X-Vector Speaker Embedding Systems

Eastern Washington University

Tobias Rebant
Thesis Advisor: Prof. Viktoria Taroudaki

Winter/Spring 2024

Contents

1 Introduction

With the rise of artificial intelligence (AI), we see its application in many industries, driving innovation and reshaping traditional processes across sectors such as healthcare, finance, and education. The predictive power of machine learning models has become unmatched in many of these fields and is revolutionizing the way we work and live.

One of these new applications has been in the world of human speech processing. Within this field, several key areas - Speaker Identification and Verification, Voice Cloning, and Text to Speech applications - have seen rapid improvements with the introduction of machine learning models. In all of these fields, where once statistical methods were the norm, they have since been dominated by machine learning methods [7] .

Here is a brief overview of each of these.

1.1 Speaker Recognition

In the Speaker Recognition field, there are generally two tasks: Verification and Identification. Verification determines whether a person is who they claim to be by comparing their voice to a previously stored voice print in the system. Identification, on the other hand, matches an input voice against a database of stored voice prints to determine the speaker's identity from among multiple possibilities. These tasks typically rely on sophisticated algorithms to confirm identity through voice pattern matching.

In both cases, a fundamental challenge lies in expressing a person's voice characteristics digitally to enable these comparisons. This digital representation must capture the unique aspects of a person's voice - such as pitch, timbre, and speaking patterns - in a format that computers can efficiently process and compare.

1.2 Text to Speech

Text-to-Speech (TTS) technology, which converts written text into spoken words, represents one of the most complex and rapidly evolving areas of speech processing. Unlike human readers who naturally understand and vocalize text, TTS systems must break down this seemingly simple task into multiple sophisticated steps to produce natural-sounding speech. Modern TTS systems must not only accurately pronounce words but also capture the nuances of human speech, including emotional tone, natural rhythm, and appropriate emphasis. This technology has found widespread applications, from reading assistance for the visually impaired to generating voiceovers for digital content, automated customer service systems, and virtual assistants. The advancement of deep learning has dramatically improved the quality of synthesized speech.

At its core, TTS involves three main steps: analyzing the input text to determine what needs to be said, converting this analysis into a detailed plan for speech production, and finally generating the actual audio [8]. However,

to make the generated speech sound natural and engaging, the system needs information about how the voice should sound - its pitch, timbre, speaking style, and other unique characteristics.

1.3 Voice Cloning Model

Voice cloning takes text-to-speech technology a step further by attempting to replicate a specific person's voice. As shown in Figure 1 the process involves two key inputs: the text to be spoken and a sample of the target voice we want to clone. The power of voice cloning lies in how it captures and transfers the unique characteristics of a speaker's voice - their "voice identity" - onto new spoken content.

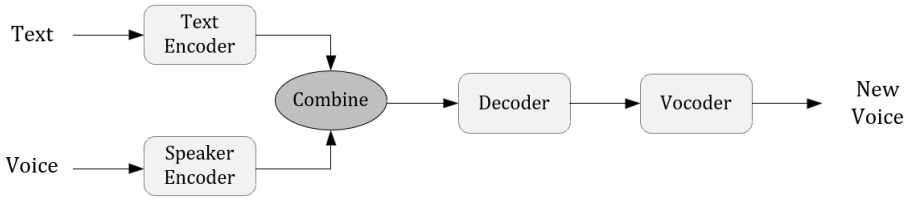

Figure 1: Simplified Voice Cloning Model [9]

The Speaker Encoder component in Figure 1 plays a crucial role here: it analyzes the target voice and creates a mathematical representation that captures the essential characteristics that make that voice unique. This representation is then combined with the encoded text to guide the synthesis process, ensuring the generated speech matches the target voice's qualities.

The Decoder and Vocoder components then work together to generate the final audio. The key insight to turn our focus on, is that the quality of the cloned voice fundamentally depends on how well we can mathematically represent the speaker's voice characteristics through these representations.

1.4 Nessesity for Speaker Representation

The key finding, is the common requirement across these applications; the need for a speaker representation. Whether serving as input for Voice Cloning and Speaker Recognition purposes, or as part of the output in Text-to-Speech systems, this speaker representation is crucial. This representation, known as the **speaker embeddings**, forms the foundation for accurately capturing and reproducing the unique characteristics of a speaker's voice.

Speaker embeddings are fixed-dimensional vectors that encapsulate the essential features of a person's voice [4]. These embeddings allow models to distinguish between different speakers and retain the unique vocal traits necessary for tasks like voice synthesis and recognition. The creation of robust and reliable

speaker embeddings is a complex process involving advanced machine learning techniques and large datasets of voice samples.

The importance of speaker embeddings cannot be overstated. In Speaker Recognition, they enable systems to verify identities and match voices with high precision. In Voice Cloning, they allow for the seamless transfer of vocal traits from one voice to another, creating lifelike synthetic speech. In Text-to-Speech, embeddings ensure that synthesized speech maintains the unique characteristics of the intended speaker, making the output more natural and personalized.

This report will delve deeply into the topic of speaker embeddings, exploring their development, and implementation. By understanding the intricacies of speaker embeddings, we can better appreciate their pivotal role in the advancements of AI-driven speech.

1.5 History of Speaker Embeddings

The evolution of speaker recognition technology reflects a fundamental shift in how we mathematically represent human voice characteristics. This journey - from direct signal analysis to learned embeddings - demonstrates the field's progress toward more robust and efficient speaker recognition systems.

1.5.1 Early Approaches: Direct Feature Analysis

Early speaker recognition systems worked directly with acoustic features extracted from speech signals, primarily using Spectrograms and Mel-Frequency Cepstral Coefficients (MFCCs) [11]. While these features effectively captured voice characteristics, they required complex statistical methods to make speaker recognition decisions. Two key statistical approaches emerged:

Gaussian Mixture Models (GMMs) modeled the probability distribution of these acoustic features, creating speaker-specific statistical profiles that could distinguish between different voices. And Hidden Markov Models (HMMs), complemented GMMs by modeling how these features changed over time, capturing the dynamic aspects of speech [11].

These methods were effective, but struggled with variations in recording conditions and required significant computational resources for speaker comparison [11]. While these approached worked, and were innovative for their time, further innovations were nessesary for reliable and widespread use.

1.5.2 The Rise of Speaker Embeddings

The field's first major paradigm shift came around 2010 with the introduction of i-vectors (identity vectors) [10]. This innovation marked the transition from working with raw acoustic features to using fixed-dimensional embeddings - compact mathematical representations of speaker characteristics. I-vectors achieved two crucial improvements:

They captured both speaker characteristics and recording session variability

in a low-dimensional space They made speaker recognition more computationally efficient by enabling direct comparison of these compact representations

1.5.3 Deep Learning Revolution

The emergence of deep learning brought another fundamental shift in approach. Rather than relying on hand-crafted features and statistical models, neural networks could learn optimal speaker representations directly from speech data [11].

d-vectors

One of the early neural network-based approaches was the d-vector. Introduced by Google in 2014, this method involved training a deep neural network for speaker classification tasks, where speaker-specific features were extracted from the second-to-last layer (also called the penultimate layer) of the neural network. The resulting d-vectors offered a more refined representation of speaker characteristics compared to traditional methods.

x-vectors

The development of x-vectors happened between 2016 and 2018, and represented the next major advancement in neural speaker embeddings. This evolution occurred in several key stages:

1. Initial Innovation (2016): X-vectors introduced a Convolutional Neural Network with Network-in-Network (NiN) architecture, specifically designed to capture long-term speaker characteristics [5].

2. Architectural Refinement (2017): Snyder et al. made crucial modifications: Separated the embedding extraction from the verification task implemented variable-length input processing using Time-Delay Neural Networks Introduced a more effective training approach [4].

3. Maturation (2018): The seminal paper "X-VECTORS: ROBUST DNN EMBEDDINGS FOR SPEAKER RECOGNITION" established x-vectors as the new standard. This paper, while not improving upon the model architecture did introduce ways to augment data to increase model performance, which highlights the critical role of high-quality data in achieving robust performance in deep learning tasks [3].

1.5.4 Impact on Speaker Recognition

The introduction of x-vectors marked a major milestone in speaker recognition technology. X-vectors provided a more accurate and reliable method for speaker embedding, outperforming previous methods in various benchmarks. They offered enhanced robustness to variations in recording conditions and speaker states, such as changes in emotion or health. This approach became the norm at the time of release [3].

