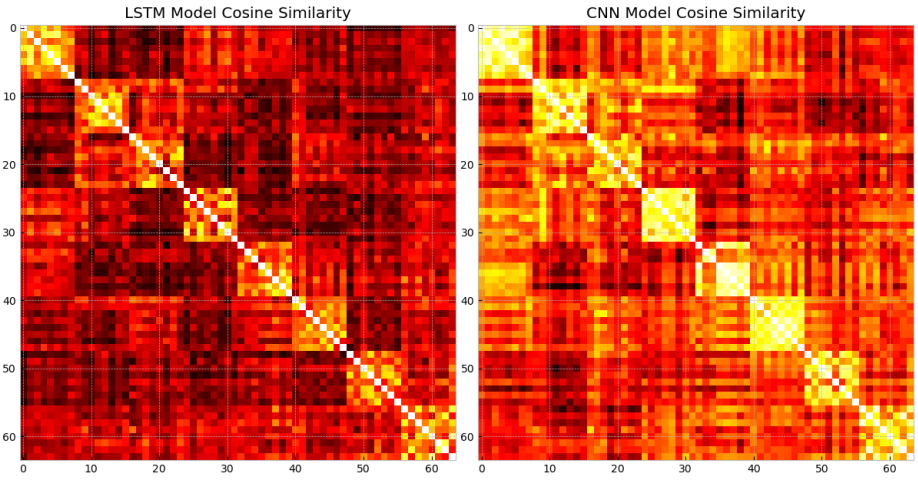


Figure 11: VCTK Single Batch Embedding Pairwise Cosine Similarity

Note here that the brighter sections denote a higher similarity and, conversely, the darker ones a lower similarity. The average cosine similarity matrix across the entire test set can be found in the *Appendix*. In order to gain an understanding of the performance across the entire test LibriSpeech and VCTK datasets, in Figures 12 and 13 below, we show the distribution of every pairwise embedding cosine similarity across the test dataset, stratified by embeddings from the same speaker and those from different speakers.

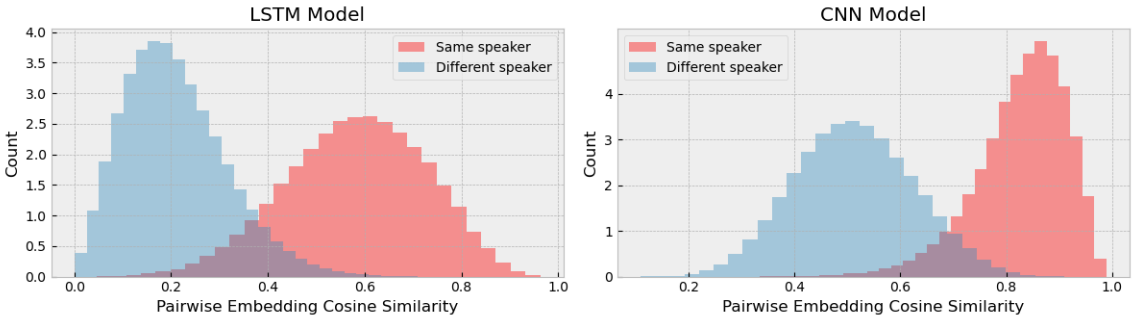


Figure 12: LibriSpeech Pairwise Embedding Cosine Similarity Distribution

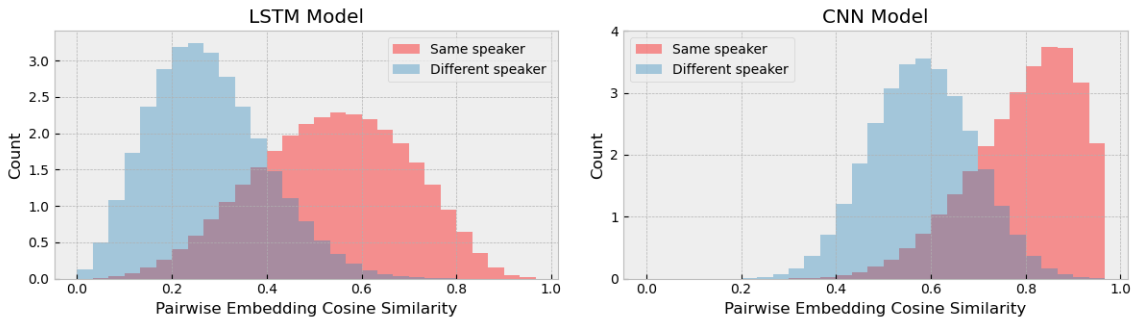


Figure 13: VCTK Pairwise Embedding Cosine Similarity Distribution

Distributions from all trained models can be found in the *Appendix*.

4.1.4 *t*-SNE

Ultimately, our model should be able to *separate* speaker voiceprints, their centroids, in our embedding space. Then, given an input utterance u and its embedding vector $\mathbf{e}_u$ it will be easier to assign that embedding to a specific speaker centroid. If we do not achieve clear separation, such a task will prove difficult.