The evolution of speaker embeddings from traditional statistical methods to advanced deep learning techniques highlights the significant progress made

in the field. These advancements have paved the way for more accurate, reliable, and scalable speaker recognition systems, revolutionizing applications in security, forensics, and personal assistants, among others.

1.6 Speaker Encoder Overview

The system that extracts the Speaker Embedding is called the Speaker Encoder. The Speaker Encoder translates raw audio inputs into the mathematical representation of a speaker's unique vocal characteristics.

This process has three basics steps we will cover in brief here, and then later delve into, in more detail

Raw Audio Processing

Input audio from a target speaker is preprocessed to normalize volume, which is essential for the model to focus on the timbral qualities of the voice rather than volume discrepancies. The audio is also cut (or Framed) into the pre-determined lengths for input into the model, to efficiently be fed into the neural network, as these fixed-size segments help the model process data consistently while focusing on relevant time intervals, ensuring stable performance and reducing computational overhead. Other filters are applied to possibly reduce noise or remove irrelevant sounds. This prepares our data for feature extraction.

Feature Extraction

Following preprocessing, the audio is transformed into a structured feature representation that a neural network can effectively process. This is achieved by converting the raw waveform into time-frequency features, in our case through the use of Mel-scale filter banks (FBANKs).

FBANKs are a set of filters that mimic the human ear's response to sound, capturing the energy distribution across different frequency bands. They are computed by applying a series of triangular filters to the power spectrum of the audio signal, which is obtained through the Short-Time Fourier Transform (STFT). The resulting FBANKs represent the energy in each frequency band over time, providing a rich and informative representation of the audio signal.

These features are particularly well-suited for modern deep learning models, which can learn directly from them without additional processing. FBANKs are considered more expressive than compact features like Mel-Frequency Cepstral Coefficients (MFCCs), which reduce dimensionality and decorrelate features further than f-banks. MFCCs remain popular in traditional systems due to their efficiency and smaller size.

By extracting these features we provide the neural network with a perceptually meaningful and information-rich representation of speech, laying the foundation for accurate and robust speaker embedding.

Speaker Embedding Extraction

This is where the previously discussed Deep Neural Networks extract, in our case, the x-vector. The x-vector extraction network works like a one dimensional convolutional neural network (CNN). Specifically, it uses convolutional filters to scan speech segments for characteristic patterns, and then through multiple layer in the network, it captures the speaker's vocal identity across multiple

frames. This let's the network extract Long-term speaker features which are the stable vocal traits, such as timbre or pitch range, that persist over different utterances, ensuring a consistent and reliable voice representation.

1.7

Having laid a solid groundwork of what Speaker Embedding systems are, we can now delve into the specifics of each stage involved in their development and implementation. By examining the nuances of raw audio preprocessing, feature extraction, and the extraction of speaker embeddings, we aim to shed light on the critical aspects that contribute to the performance and reliability of these systems. This detailed exploration will give us an understanding of how these systems work from beginning to end, and prepare us for future studies of more advanced topics in Speaker Embedding Systems.

2 Understanding Sound and Signals

To understand how speaker embeddings capture voice characteristics, we must first understand how sound itself is represented mathematically. At its most fundamental level, all sound - whether a musical note, a vowel sound, or a consonant - can be described using sine waves. While real-world sounds are complex combinations of many overlapping waves with different frequencies, amplitudes, and phases, we can mathematically represent them using the principles of sine waves. We represent them as such:

$$x(t) = A\sin(2\pi \times (ft - \varphi))$$

where A is amplitude, f is frequency, t is time, φ is phase. Each of these affects a different part of the sine wave representation of sound.

Amplitude (A)

Amplitude is a measure of the strength, intensity, or loudness of a wave, visually represented by the height of the wave's peak from its rest position.

Frequency (f)

Frequency is the number of cycles a wave completes in one second, measured in Hertz (Hz). It determines the pitch of the sound; higher frequencies correspond to higher pitches, and lower frequencies correspond to lower pitches. In the sine wave formula, $A\sin(2\pi \times (ft - \varphi))$, frequency directly influences the rate at which the sine wave oscillates.

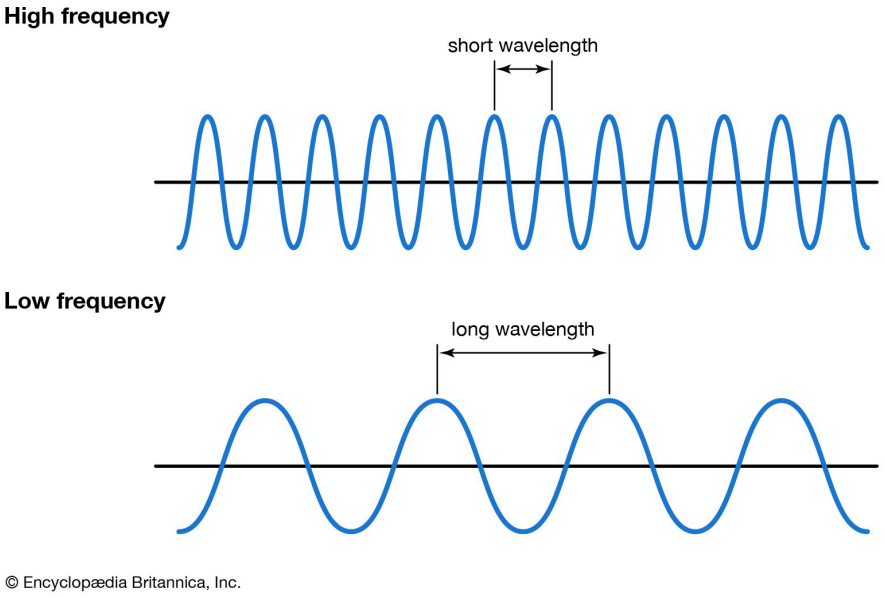

Figure 2: Frequency of Sound

Time (t)

Time, in the context of the sine wave equation, represents the progression of the wave as it moves forward. Varying the time variable doesn't fundamentally change the characteristics of the sine wave, but rather, lets us examine the signal at different *time* points.

Phase (φ)

Phase represents the initial angle of the sine wave at time $t = 0$. It's measured in radians and affects the starting point of the sine wave. In terms of sound, phase changes the position of the wave's peaks and troughs without altering its frequency or amplitude. This means the sound's pitch and loudness remain unchanged, but the wave's shape shifts left or right depending on the phase value.

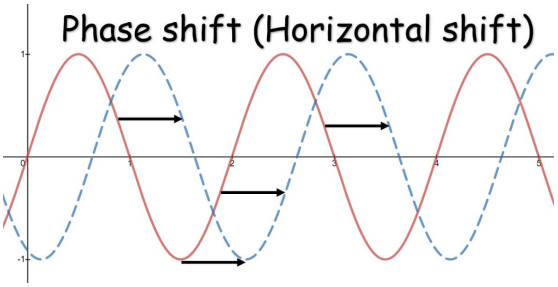

Figure 3: Phase of Sound

Signal Superposition

Real-world signals, such as music, speech, and environmental sounds, are not simple sine waves but rather composite signals made up of many different sine waves of varying frequencies, amplitudes, and phases. This complex composition is what gives these signals their unique characteristics.

The superposition principle explains how we can represent these signals coherently. The superposition principle states, that in linear systems, the net response at a given place and time caused by two or more stimuli is the sum of the responses that would have been caused by each stimulus individually. Or more simply put, when multiple waves occur together, they combine by adding their instantaneous values at each point in time. The resulting complex wave is simply the sum of all contributing sine waves:

Thus, if a complex signal can be considered as composed of multiple sine waves, the overall signal is the sum of these individual sine waves:

$$x(t) = \sum_{n} A_n \sin(2\pi f_n t + \varphi_n)$$

where A_n, f_n, and φ_n represent the amplitude, frequency, and phase of the n-th sine wave component, respectively. This principle allows us to understand complex signals as combinations of simpler, pure sine wave components.