To demonstrate that our model can achieve such separation, here we show an example of separation using the dimensionality reduction technique of t-distributed stochastic neighbor embedding (t-SNE) [14]. This algorithm is optimized to preserve neighborhood identity, which is exactly what we want to demonstrate here. The embedding vector $\mathbf{e}_m$ for all m utterances of a particular speaker s should end up in the same neighborhood if our model is performing well, and each speaker's respective centroid $\mathbf{c}_s$ should have a tight cluster. Examples from unseen speakers in the LibriSpeech and VCTK test sets are presented in Figures 14 and 15 below.

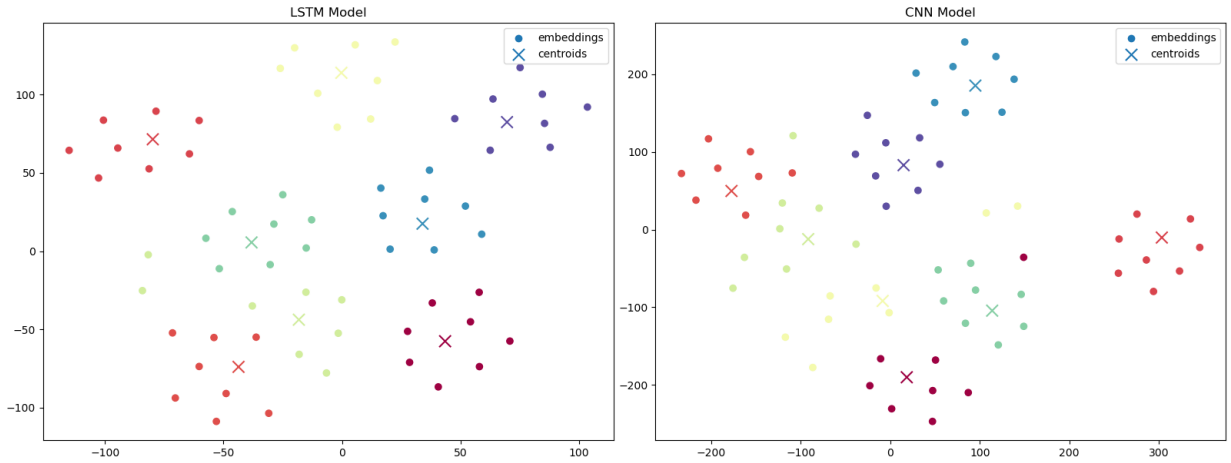


Figure 14: LibriSpeech $t-SNE$ Utterance Embeddings and Centroids

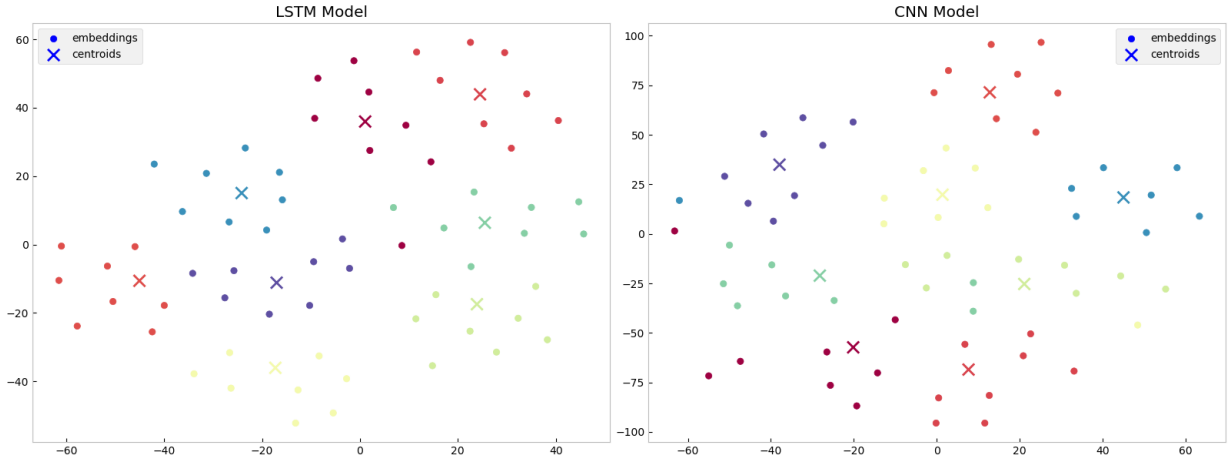


Figure 15: VCTK $t-SNE$ Utterance Embeddings and Centroids

4.1.5 k-Means

Rosenberg and Hirschberg (2007) proposed the V-measure, "validity", for evaluation of clustering methods [36]. It can be described in terms of its components *homogeneity* and *completeness*. Both metrics are bounded $(0, 1]$ where the closer to 1 they are, the more favorably the clustering method has performed in relation to its task. Given a set of data points N, a set of classes $C = \{c_i | i = 1, \ldots, n\}$, a set of clusters, $K = \{k_i | 1, \ldots, m\}$, we define A as the contingency table produced by the input clustering algorithm, where $a_{ij} \in A$ is the number of data points that are members of class c_i and elements of cluster k_j. In these terms we can define *completeness* and *homogeneity* in turn.

Completeness is a metric denoting how many members of a given class are members of the same cluster. It is defined,

$$c = \begin{cases} 1 & if\ H(K,C) = 0 \\ 1 - \frac{H(K|C)}{H(K)} & else \end{cases} \tag{4.2}$$

where

$$H(K|C) = -\sum_{c=1}^{|C|}\sum_{k=1}^{|K|}\frac{a_{ck}}{N}\log\left(\frac{a_{ck}}{\sum_{k=1}^{|K|}a_{ck}}\right)$$
$$H(K) = -\sum_{k=1}^{|K|}\frac{\sum_{c=1}^{|C|}a_{ck}}{n}\log\left(\frac{\sum_{c=1}^{|C|}a_{ck}}{n}\right)$$

Homogeneity is symmetric with *completeness*, and measures if clusters contain points from only a single class. It is defined,

$$h = \begin{cases} 1 & if\ H(C,K) = 0 \\ 1 - \frac{H(C|K)}{H(C)} & else \end{cases} \tag{4.3}$$

where

$$H(C|K) = -\sum_{k=1}^{|K|}\sum_{c=1}^{|C|}\frac{a_{ck}}{N}\log\left(\frac{a_{ck}}{\sum_{c=1}^{|C|}a_{ck}}\right)$$
$$H(C) = -\sum_{c=1}^{|C|}\frac{\sum_{k=1}^{|K|}a_{ck}}{n}\log\left(\frac{\sum_{k=1}^{|K|}a_{ck}}{n}\right)$$

Given the definitions of completeness, c, in Equation 4.2, and homogeneity, h, in Equation 4.3 the v-measure for a given β, V_β, is defined as the harmonic mean of h and c,

$$V_\beta = \frac{(1+\beta)hc}{(\beta h) + c} \tag{4.4}$$

In Table 16, below, we show the average of all three measures after performing the k-Means algorithm on the output embedding vectors of each batch contained within our test set in the high dimensional embedding vectors space.

Table 16: LibriSpeech Embedding k-Means Clustering V-Measures

Model	**Architecture**	**Completeness**	**Homogeneity**	V_1
[1]	Convolutional w/ Avg. Pooling	0.969	0.961	0.965
[2]	Convolutional	0.962	0.942	0.952
[3]	Convolutional	0.962	0.943	0.952
[4]	Bidirectional LSTM	0.983	0.980	0.981
[5]	LSTM	0.964	0.955	0.959
[6]	GRU	0.979	0.975	0.977

Table 17: VCTK Embedding k-Means Clustering V-Measures

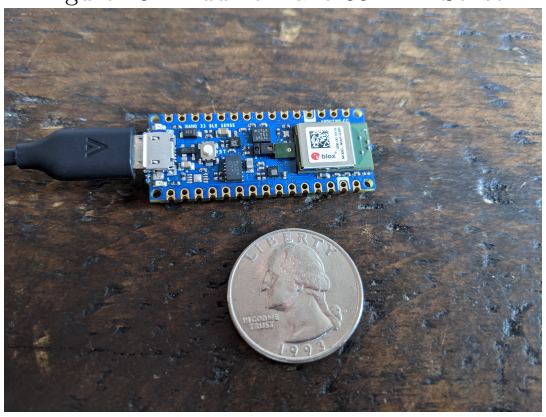
Model	**Architecture**	**Completeness**	**Homogeneity**	V_1
[1]	Convolutional w/ Avg. Pooling	0.800	0.777	0.788
[2]	Convolutional	0.820	0.790	0.804
[3]	Convolutional	0.813	0.780	0.796
[4]	Bidirectional LSTM	0.806	0.784	0.795
[5]	LSTM	0.787	0.757	0.772
[6]	GRU	0.805	0.781	0.793

Note that here $k = N = 8$, meaning each batch contains 8 unique speakers, N, and thus *should* contain 8 distinct clusters in our embedding space.

4.2 Edge Device Deployment

Ultimately, our goal is to place these trained models on our Arduino Nano edge device, pictured below in Figure 16.

Figure 16: Arduino Nano 33 BLE Sense



For this task we face two specific challenges. First, Arduino devices only support a small subset of `C/C++` written programs, thus we needed to re-implement the feature engineering steps detailed in Section 3.1.5. Secondly, as discussed in Section 3.3.1, the physical constraints of our edge device require that both our model and blocks required for feature computations fit on the space available. Below, we will present the approach to the first problem in Section 4.2.1, and the second in Section 4.2.2.

4.2.1 Feature Engineering

In order to generate input training features as described in Section 3.1.5, we made use of the `librosa` Python package [22]. Our edge device inference environment requires programs to be written in `C/C++`. For this reason, it was necessary to port the functionality of `librosa.feature.melspectrogram` to `C/C++`. These steps were implemented in the `feature_provider.h` file contained in our repository. Primarily, we provide a `FeatureProvider` object which follows the RAII (resource allocation is initialization) principle and is constructed by passing in a reference to a static raw waveform buffer where the raw audio samples are stored. The caller makes use of the `waveform_to_feature` method when this buffer is full of new samples. An example of how the caller uses this class is shown below in Listing 1,

Listing 1: `FeatureProvider` instantiation

```
float raw_waveform_buffer[waveform_length];
float feature_buffer[n_filter][num_frames];

feature::FeatureProvider* feature_provider = nullptr;

static feature::FeatureProvider fp(waveform_length, raw_waveform_buffer,
    window_length, hop_length, n_filter,
    signal_rate, nfft, num_frames);
feature_provider = &fp;
```

```
feature_provider->waveform_to_feature(feature_buffer);
```

where `waveform_length` is the number of samples in our buffer (e.g. 16kHZ * 1.2 sec = 19200), `n_filter` is the number of mel filters in our filter bank, and `num_frames` is the number of frames in our windowed audio. From there, the `feature_buffer` acts as the input `TfLiteTensor*` which gets passed into our `tflite::Model*`. The complete program can be found in the *Appendix*.