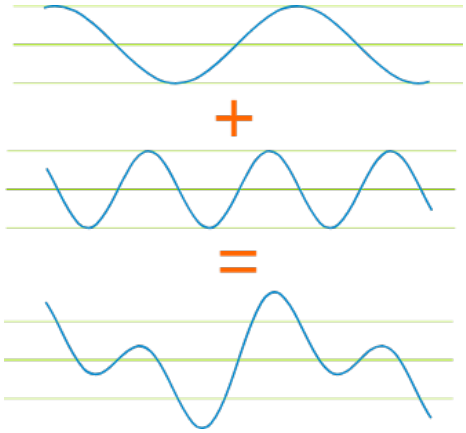

Figure 4: Signal Superposition

Sample Rate Sample Rate, while not a fundamental aspect of sound, is crucial for digital audio processing. It very simply refers to the number of samples taken per second when converting an analog signal into a digital format. So if we have a 3 second clip, and a sample rate of 22050, we would divide our signal into 66150 equally spaced points, and record the amplitude of the signal at those points. All of the audio data we process is digital, and thus it's important to understand this very simple, but crucial, concept.

Having established how sound can be mathematically represented and analyzed as a combination of sine waves and their properties, we can now turn our attention to how these raw audio signals are prepared for use in machine learning systems. Before any meaningful features can be extracted or models trained, the audio must undergo a series of preprocessing steps. These steps—such as trimming silence, segmenting the signal into frames, and normalizing amplitude—ensure that the data is clean, consistent, and suitable for further analysis.

3 Audio Preprocessing

Audio preprocessing is a critical step in preparing raw audio data for machine learning applications, particularly in the context of speaker recognition and embedding extraction. The goal of preprocessing is to transform the raw audio signal into reliable and consistent format that is more suitable for feature extraction and subsequent analysis. This involves several key steps: loading the audio, framing it into manageable segments, applying voice activity detection (VAD) to remove silence, and extracting features that capture the essential characteristics of the speaker's voice. Ultimately the Audio Preprocessing stage will culminate in the extraction of Mel-scale filter banks (FBANKs), which serve as the input to the speaker embedding model.

3.1 Loading in the Audio

The first step in processing raw audio is to load the audio file into a format that can be manipulated. This typically involves reading the audio data from a file, which is often stored in formats like WAV, MP3, or FLAC. We read the audio signal as a one-dimensional array of samples x, where each sample $x[i]$ represents the *amplitude* of the sound wave for a given sample i. The defualt sampling rate in Librosa is 22050 Hz [13]. This sampling rate is a common choice for speech processing tasks, as it captures the essential frequency range of human speech while keeping the file size manageable.

3.2 Framing and hop-length

Framing is the process of segmenting the continuous speech signal into discrete "frames". The primary purpose of framing is to make possible, the analysis of the continuously changing speech signal by breaking it down into smaller, temporally stationary sections. We also use framing when applying Voice Activity Dectection filters (see next section). The process of framing involves slicing the speech into fixed-length segments, typically ranging from 20-40 milliseconds. Often, these segments overlap to maintain continuity between frames, which is also called the hop length. For x-vector extraction we use a frame length of 25 ms and a hop length of 10 ms . At a sampling rate of 22050 Hz, this means that each frame contains $0.025 \times 22050 \approx 551$ samples and the hop length is $0.01 \times 22050 \approx 220$ samples ([3] Section 2.3).

3.3 Voice Activated Detection filter

Speech recordings almost always contain leading or trailing regions in which no one is talking: microphone handling noise, room ambience, or the half-second of silence that precedes the first phoneme while the speaker inhales. Due to this we apply a energy-based *voice-activity detection* (VAD) filter before moving to feature extraction. The general principle of a VAD is to base the trimming on

the amplitude level (as opposed to energy) of individual frames relative to the highest amplitude observed across the entire signal.

If we let L be our frame length, and x_n be the n^{th} frame, the peak energy of each frame A_n is defined as:

$A_n = max_{i \in [0,L)} |x_n[i]|$

We then identify the maximum amplitude over all frames A_{max}. A frame is retained only if its peak amplitude satisfies the following decibel condition:

$20 \log_{10} \left(\frac{A_n}{A_{max}} \right) > -40$

Practically we us librosa to apply this filter (see https://librosa.org/doc/main/generated/librosa.effects.split. The only issue left is that the VAD filter removes every frame that does not meet the decibel condition, which can lead to the removal of frames that are actually part of the speech signal. To mitigate this, we identify the first and last frames that meet the decibel condition and retain all frames in between. This ensures that we do not lose any non-speech, speaker identifying information like natural pauses or breaths, which would otherwise be lost.

This returns us a new speech signal x_{vad} (but we will simply refer to it as x from here on out) which is trimmed of leading and trailing silence, while keeping all frames that are part of the speech signal. This is crucial for ensuring that the subsequent feature extraction process focuses solely on the relevant parts of the audio, enhancing the accuracy and efficiency of the speaker recognition system.

3.4 Connecting Time-Domain Preprocessing to Frequency Analysis

Once the raw audio has been framed and filtered through VAD, the next goal is to extract frequency-based characteristics that are more closely tied to how humans perceive sound. While the time-domain signal reveals how amplitude changes over time, it does not explicitly expose pitch, formants, and other spectral features that are essential for distinguishing between speakers.

4 Feature Extraction

The perception of pitch plays a pivotal role in how we understand and interact with sound. Analyzing this characteristic requires transforming speech signals from their time domain into the frequency domain. To capture these speaker-relevant features, we apply the Short-Time Fourier Transform (STFT) to each frame. However, before performing this transformation, a *windowing function* is applied.

4.1 Windowing

Windowing is a crucial step in the pre-processing of speech signals, employed to mitigate the issue of spectral leakage. Spectral leakage occurs when the abrupt beginnings and endings of signal frames introduce spurious frequency components that were not present in the original signal. This can significantly distort the frequency analysis of the speech signal.

To address this problem, a windowing function is applied to each frame of the signal. The windowing function smooths the transitions at the edges of the frames, thereby reducing the introduction of artificial frequency components and preserving the integrity of the original speech signal. Figure 5 illustrates a signal processed without a windowing function, showing the abrupt transitions that windowing aims to eliminate.

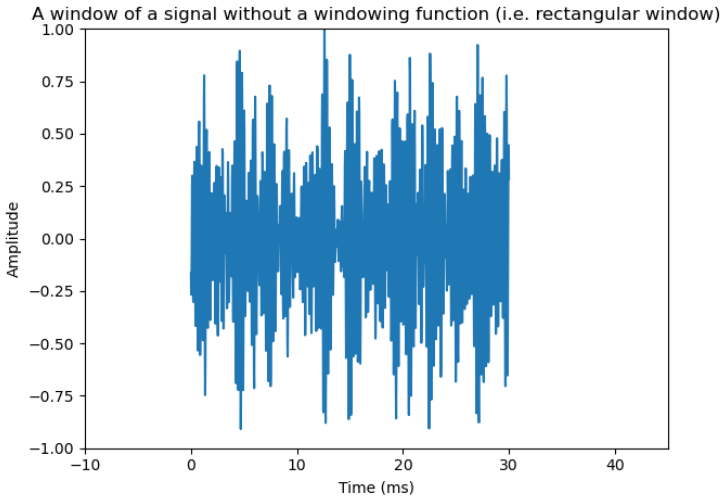

Figure 5: Rectangular Windowing

Several windowing functions have been developed to address the issue of spectral leakage, each designed to smoothly taper the signal at the frame boundaries, thereby reducing spectral leakage. For x-vector extraction we use Hann 6 windowing [2].

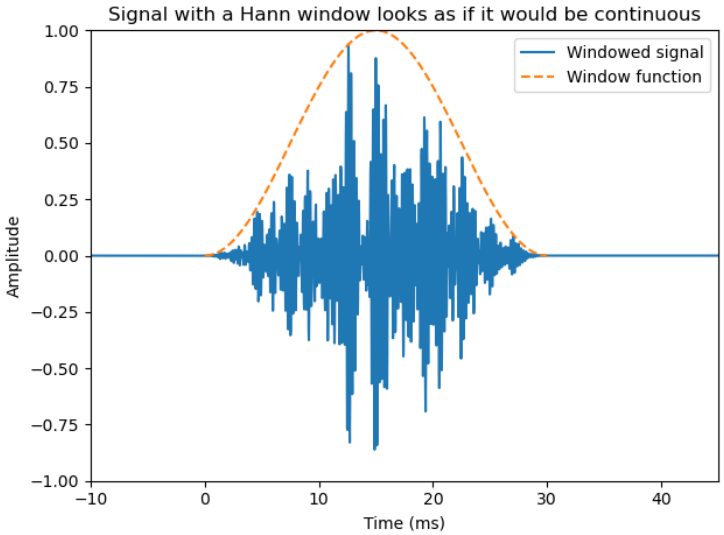

Figure 6: Hann Windowing

Given an input of x which has been framed into N frames x_n, and a windowing function $w[i]$, where L is the length of the frame. The window function is expressed as:

$$\tilde{x}_n[i] = x_n[i] \cdot w[i] \text{ for } i \in [0, L)$$

The Hann windowing function specifically would be $w[i] = [\sin(\pi i/L)]^2$.