After re-implementing the steps described in Section 3.1.5, our `FeatureProvider` class is able to generate a mel-based spectrogram with similarity to those generated by `librosa` used in training, as shown below in Figure 17,

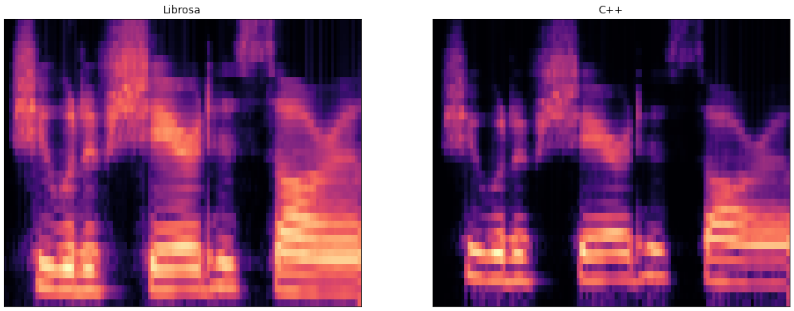


Figure 17: C++ Mel Spectrogram

4.2.2 TensorflowLite Micro Model

As mentioned, we made use of the TensorflowLite Micro API which has support for the ARM-Cortex M4 instruction set. As discussed in Section 2.7, TensorflowLite Micro supports a subset of all Tensorflow Operations. Given a Tensorflow graph in the Python API (how the model is trained), we can make use of the `tf.lite.TFLiteConverter.from_saved_model` method to provide a directory of Protocol Buffers containing the original model. When the target device is a microcontroller, the model will be serialized into a `tflite::FlatBufferModel` which is a `C` array of `char` placed in a header file, `model.h`. In Table 18, below, we report the serialized sizes of each model with their references from Section 3.2.2.

Table 18: TensorflowLite Micro Serialized Model Sizes

Model	Architecture	Serialized Model Size
[1]	Convolutional w/ Avg. Pooling	20kB
[2]	Convolutional	32kB
[3]	Convolutional	108kB
[4]	Bidirectional LSTM	NA[1]
[5]	LSTM	1.2MB
[6]	GRU	744kB

[1]Bidirectional RNNs are currently not supported.

A sample program of how a model is used from the header file is shown below in Listing 2,

Listing 2: Tensorflow Lite `MicroInterpreter` Usage

```
#include "model.h"

const tflite::Model* model = nullptr;
tflite::MicroInterpreter* interpreter = nullptr;

model = tflite::GetModel(speaker_model); // speaker_model defined in model.h
tflite::AllOpsResolver resolver;
uint8_t tensor_arena[tensor_arena_size];
static tflite::MicroInterpreter static_interpreter(model, resolver,
    tensor_arena, tensor_arena_size, error_reporter);
interpreter = &static_interpreter;
TfLiteStatus allocate_status = interpreter->AllocateTensors();
model_input = interpreter->input(0);

TfLiteStatus invoke_status = interpreter->Invoke();
TfLiteTensor* output = interpreter->output(0);
```

The `tflite::AllOpsResolver` will provide the Tensorflow operations that we need to specify our model. The `tf.lite.Interpreter` object asks us to set an input `TFLiteTensor` and then we can call the `Invoke` method to get an output tensor, in our case the embedding vector for a given input utterance.

As stated in Section 3.2, our objective then is to compare the model's output embedding vector with that of an enrolled speaker. For us, similarity is defined by cosine similarity and we set a threshold t to accept or reject the new utterance. In our program this looks like,

Listing 3: Embedding Acceptance / Rejection

```
#include "embedding.h"
#include "matrix_math.h"

const float threshold = 0.3;

float similarity = MatrixMath::cosine_similarity(
    output->data.f, enrolled_embedding, embedding_len
);
if (similarity > threshold) {
    // perform acceptance action
} else {
    // perform rejection action
}
```

where embedding.h contains the enrolled speaker's embedding vector as a `float[EMBEDDING_LEN]` and defines the `const EMBEDDING_LEN`, which is dependent on the model we use and the shape of its final Dense layer. Here, we make use of the `cosine_similarity` function defined in the `MatrixMath` namespace contained in the `matrix_math.h` header file. Those implementations can be found in the *Appendix*.

5 Conclusion

In this work, we have demonstrated that it is feasible to build a *reasonably* accurate, dependent on the use case, text-independent speaker verification system that will fit onto even the smallest microcontrollers containing little more than an onboard microphone and 1MB of Flash memory. Here, we'll briefly summarize our findings and provide some motivations for future work.