This mathematical treatment ensures that each frame of the signal is smoothly tapered at its edges, thereby minimizing spectral leakage and preserving the integrity of the signal's frequency content for accurate analysis. To extract features from speech that are relevant to speaker identity, we must first convert the raw audio from the time domain (amplitude over time) to the frequency domain (energy at different frequencies). This process reveals important characteristics such as pitch, timbre, and formants.

This transformation is accomplished using the Short-Time Fourier Transform (STFT), which relies on the principles of the Discrete Fourier Transform (DFT).

4.2 Short Time Fourier Transform

The Fourier Transform is a mathematical transformation used to analyze the frequencies contained within a signal. It's based on the principle that any signal, no matter how complex, can be decomposed into a sum of simple sine waves of various frequencies, amplitudes, and phases. This decomposition is crucial for understanding the frequency content of signals.

Discrete Fourier Transform (DFT) In order to apply the Fourier Transform to our time signal, we need to learn more about the Discrete Fourier

Transform (DFT). The DFT allows us to express a discrete-time signal $x[i]$ as a sum of sinusoids of different frequencies, by decomposing the signal into its constituent sine waves.

The Discrete Fourier Transform (DFT) of a discrete signal $x[i]$, where $i \in [0, N)$, is defined as:

$$X[k] = \sum_{i=0}^{N-1} x[i] \cdot e^{-j2\pi \frac{ki}{N}}$$

where:

- $X[k]$ is the DFT coefficient at frequency index k,
- $x[i]$ is the i-th time-domain sample of the signal,
- N is the total number of samples,
- j is the imaginary unit ($j^2 = -1$),
- $e^{-j2\pi \frac{ki}{N}}$ is the complex sinusoidal basis function (DFT kernel).

This produces a one-dimensional array of complex numbers, where each element represents the contribution of a specific frequency in the signal.

While the DFT is invaluable for signals that are stationary, many real-world phenomena, such as speech or music, are dynamic. For such signals, the frequency content at one moment might differ significantly from another, making a single DFT insufficient for capturing the signal's time-varying nature. We need a tool that can provide a frequency analysis that evolves as the signal does—a time-frequency representation.

The Short-Time Fourier Transform (STFT) extends the Discrete Fourier Transform (DFT) to non-stationary signals such as speech by analyzing them over localized, overlapping frames of time. Rather than applying the DFT to the entire signal at once, we first divide the signal into overlapping frames $x_n[i]$, and apply a windowing function $w[i]$ to each frame resulting in a sequence of windowed signals $\tilde{x}_n[i]$. By taking the DFT of each windowed frame, we obtain a time-localized spectral representation of the signal — the STFT.

The STFT of a signal $x[i]$ is given by:

$$X_n[k] = \sum_{i=0}^{L-1} \tilde{x}_n[i] \cdot w[i] \cdot e^{-j2\pi \frac{ki}{L}}$$

where:

- $X_n[k]$ is the STFT coefficient for frame n at frequency bin k,
- $x_n[i]$ is the i-th sample of the n-th frame,
- $w[i]$ is the windowing function (e.g., Hann window),

- L is the length of each frame,
- j is the imaginary unit, and
- $e^{-j2\pi\frac{ki}{L}}$ is the complex sinusoidal kernel of the DFT.

The result is a two-dimensional time-frequency representation of the signal, where the horizontal axis corresponds to time (via frame index n), and the vertical axis corresponds to frequency bin k. Each value $X_n[k]$ reflects the contribution of frequency k at time segment n, capturing how the spectral content of the signal evolves over time.

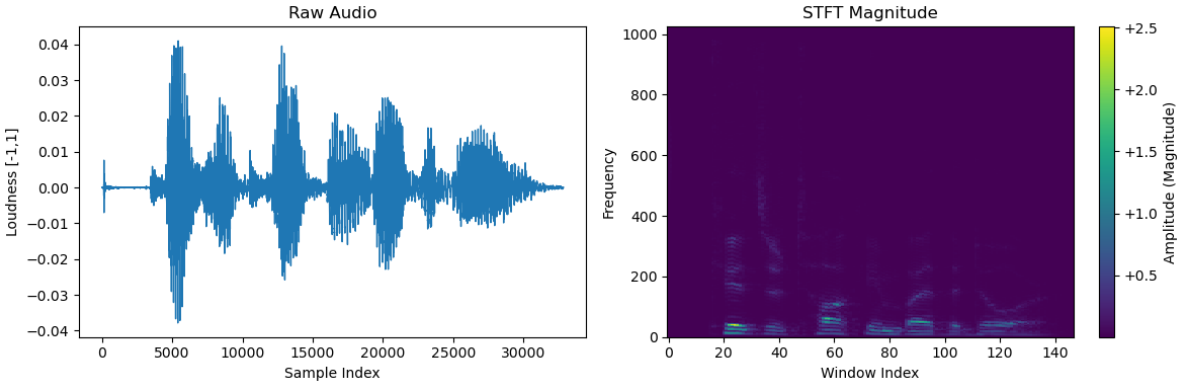

Figure 7: Raw Audio Signal and it's STFT counterpart [14]

Power Spectrum The last step before feature extraction is to convert the complex-valued output of the STFT into a more useful representation. This is done by computing the power spectrum.

The power spectrum $P_n[k]$ for frame n and frequency bin k is computed by taking the squared magnitude of the complex STFT coefficients:

$$P_n[k] = |X_n[k]|^2 = X_n[k] \cdot X_n^*[k]$$

where:

- $X_n[k]$ is the STFT coefficient for frame n and frequency bin k,
- $X_n^*[k]$ denotes the complex conjugate of $X_n[k]$,
- $P_n[k]$ is the power at frequency k for frame n.

The resulting power spectrum $P_n[k]$ is a real-valued matrix that provides a clearer representation of how the signal's energy is distributed across frequencies over time.

4.3 Mel-Scale Filter Banks

After transforming the time-domain signal into a frequency-domain representation using the Short-Time Fourier Transform (STFT), we apply a series of perceptually motivated transformations for the following reasons:

1. Pitch perception is logarithmic with respect to frequency. To reflect this, we map the linear frequency axis to the Mel scale, which spaces frequencies more densely where humans are more sensitive.

2. Loudness perception is logarithmic with respect to amplitude or power. To account for this, we take the logarithm of the power spectrum, converting energy into decibel units.

3. Spectrograms contain more detail than needed for speech modeling. To reduce redundancy, we apply a triangular filter 8 that compresses the spectrum into 24 bins, preserving perceptually important information.

One can also visually see point 1 and 2 in figure 7, where the STFT Magnitude Spectrum is shown. Here we see that Most of the energy is concentrated in the lower frequencies. The process by which we resolve these issues known as Mel-scale filter bank, or FBANK, extraction.

4.3.1 Filter Bank Extraction

As seen above, the spectrogram provides a 2D view of how energy is distributed across frequencies over time. To prioritize perceptually important frequencies, we apply a triangular filter bank to the power spectrum generated from each frame of the STFT.

This triangular filter bank consists of a series of overlapping triangular filters, each centered on a specific frequency where it comes to a point. Each filter spans a range of frequencies, and the height of the filter at each frequency corresponds to the weight assigned to that frequency in the final feature representation. In Figure 8, we see a simplified visual example of a filter bank. Each triangular filter is applied to the power spectrum by multiplying its weights with the spectral energies, and summing the result to produce a single scalar energy per filter. This process is repeated for all filters, resulting in a lower-dimensional, perceptually relevant representation of the audio signal.