5.1 Summary

We have demonstrated that the computational and space limitations of microcontrollers pose a significant challenge to accurately representing a speaker s, via their centroid $\mathbf{c}_s$ embedding vector, in a high dimensional space. Unlike previous research, where online inference is done with sufficent computational resources and, thus, does not have a constrained model space imposed upon it, here we have investigated what smaller models can do. Given our almost four-fold increase in equal error rate when compared to most state-of-the-art systems, it is clear that deeper, wider, models are able to more accurately represent speakers via their embeddings. These models are able to represent cross-connections with a longer range of temporal and frequency features.

That said, we argue that our models do in fact learn something. They have shown they are able to separate speakers in some high dimensional spaces. In fact, even given both enrollment and inference utterances which the model has not seen it is still capable of assigning them to particular speakers, as we demonstrated in Section 4. This would suggest that it has possibly learned *something* about representing speakers.

Here, we made use of both recurrent and convolutional neural networks in order to benchmark them. Though both performed relatively well, it appears that convolutional models can be just as accurate with many fewer parameters for our use case, and thus are more easily utilized on edge devices. As discussed in Section 2.5, both model architectures continue to be used in modern, state-of-the-art, systems. Though, convolutional models typically have been more applicable to the image-like formulation we have used by representing our utterance's via their spectrogram. It is also important to note that convolutional models of this size are *not* able to connect long-term temporal dependencies given the same filter size. This is something recurrent networks can achieve. One could argue, as shown by research presented in Section 2.2, that a significant chunk of speaker information is contained in those long-term dependencies. We are, then, effectively throwing that information away.

Our work constitutes a solid starting point for attempting to express the modeling logic behind larger models directly in the context of a constrained model space.

5.2 Further Work

In the course of our work, it became apparent these are still plenty of avenues for the field of edge device speaker verification to explore. Here, we will point out relevant suggestions in both the modeling and systems spaces.

Much is still to be done on developing models that are both highly accurate and *small enough* to be useful in real world applications at the edge. Here, our accuracy is likely acceptable in domains where there exist other, potentially manual, secondary systems to confirm results. However, for applications, such as security, which require a high degree of confidence our system would likely not be appropriate. Promising research is being done to investigate how to prune, or attune, larger models, which are potential significantly sparse [9, 50]. Hopefully, this research will allow for the compression of larger, more accurate, models into smaller versions which retain much of the parent model's performance.

There exists little support for robust and well-tested audo feature engineering frameworks targeting microcontroller devices, such as the Arduino Nano. The `librosa` package provides an extensive Python API to convert many source audio files and raw waveforms into different representatiosn which are useful for machine learning tasks. Native packages for Pytorch, in `torchaudio`, and Tensorflow, in `tf.audio`, have also been under heavy development. A system which exposes this functionality to users targeting the lower level languages, such as `C`, which run on microcontrollers would allow for easier, and more robust, development in the microcontroller domain. The presence and popularity of TensorflowLite Micro indicates that there is real interest in both research and industry for machine learning applications at the edge. It is important we continue to consider the centrality of the feature engineering steps in these systems and contribute to their development.

Appendix

Cosine Similarity Program Implementation

```
float cosine_similarity(float a[], float b[], int length) {
    float numerator = dot_product(a, b, length);
    float a_norm = std::sqrt(dot_product(a, a, length));
    float b_norm = std::sqrt(dot_product(b, b, length));
    return numerator / (a_norm * b_norm);
};
```

Average Embedding Cosine Similarity Matrix

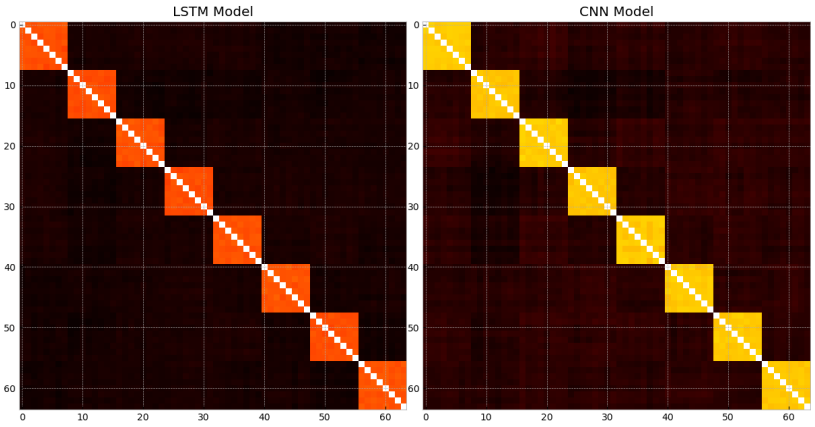


Figure 18: LibriSpeech Average Embedding Cosine Similarity Matrix

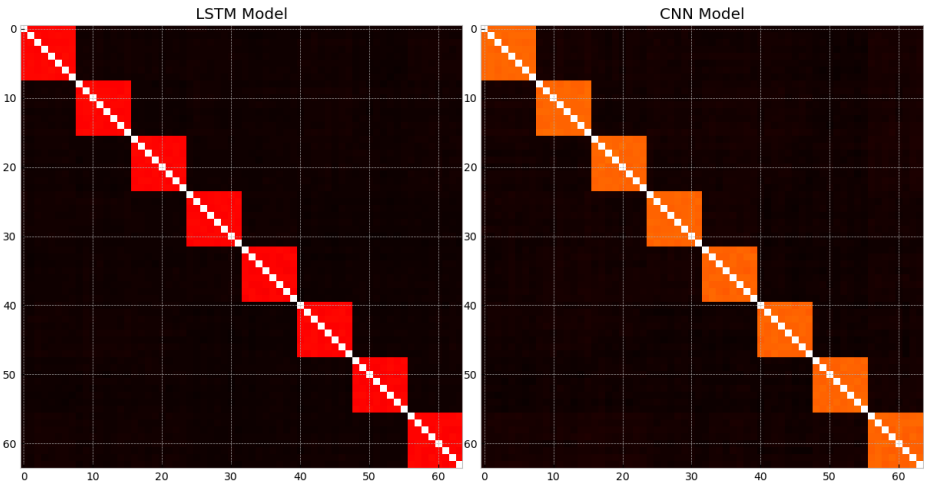


Figure 19: VCTK Average Embedding Cosine Similarity Matrix

Pairwise Embedding Cosine Similarity Distributions

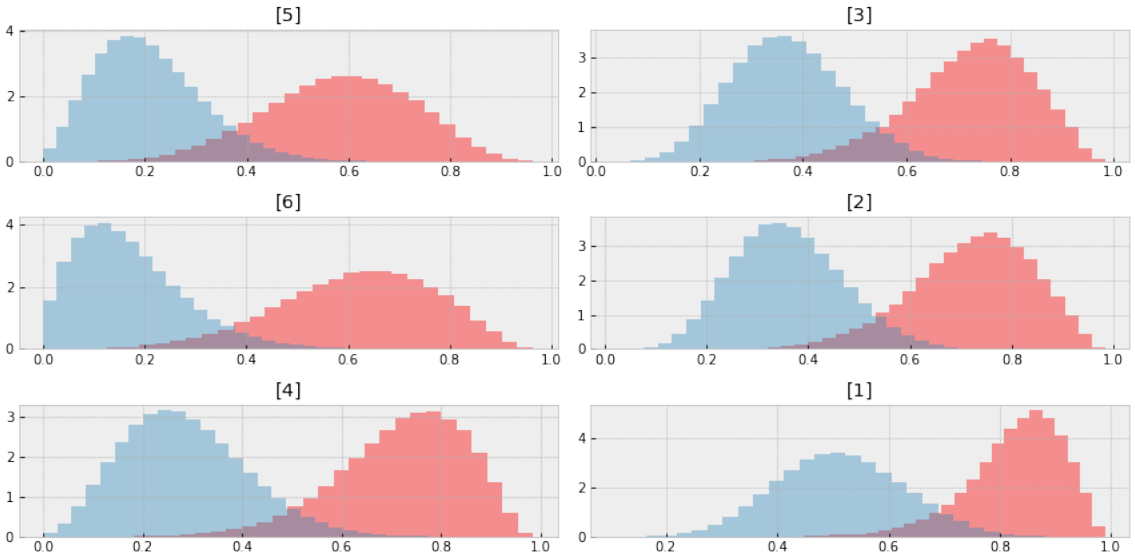


Figure 20: LibriSpeech Pairwise Embedding Cosine Similarity Distributions

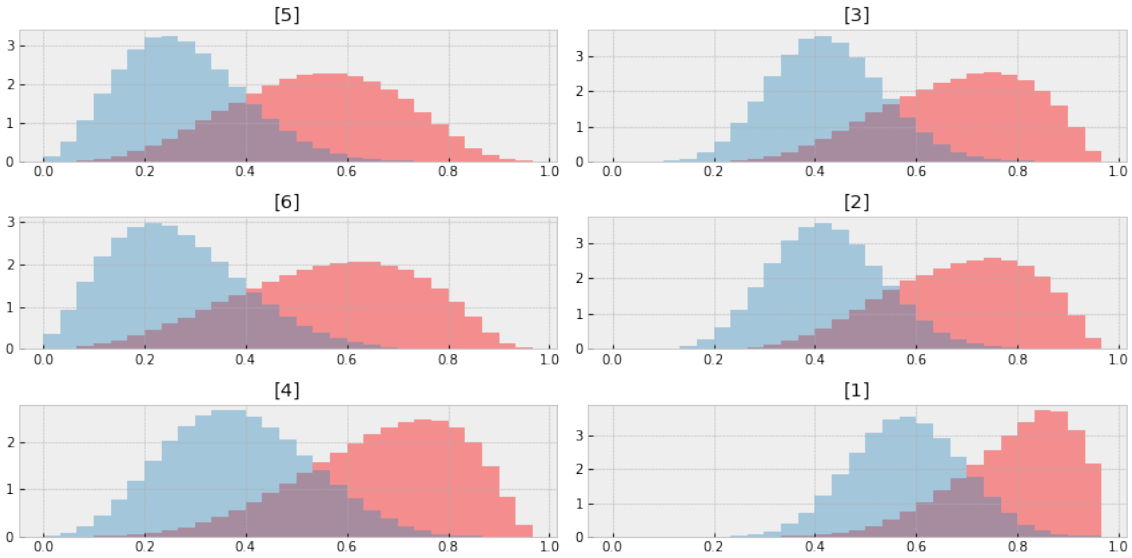


Figure 21: VCTK Pairwise Embedding Cosine Similarity Distributions

References

[1] S. Ahmed *et al*, "IoT Based Real Time Noise Mapping System for Urban Sound Pollution Study," arXiv:2002.11188v1 [cs.NI], Feb. 2020.