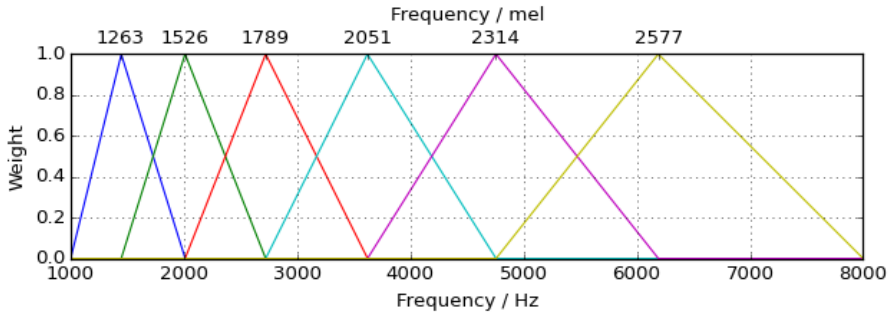

Figure 8: Example of a filterbank [**?**]

We apply this filter using Librosa's `librosa.feature.melspectrogram` function, which automatically generates and applies the triangular filters based on the desired number of Mel bins and frequency range. The key steps involved are:

- Convert the frequency axis from linear (Hz) to the Mel scale using:

$$f_{\text{mel}} = 2595 \log_{10}\left(1 + \frac{f}{700}\right)$$

- Divide the Mel scale range evenly into a set number of bins (e.g., 24 in our case).
- Map these Mel-scaled frequencies back to their corresponding values in Hz to determine filter boundaries.

$$f = 700\left(10^{\frac{f_{\text{mel}}}{2595}} - 1\right)$$

- Use these mapped frequencies to construct a series of overlapping triangular filters. Each filter spans three points: a start frequency, a center frequency, and an end frequency. The filters are defined such that each frequency bin within their range contributes a weight between 0 and 1.
- Apply each Mel filter to the power spectrum of every STFT frame by taking a weighted sum over the linear frequency bins k. Specifically, for each Mel bin m, the filter bank energy is computed as:

$$P_n[m] = \sum_k H_m[k] \cdot P_n[k]$$

where $P_n[k]$ is the power at linear frequency bin k for frame n, and $H_m[k]$ is the weighting function from the triangular filter.

This transformation compresses the frequency axis, emphasizing lower frequencies and de-emphasizing higher ones, which aligns with human auditory perception. The result, as seen on the left side of Figure 9, is a sequence of 24 filter bank energies per window.

4.3.2 Log-Scaled Mel Filter Bank Energies

As mentioned earlier, human hearing perceives loudness on a logarithmic scale. To reflect this perceptual property and compress the dynamic range of the Mel filter bank features, we convert the power values to a logarithmic (decibel) scale using Librosa's `power_to_db` function. This transformation helps prevent large variations in amplitude from dominating the learning process in downstream neural networks.

The decibel-scaled Mel power $L_n[m]$ for frame n and Mel bin m is computed as:

$$L_n[m] = 10 \log_{10} \left(\frac{P_n[m]}{P_{n,\max}} \right)$$

where:

- $P_n[m]$ is the power in Mel bin m for frame n,
- $P_{n,\max}$ is the maximum Mel bin power in that frame, used for normalization,
- $L_n[m]$ is the resulting log-scaled value in decibels (dB).

This log transformation not only matches human loudness perception but also stabilizes the feature range for more effective learning.

4.3.3 Log Mel Filter Bank Normalization

To further enhance the robustness of the features, we apply a normalization step to the log Mel filter bank energies. This normalization is crucial for reducing variability across different recordings and ensuring that the features are more consistent and comparable across different speakers and recording conditions. The normalization is performed by subtracting the mean and dividing by the standard deviation of the log Mel filter bank energies across all frames. This process is often referred to as *mean-variance normalization* or *z-score normalization*:

The normalized feature $\hat{L}_n[m]$ is computed as:

$$\hat{L}_n[m] = \frac{L_n[m] - \mu_m}{\sigma_m}$$

where:

- $\hat{L}_n[m]$ is the normalized log Mel energy,
- μ_m is the mean of $L_n[m]$ over all frames n,

- σ_m is the standard deviation over all frames n.

This normalization ensures that the features for each Mel bin are centered around zero and have unit variance, improving the stability and convergence of learning algorithms.

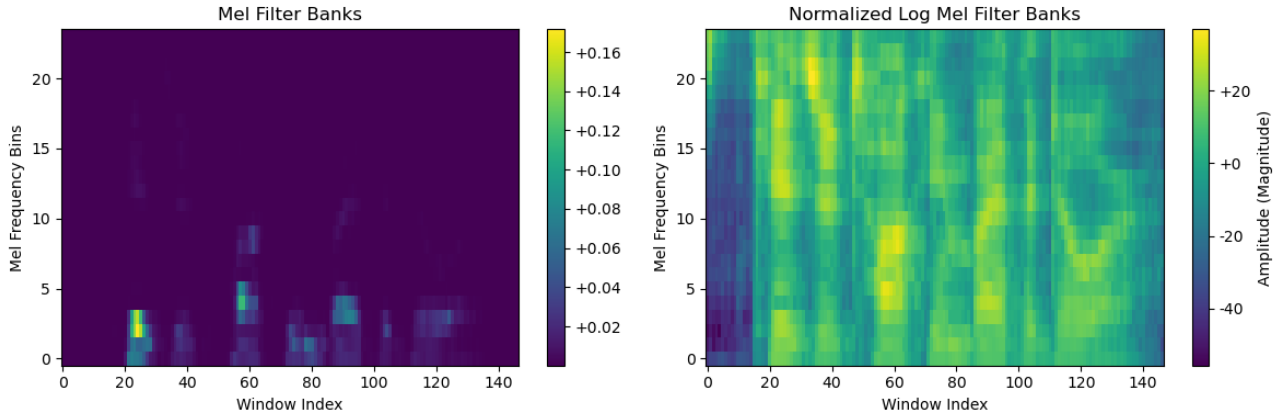

Figure 9: Mel Filter Banks vs Log Normalized Comparison

The final output of this feature extraction process is a sequence of normalized log Mel filter bank energies (see Figure 9), which serve as the input features for the speaker embedding extraction network.

The final output is a matrix of shape (24, T) where T is the number of frames.

4.3.4 Practical Implementation Details

In our codebase, we compute FBANK features using the following steps:

1. Apply Voice Activity Detection to isolate voiced segments.
2. Compute the STFT using a frame length of 551 samples and hop length of 220 samples, using a Hann window.
3. Apply 24 Mel-scaled triangular filters to the power spectrum.
4. Convert to log scale using with reference to the maximum.
5. Normalize the log Mel filter bank energies by subtracting the mean and dividing by the standard deviation across all frames.
6. Return the normalized log Mel filter bank energies as the final feature representation.

This pipeline is aligned with Snyder et al. (2018), ensuring the extracted features are suitable for speaker recognition.

4.4 Conclusion

Mel-scale filter banks provide a compact, perceptually motivated representation of speech that emphasizes frequencies relevant to speaker identity. These perceptually grounded features, when fed into time-delay neural networks (TDNNs), help generate robust speaker embeddings that retain key identity traits and outperform more traditional features like MFCCs in speaker recognition tasks.

5 Convolutional Neural Networks (CNNs)

Having extracted normalized log Mel filter bank (FBANK) features, we now possess a compact, perceptually meaningful representation of the speech signal.

To move from these low-level features to robust, discriminative speaker representations, we require models capable of learning complex patterns, expecially from variable input data. This is where deep learning—and specifically, Convolutional Neural Networks (CNNs)—becomes essential. Although they fall short in working with variable input, even smiple CNNs are designed to automatically learn hierarchical feature representations from input data, making them well-suited for processing structured signals like images and, with appropriate adaptation, audio feature sequences.

When we expand CNNS into the temporal domain, using the Time Delayed Neural Network (TDNN) architecture, we can effectively capture the temporal dependencies in the input data. This allows us to learn long-term speaker characteristics from the FBANK features, which are crucial for generating robust speaker embeddings.

5.1 Kernel Convolution

Let's learn about the first part of the Convolutional Neural Network; the Convolution. Convolutions are a foundational mathematical operation in the field of image processing, often used for tasks such as blurring, sharpening, and edge detection in images. At its core, a convolution involves taking the original image and applying a filter (or kernel) to produce a transformed version of the image.

The Convolutions include the following basic steps:

1. **Filter/Kernel**: This is a small matrix used to apply effects like blurring, sharpening, edge enhancement, and more. This kernel can be many different things like a basic pairwise multiplication (as shown below).

2. **Sliding Window**: The filter is applied over the image by sliding it across every possible position of the image. At each position, the filter overlays a corresponding part of the image.

3. **Element-wise Multiplication and Sum**: For each position of the filter over the image, element-wise multiplication is performed between the part of the image under the filter and the filter itself. The results of these

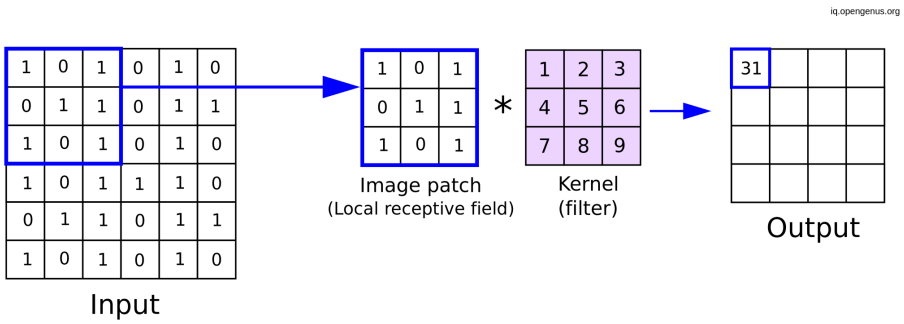

Figure 10: Kernel example

multiplications are then summed up to get a single output pixel. This operation is repeated across the entire image.

5.1.1 Architectural Decisions

There are a few options we have to change the way that the Convolution transforms our input:

1. **Stride**: This defines how much the filter moves across the image. A stride of 1 moves the filter one pixel at a time, while a stride of more than 1 skips pixels, leading to a smaller output image.

2. **Padding**: Sometimes, padding is added around the edge of the original image. A common option is zero padding. This allows the filter to be applied to the border pixels of the image which helps control the spatial size of the output image. Without some sort of padding, our output image would always be smaller than the input.

3. **Kernel Size**: We can increase the scope of our convolution by integrating over a larger area. Instead of a kernel size of 3x3 like in the above example, we can increase the size, usually by odd numbers, to 5x5, or 7x7.

4. **Dilation**: One of the drawbacks of increasing our kernel size is that we increase our computational requirements. This can usually be mitigated through *dilation*, where we set some portion of the kernel to zero. In the image below we see an example of dilation. In the first example, a 3x3 kernel with dilation of 1 means no dilation. In the second example, we have a 5x5 kernel with dilation of 2, and the third is a 7x7 kernel with dilation 3.

5. **Depth**: Depth refers to the number of channels in the input data. This aspect of a CNN's architecture is dictated by the nature of the input,

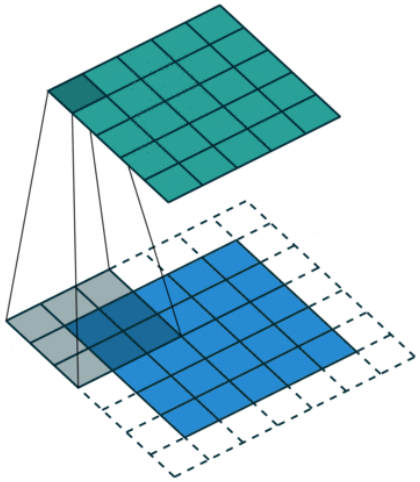

Figure 11: Padding example

rather than a parameter that can be freely adjusted. For example, a grayscale image has a depth of 1, representing the single intensity channel. In contrast, a color image typically has a depth of 3, corresponding to the three channels in the RGB color model—Red, Green, and Blue. In the case of an RGB image, we would, for example, use a 3x3x3 kernel.

5.2 Onto CNNs

Continuing from the discussion of convolution operations, CNNs extend these principles into deeper architectures specifically designed for tasks like image and audio processing. A CNN typically comprises several layers of convolutions, interspersed with activation functions, pooling layers, and fully connected layers towards the end. Let's introduce these parts, and then look at an example of a full Convolutional Neural Network.

5.2.1 Pooling Layers

Pooling layers reduce the dimensionality of the data passing through the network, which decreases computational requirements and helps prevent overfitting. The most common form of pooling is max pooling, where the maximum value from a group of pixels in the feature map is selected. Below is an example of a 2x2 max-pooling layers with stride 2.

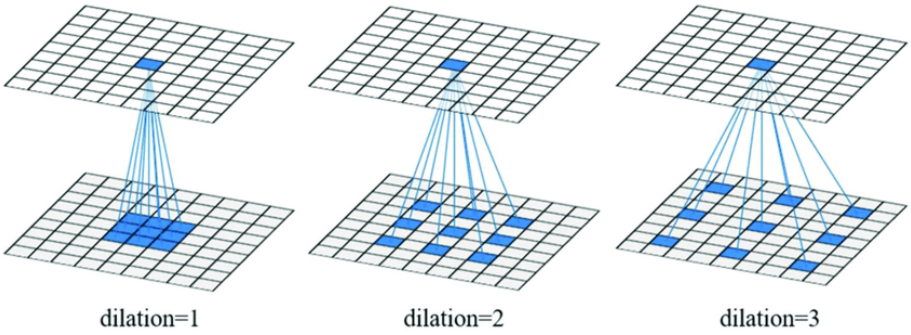

Figure 12: Dilation example

5.2.2 Fully Connected Layers

After several convolution and pooling layers, CNNs often conclude with one or more fully-connected layers. These layers connect every neuron in one layer to every neuron in the next layer, essentially combining all learned features from the previous layers to determine the final output, such as the classification of an image.

5.2.3 Network Architecture

Let's give a full end-to-end account of a Classification Convolutional Neural Network. For this example, let us take the input of a 100x100x1 (Width x Height x Depth). This image is input into the first Convolutional layer. The convolutional layer has (for the sake of example), 10 nodes. Each node will have a randomly initialized filter or kernel of a certain size (let's use a 5x5 filter for our example), this filter will attempt to find some certain features in the image. Each of the nodes will have an output, called a feature map. This feature map will generally have an output size dictated by the following formula:

$$\text{Output} = \left[\frac{n + 2p - f}{s} + 1\right]$$

where n is the side length of the input image, and f is the side length of the filter. These would be the same for a nxn image, and would be two different calculations for a nxm image where $n \neq m$. p is the padding, and s is the stride. So, if we had a filter size of 5x5, with let's say no padding, and a stride of 1, since both our Input image and kernel are square we only do one calculation, which yields:

$$\frac{100 + 2(0) - 5}{1} + 1 = 96$$

Our convolutional layer would yield 10 96x96 feature maps.

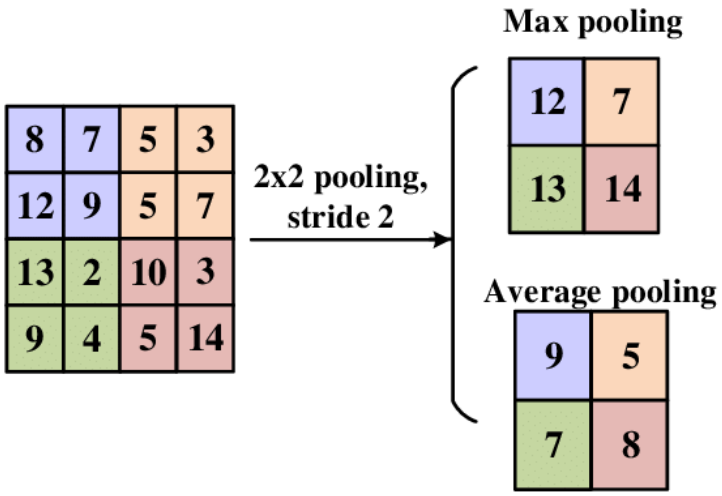

Figure 13: Pooling Layer Examples

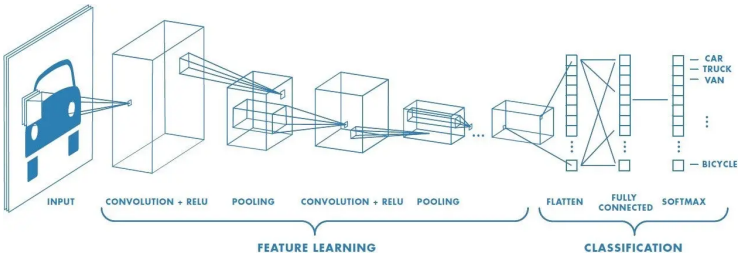

Figure 14: Network Architecture

The next step would be to implement some sort of activation. A very common activation function would be a ReLu activation function which is defined as $f(x) = \max(0, x)$. This means that it would iterate over every single pixel in every single feature map and apply this activation function.