[2] R. Ardila et al, "Common Voice: A Massively-Multilingual Speech Corpus," arXiv:1912.06670v2 [cs.CL], Mar. 2020.

[3] C.R. Banbury *et al*, "Benchmarking TinyML Systems: Challenges and Direction," arXiv:2003.04821v3 [cs.PF] Aug. 2020.

[4] H. Beigi, "Fundamentals of Speaker Recognition," doi: 10.1007/978-0-387-77592-0 Springer Science: New York, NY, 2011

[5] J. Chorowski *et al*, "Unsupervised Speech Representation Learning Using WaveNet Autoencoders," arXiv:1901.08810v2 [cs.LG], Sep. 2019.

[6] J.W. Cooley and J.W. Tukey, "An Algorithm for the Machine Calculation of Complex Fourier Series"

[7] R. David *et al*, "Tensorflow Lite Micro: Embedded Machine Learning on TinyML Systems," arXiv:2010.08678v2 [cs.LG], Oct. 2020.

[8] N. Dehak, P. Kenny, R. Dehak, P. Dumouchel, and P. Ouellet, "Front-end Factor Analysis for Speaker Verification," IEEE Transactions on Audio, Speech, and Language Processing, Mar. 2010.

[9] J. Esling *et al*, "Diet Deep Generative Audio Models with Structured Lottery," arXiv:2007.16170v1 [cs.LG], Jul. 2020.

[10] B. Gu and W. Guo, "Gaussian Speaker Embedding Learning for Text-Independent Speaker Verification,"

[11] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," arXiv:1512.03385v1 [cs.CL] Dec. 2015.

[12] Y. He *et al*, "Streaming End-to-End Speech Recognition for Mobile Devices," arXiv:1811.06621v1 [cs.CL] Nov. 2018.

[13] G. Heigold *et al*, "End-to-End Text-Dependent Speaker Verification,"

[14] G. Hinton and S. Roweis, "Stochastic Neighbor Embedding," https://cs.nyu.edu/~roweis/papers/sne_final.pdf

[15] J.W. Jung *et al*, "RawNet: Advanced End-to-End Deep Neural Network Using Raw Waveforms for Text-Independent Speaker Verification," arXiv:1904.08104v2 [eess.AS], Jul. 2019.

[16] C.C. Kao *et al*, "Sub-band Convolutional Neural Networks for Small-footprint Spoken Term Classification," arXiv:1907.01448v1 [eess.AS], Jul. 2019.

[17] Kinnunen *et al*, "t-DCF: a Detection Cost Function for the Tandem Assessment of Spoofing Countermeasures and Automatic Speaker Verification," arXiv:1804.09618v2 [eess.AS] Apr. 2019.

[18] Y. LeCun, Y. Bengio, and G. Hinton, "Deep Learning," *Nature*, 521, May 2015.

[19] C. Li et al, "Deep Speaker: an End-to-End Neural Speaker Embedding System," arXiv:1705.02304v1 [cs.CL], May 2017.

[20] A.T. Liu *et al*, "Mockingjay: Unsupervised Speech Representation Learning with Deep Bidirectional Transformer Encoders," arXiv:1910.12638v2 [eess.AS], Feb. 2020.

[21] M.S. Mary N J, S.V. Katta, and S. Umesh, "S-vectors: Speaker Embeddings based on Transformer's Encoder for Text-Independent Speaker Verification," arXiv:2008.04659v1 [eess.AS], Aug. 2020.

[22] B. McFee *et al*, "librosa: Audio and music signal analysis in python." In Proceedings of the 14th python in science conference, pp. 18-25. 2015.

[23] I. McGraw et al, "Personalized Speech Recognition on Mobile Devices," arXiv:1603.031v2 [cs.CL], Mar. 2016.

[24] Z. Meng, Y. Zhao, J. Li, and Y. Gong, "Adversarial Speaker Verification," arXiv:1904.12406v1 [cs.SD], Apr. 2019.

[25] M. Merenda, C. Porcaro, and D. Iero, "Edge Machine Learning for AI-Enabled IoT Devices: A Review," Sensors, 20 (2533), Apr. 2020.

[26] A. Nagrani, J.S. Chung, and A. Zisserman, "VoxCeleb: A Large-scale Speaker Identification Dataset," *INTERSPEECH*, 2017

[27] A. Nagrani, J.S. Chung, W. Xie, and A. Zisserman, "Voxceleb: Large-scale Speaker Verification in the Wild," *Computer Science and Language*, (60), 2020

[28] Nordic Semiconductor, "nRF52840: Product Specification,", Feb. 2019.