After the activation function is applied, we apply a pooling layer. This pooling layer functions as an aggregator of the features of the previous layers, and also, crucially, a way to reduce the spatial size (i.e., width and height, not depth) of the feature maps. Max pooling is a common technique used to perform this dimensionality reduction, where the maximum element from the rectified feature map within a certain window is taken and the rest are discarded.

5.2.4 Issues with applications in Audio

Applying CNNs to images is straightforward and intuitive because images are naturally represented as $n \times n$ grids. However, when working with audio data,

we face an additional challenge. The challenge is that Mel-Frequency Cepstral Coefficients (MFCCs) are represented as vectors rather than $n \times n$ images. Therefore, the convolution operations need to be adapted from handling spatial, to handling temporal inputs.

6 Time-Delay Neural Networks (TDNNs)

Speech is inherently temporal. For speaker recognition, understanding how speech evolves over time is crucial - static frame-level information alone is insufficient. To adress this challenge we introduce the Time-Delay Neural Network (TDNN) Architecture which changes the model at the Frame Layer. TDNNs solve this by learning temporal features over increasingly larger contextual windows. This approach also allows the network to accept variable-length inputs, unlike traditional CNNs, which apply spatial convolutions over two-dimensional data (like images), TDNNs apply one-dimensional temporal convolutions over time series. This makes them especially suited for tasks like speaker recognition, where capturing local and long-range temporal patterns is essential.

To understand how TDNNs work, we can draw an analogy to CNNs. In a CNN, we apply a convolutional kernel to an image, iterating through the spatial dimensions (width and height) of the image. In contrast, TDNNs use 1D convolutions that move along the time dimension of a sequence, processing sequential frames of audio data. This allows TDNNs to learn temporal features by aggregating information from multiple time steps, similar to how CNNs aggregate spatial features from neighboring pixels.

Consider the architecture diagram in Figure 15. This figure illustrates the TDNN with its layers and the corresponding temporal contexts:

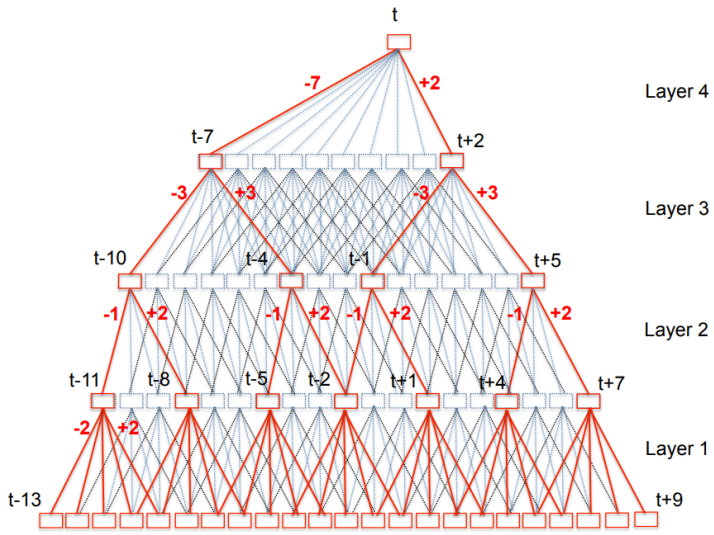

Figure 15: Time-Delay Neural Network (TDNN) Architecture

Explanation of the TDNN Layers: Frame-Level Layers:

1. **Frame 1**: Processes frames with a context of $t-2$ to $t+2$, giving a total context of 5 frames. This is similar to how a convolutional layer would

process an image patch.

2. **Frame 2**: Takes the output from Frame 1 and processes it with a context of $t-2$ and $t+2$ again, with an effective dilation of 2. Thus expanding the context to 9 frames.

So much in the same way that a CNN applies a filter to an image, the TDNN applies a filter to a sequence of frames. And this is what allows it to learn temporal features over time, which is crucial for our task of speaker recognition.

There is one more important aspect to understand about the TDNN, which is that it is not a simple 1 dimensional convolution. Given that our input are $T \times 24$ filter bank features not just a simple array, as might be assumed when thinking of a true 1 dimensional convolution. Rather, the input to the TDNN is a sequence of frames, each frame being a vector of filter bank features.

The term 1 dimersional convolution in this context refers to the fact that the convolution is applied across the temporal dimension of the sequence; it only goes in one direction (temporally), instead of being applied horizontally and vertically like in a 2D convolution.

7 The X-Vector Speaker Embedding System

Now that we have a thourough understanding of all of the basic pieces of the Speaker Embedding systems, we can begin to put the pieces together and fully understand the x-vector speaker embedding process.

In practical , the Speaker Encoder is a deep neural network trained to produce speaker embeddings from raw audio inputs. Specifically, the 2017 refinement of the x-vector architecture, which became the standard in 2018, introduced a modular and more effective structure.

In our system, we adopt this same modular strategy — focusing solely on the embedding extraction component. Rather than training for speaker verification using PLDA or contrastive loss, we treat the model as a feature extractor, tuned to capture speaker-discriminative information from preprocessed speech segments.

This implementation, while faithful to the x-vector framework, leaves us open for a wider set of usecases. Therefore, we omit the backend PLDA classifier and instead emphasize the encoder's ability to generate robust, expressive speaker representations. The following sections explain, in technical detail, how this encoder operates — from temporal convolution with TDNN layers to the aggregation and projection of speaker characteristics into a fixed-length vector.

7.1 Input

The input to the model are the 24 dimnsional f-bank features we discussed in the previous section. These features are extracted from the audio signal, and are then passed into the model as a sequence of frames. We use a different dataset

that the 2018 paper, so while the final output of softmax layer is 4733 speakers, our dataset only contains 24 speaker and thus will only output 24 speakers.

7.2 Architecture

We began to discuss the architecture of the x-vector in the previous section, but now we will go into more detail about the specific layers and how they interact with each other. The x-vector architecture consists of several layers, each designed to capture different aspects of the input features. The architecture is summarized in Table 1.

7.2.1 Frame-Level Layers

The first five layers of the network are Time-Delay Neural Network (TDNN) layers, implemented as one-dimensional convolutions.

Given $\mathcal{C}_\ell$ defines the context window for layer ℓ (e.g., $\{-2, -1, 0, 1, 2\}$ for the first layer), These layers capture increasingly broad temporal contexts:

$$\mathbf{h}_t^\ell = f\left(\sum_{\delta \in \mathcal{C}_\ell} \mathbf{W}_\delta^\ell \cdot \mathbf{h}_{t+\delta}^{(\ell-1)} + \mathbf{b}^{(\ell)}\right)$$

Here:

- $\mathbf{h}_t^\ell$ is the output vector at time t from layer ℓ.
- $\mathbf{W}_\delta^\ell$ is the weight matrix associated with offset δ.
- $\mathbf{b}^\ell$ is a bias vector.
- $f(\cdot)$ is the ReLu activation functions as per Snyder et al. (2018) [3].

The layers are defined as follows:

- **TDNN Layer 1:** Uses $\mathcal{C}_1 = \{-2, -1, 0, 1, 2\}$, i.e., kernel size 5, no dilation. This is similar to splicing five consecutive frames together. The input is a sequence of 24-dimensional log-Mel filterbanks, resulting in a feature vector of size $5 \times 24 = 120$. These features are processed through a fully connected (dense) layer, which has an output of 512 features.
- **TDNN Layer 2:** Uses $\mathcal{C}_2 = \{-2, 0, 2\}$, i.e., kernel size 3, dilation 2. Because the previous layer outputs 512 features per frame, this spliced input has $3 \times 512 = 1536$ dimensions, again mapped to 512-feature vector through a dense layer.
- **TDNN Layer 3:** Expands the context to $\mathcal{C}_3 = \{-3, 0, 3\}$, i.e., kernel size 3, dilation 3.. Like Layer 2, it processes 1536 features and outputs 512.

- **TDNN Layer 4 and 5:** These use $\mathcal{C}_4 = \mathcal{C}_5 = \{0\}$, i.e. point-wise 1D convolutions (kernel size = 1), operating at the frame level. Layer 4 retains the 512-feautre output, while Layer 5 expands it to 1500 features. This final frame-level representation encodes a rich temporal context of 15 frames.

In Practice, we implement these using TensorFlow's 'tf.keras.layers.Conv1D' with appropriate parameters for kernel size, dilation, and ReLu activation function. While it is not explicitly stated in the paper that these layers use bias terms, we will follow the conventions of Kaldi's x-vector implementation, which does include bias terms in each layer [6]. These are handled internally by TensorFlow's Conv1D layer.

Layer	Layer context	Total context	Input x output
frame1	{t - 2,t - 1,t, t + 1, t + 2}	5	120x512
frame2	{t - 2, t, t + 2}	9	1536x512
frame3	{t - 3, t, t + 3}	15	1536x512
frame4	{t}	15	512x512
frame5	{t}	15	512x1500
stats pooling	{0, T}	T	1500x3000
segment6	{0}	T	3000x512
segment7	{0}	T	512x512
softmax	{0}	T	512xN

Table 1: Neural Network Architecture

7.2.2 Statistics Pooling

The statistics pooling layer serves to transform a variable-length sequence of frame-level outputs into a fixed-length segment-level representation.

Suppose the output from the final TDNN layer (Layer 5) is a tensor:

$$\mathbf{H} = [\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_T] \in \mathbf{R}^{1500 \times T}$$

where T is the number of frames, and each $\mathbf{h}_t \in \mathbf{R}^{1500}$ is a feature vector window index t.

We compute two summary statistics over time: the mean and the standard deviation:

$$\boldsymbol{\mu} = \frac{1}{T}\sum_{t=1}^{T}\mathbf{h}_t \quad \text{and} \quad \boldsymbol{\sigma} = \sqrt{\frac{1}{T}\sum_{t=1}^{T}(\mathbf{h}_t - \boldsymbol{\mu})^2}$$

These vectors are concatenated to produce a fixed-length embedding:

$$\mathbf{s} = [\boldsymbol{\mu}; \boldsymbol{\sigma}] \in \mathbf{R}^{3000}$$

This vector $\mathbf{s}$ is then fed into the segment-level layers, which operate globally on the entire utterance.

7.2.3 Segment-Level Layers

Following the statistics pooling layer, the TDNN architecture transitions from frame-level processing to segment-level operations. At this stage, we are no longer dealing with individual time frames, but with a single, fixed-length representation of the entire utterance. Again, this is achieved by concatenating the mean and standard deviation across all frame-level outputs from TDNN Layer 5, resulting in a 3000-dimensional vector.

The segment-level layers are fully connected layers that refine this 3000-dimensional pooled representation into a speaker-discriminative embedding. These layers function globally, operating on the entire utterance to extract characteristics that remain consistent across time—such as vocal tract shape, speaking style, and other speaker-specific traits.

- **Segment Layer 6:** This is a dense layer that reduces the 3000-dimensional pooled vector to 512 dimensions. In Keras, this is implemented as `layers.Dense`.
- **Segment Layer 7:** Another dense layer that maintains the 512-dimensional size, further refining the speaker representation through nonlinear transformation. In Keras, this is handled by another `layers.Dense`.
- **Softmax Layer:** This final layer maps the 512-dimensional representation to the output space of speaker classes. In Keras, this is implemented as `layers.Dense`.

These layers play a critical role in converting the pooled temporal statistics into a compact and expressive speaker embedding. In practice, we extract the output of Segment Layer 6 as the final x-vector, which serves as a robust speaker representation suitable for downstream tasks such as verification, TTS or Voice Cloning.

7.2.4 Segment-Level Dense Layers

This segment-level vector is then passed through one or more dense (fully connected) layers. Each dense layer performs the following operation:

$$\mathbf{z} = f(\mathbf{W} \cdot \mathbf{s} + \mathbf{b})$$

where:

- $\mathbf{s} \in \mathbf{R}^{d_{\text{in}}}$ is the input vector (e.g., $d_{\text{in}} = 3000$),
- $\mathbf{W} \in \mathbf{R}^{d_{\text{in}} \times d_{\text{out}}}$ is the weight matrix,
- $\mathbf{b} \in \mathbf{R}^{d_{\text{out}}}$ is the bias vector,
- $f(\cdot)$ is a non-linear activation function, typically ReLU.

For example, in the x-vector architecture, this process continues through two segment-level layers:

$$\mathbf{z}_1 = f_1(\mathbf{W}_1 \cdot \mathbf{s} + \mathbf{b}_1) \qquad \text{(Segment Layer 6)}$$
$$\mathbf{z}_2 = f_2(\mathbf{W}_2 \cdot \mathbf{z}_1 + \mathbf{b}_2) \qquad \text{(Segment Layer 7)}$$

The output $\mathbf{z}_1 \in \mathbf{R}^{512}$ is typically used as the final speaker embedding — the x-vector. If classification is performed, $\mathbf{z}_2$ is passed to a softmax layer:

$$\mathbf{y} = \text{softmax}(\mathbf{W}_3 \cdot \mathbf{z}_2 + \mathbf{b}_3)$$

This structure allows the model to map the pooled segment-level information into a more compact and speaker-discriminative representation.

7.3 Conclusion

The purpose of this report has been to provide insight into the process by which we extract speaker embeddings from audio signals. We have discussed the various components of the x-vector system, including the audio pre-processing, feature extraction process, and the TDNN architecture used to generate the speaker embeddings. X-vectors marked a pivotal point in speaker verification technology, offering vast improvements in system robustness and error rates. This system set new benchmarks for the field.

This report sets us up to futher explore the applications of speaker embeddings in various domains, such as speaker verification, identification, text-to-speech and voice cloning. The x-vector system's modular design allows for easy adaptation to different datasets and tasks, making it a versatile tool in the field of speaker recognition.

References

[1] https://link.springer.com/article/10.1007/s10462-023-10688-w

[2] Bäckström, T. et. al. 2022, Introduction to Speech Processing - Introduction to Speech Processing 2dn edition. `https://speechprocessingbook.aalto.fi`. 10.5281/zenodo.6821775

[3] Snyder, D., Garcia-Romero, D., Sell, G., Povey, D., and Khudanpur, S. (2018). X-vectors: Robust DNN embeddings for speaker recognition. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5329–5333. IEEE.

[4] Snyder, D., Garcia-Romero, D., Povey, D., and Khudanpur, S. (2017). Deep neural network embeddings for text-independent speaker verification. In *Proceedings of Interspeech 2017*, pp. 999–1003.

[5] Snyder, D., Ghahremani, P., Povey, D., Garcia-Romero, D., Carmiel, Y., & Khudanpur, S. (2016). Deep Neural Network-based speaker embeddings for end-to-end speaker verification. 2016 IEEE Spoken Language Technology Workshop (SLT), 165–170. https://doi.org/10.1109/slt.2016.7846260

[6] https://kaldi-asr.org/doc/_2nnet-component_8cc_source.html#l03900

[7] Sztahó, D., Szászák, G., and Beke, A. (2019). Deep learning methods in speaker recognition: A review. *arXiv preprint arXiv:1911.06615*.

[8] Alim, S. A. and Rashid, N. K. A. (2018). Some Commonly Used Speech Feature Extraction Algorithms. In *From Natural to Artificial Intelligence – Algorithms and Applications*. IntechOpen.

[9] Amezaga, N. and Hajek, J. (2022). Availability of voice deepfake technology and its impact for good and evil. In *Proceedings of the 23rd Annual Conference on Information Technology Education (CITEd '22)*, pages 1–10. ACM. DOI:10.1145/3537674.3554742.

[10] Li, W., Fu, T., and Zhu, J. (2015). An improved i-vector extraction algorithm for speaker verification. *EURASIP Journal on Audio, Speech, and Music Processing*, 2015:18.

[11] An Overview of the Development of Speaker Recognition. (2022). *Journal of Communications*, JOCM. "The Gaussian Mixture Model (GMM) . . . ", "bounded by domain variations . . . " etc.

[12] Variani, E., Lei, X., McDermott, E., Moreno, I. L., & Gonzalez-Dominguez, J. (2014). Deep Neural Networks for small footprint text-dependent speaker verification. 2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). https://doi.org/10.1109/icassp.2014.6854363

[13] Librosa documentation - librosa 0.11.0 documentation. (n.d.). https://librosa.org/doc/0.11.0/index.html

[14] See my own code at https://github.com/Tobyrrr00/X-Vector-Extraction