[29] V. Panayotov, G. Chen, D. Povey, and S. Khundanpur, "LibriSpeech: An ASR Corpus Based on Public Domain Audio Books," ICASSP, 2015

[30] H. Park, J. Park, and S.W. Lee, "End-to-End Trainable Self-Attentive Shallow Network for Text-Independent Speaker Verification,"

[31] L.R. Rabiner and R.W. Schafer, "Introduction to Digital Speech Processing," Foundation and Trends in Signal Processing, 1 (1-2), doi:10.1561.2000000001, 2007

[32] S. Ramoji, P. Krishnan, and S. Ganapathy, "NPLDA: A Deep Neural PLDA Model for Speaker Verification," arXiv:2002.03562v2 [eess.AS], May 2020.

[33] M. Ravanelli and Y. Bengio, "Speaker Recognition from Raw Waveform with Sincnet," arXiv:1808.00158v3 [eess.AS], Aug. 2019.

[34] D.A. Reynolds and R.C. Rose, "Robust Text-Independent Speaker Identification Using Gaussian Mixture Speaker Models," IEEE Transactions on Speech and Audio Processing, 3 (1), Jan. 1995.

[35] Z. Ren, Z. Chen, and S. Xu, "Triplet Based Embedding Distance and Similarity Learning for Text-independent Speaker Verification," arXiv:1908.02283v1 [eess.AS], Aug. 2019.

[36] A. Rosenberg and J. Hirschberg, "V-Measure: A Conditional Entropy Based External Cluster Evaluation Measure," Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, pp. 410-420, Jun. 2007.

[37] T. N. Sainath, R. J. Weiss, A. Senior, K. W. Wilson, and O. Vinyals, "Learning the Speech Front-end with Raw Waveform CLDNNs," Interspeech, 2015.

[38] H. Salehghaffari, "Speaker Verification using Convolutional Neural Networks," arXiv:1803.05427v2 [eess.AS], Aug. 2018.

[39] B. Shi *et al*, "Compression of Acoustic Event Detection Models with Quantized Distillation," arXiv:1907.00873v1 [eess.AS], Jul. 2019.

[40] Y. Shi, Q. Huang, and T. Hain, "Improving Robustness in Speaker Identification using a Two-Stage Attention Model," arXiv:1909.11200v1 [eess.AS], Sep. 2019.

[41] Y. Shi, Q. Huang, and T. Hain, "H-Vectors: Utterance-Level Speaker Embeddings Using a Hierarchical Attention Model," arXiv:1910.07900v2 [cs.CL], Oct. 2019.

[42] Siri Team, "Hey Siri: An On-device DNN-power Voice Trigger for Apple's Personal Assistant," Apple Machine Learning Research: https://machinelearning.apple.com/research/hey-siri, Oct. 2017.

[43] D. Snyder, D. Garcia-Romero, D. Povey, and S. Khundanpur, "Deep Neural Network Embeddings for Text-Independent Speaker Verification," 2017 INTERSPEECH Proceedings

[44] D. Snyder *et al*, "X-Vectors: Robust DNN Embeddings for Speaker Recognition," ICASSP 2018

[45] Y. Tang *et al*, "Deep Speaker Embedding Learning with Multi-Level Pooling for Text-Independent Speaker Verification," arXiv: 1902.07821v1 [cs.CL], Feb. 2019.

[46] D.A. van Leeuwen and N. Brümmer, "An Introduction to Application-Independent Evaluation of Speaker Recognition Systems,"

[47] E. Variani *et al*, "Deep Neural Networks for Small Footprint Text-Dependent Speaker Verification," 2014 IEEE International Conference on Acoustic, Speech and Signal Processing (ICASSP), 2014.

[48] A. Waibel *et al*, "Phoneme Recognition using Time-Delay Neural Networks," IEEE Transactions on Acoustics, Speech, and Signal Processing, 37 (3), Mar. 1989.

[49] L. Wan *et al*, "Generalized End-to-End Loss for Speaker Verification," arXiv:1710.10467v4 [eess.AS], Jan. 2019.

[50] A. Wong, M. Famouri, M. Pavlova, and S. Surana, "TinySpeech: Attention Condensers for Deep Speech Recognition Neural Networks on Edge Devices," arXiv:2008.04245v6 [eess.AS] Oct. 2020.

[51] J. Yamagishi, C. Veaux, and K. MacDonald, "CSTR VCTK Corpus: English Multi-speaker Corpus for CSTR Voice Cloning Toolkit (version 0.92)", [sound]. University of Edinburgh. The Centre for Speech Technology Research (CSTR). https://doi.org/10.7488/ds/2645. 2019.

[52] S. Yun *et al*, "An End-to-End Text-independent Speaker Verification Framework with a Keyword Adversarial Network," arXiv:1908.02612v1 [eess.AS], Aug. 2019.

[53] N. Zeghidour *et al*, "End-to-End Speech Recognition from Raw Waveform," arXiv:1806.07098v2 [cs.CL], Jun. 2018.

[54] N. Zeghidour *et al*, "Fully Convolutional Speech Recognition," arXiv:1812.06864v2 [cs.CL], Apr. 2019.