School of Engineering
CAI Centre for Artificial Intelligence

Bachelor's Thesis

Prosodic Feature Modelling in Transformers for Speaker Verification

Authors:
Fabian Bosshard*
Andrin Fassbind*

Supervisor:
Prof. Dr. Thilo Stadelmann

June 28, 2024

*Equal contribution.

Abstract

Recent advancements in speaker verification (SV) technologies, particularly through deep neural networks (DNNs), have significantly contributed to the field of speaker recognition (SR). However, despite these advancements, challenges persist, particularly in terms of robustness in noisy environments and the effective modeling of supra-segmental temporal information (SST) — attributes such as intonation and stress, which play a crucial role in human speaker recognition capabilities.

This thesis investigates the effectiveness of transformer models, a type of DNNs known for their success in large language models (LLMs) like ChatGPT, in capturing these SST for SV tasks. We hypothesize that by pre-training transformer models to reconstruct hidden frames in speech spectrograms, the temporal nuances essential for accurate speaker recognition can be better captured. Our approach includes the development of a transformer-based SV system that is pre-trained on a task that is expected to capture these SST and subsequently fine-tuned on SV.

Initial experiments demonstrate that models trained on shuffled spectrograms, where temporal information is deliberately obscured, surprisingly do not show the expected decrease in performance. This suggests that these models may not rely on SST as significantly as hypothesized and are capable of achieving robust SV by predominantly learning from frame-based acoustic information (FBA), contrary to initial expectations.

The findings of this research could potentially improve the understanding of the dependencies of DNNs on temporal versus spectral features in SV tasks and contribute to the development of more robust SV systems capable of performing accurately under varied acoustic conditions.

Acknowledgements

We would like to thank our supervisor, Prof. Dr. Thilo Stadelmann, for his excellent guidance throughout our bachelor thesis. His respect for our ideas and his encouragement significantly contributed to our approach to research. Prof. Stadelmann's readiness to answer our queries and his conceptual guidance were fundamental to our work.

We are grateful to Dr. Marc André Stadelmann for his assistance in setting up the necessary infrastructure at the Centre for Artificial Intelligence, which provided us with access to their advanced GPU cluster. His help was crucial for the effective implementation of our project and significantly enhanced our research capabilities.

We would also like to thank Dr. Yuan Gong and his colleagues from the Massachusetts Institute of Technology for publicly releasing the code of their research, answering questions about it as well as sharing detailed performance logs with us. Dr. Gong's responsiveness and openness in discussing implementation details were very valuable to our project.

Special thanks to Daniel Neururer for his advice on optimizing data loading processes, which helped us deepen our understanding of practical data handling in machine learning environments.

Lastly, we thank our families and friends for their support throughout this journey.

Contents

List of Abbreviations

$\mathcal{M}_{\mathrm{o/o}}$	model pretrained on original and finetuned on original spectrograms
$\mathcal{M}_{\mathrm{o/s}}$	model pretrained on original and finetuned on shuffled spectrograms
$\mathcal{M}_{\mathrm{o}}$	model pretrained on original spectrograms
$\mathcal{M}_{\mathrm{s/o}}$	model pretrained on shuffled and finetuned on original spectrograms
$\mathcal{M}_{\mathrm{s/s}}$	model pretrained on shuffled and finetuned on shuffled spectrograms
$\mathcal{M}_{\mathrm{s}}$	model pretrained on shuffled spectrograms
Adam	adaptive moment estimation
CNN	convolutional neural network
DNN	deep neural network
EER	equal error rate
FAR	false acceptance rate
FBA	frame-based acoustic information
FFT	fast Fourier transform
FRR	false rejection rate
InfoNCE	information noise-contrastive estimation
LLM	large language model
MLP	multilayer perceptron
MR	misclassification rate
MSE	mean squared error
OS	original segment
ResNet	residual neural network
RNN	recurrent neural network
SC	speaker clustering
SI	speaker identification
SR	speaker recognition
SS	shuffled within segment
SSAST	self-supervised audio spectrogram transformer
SST	supra-segmental temporal information
SU	shuffled within utterance
SV	speaker verification
ViT	Vision Transformer

1. Introduction

1.1. Motivation

Speaker verification (SV) is a biometric process used to verify the identity of a speaker based on their voice characteristics. It differs from speaker identification (SI), which involves identifying a speaker among a group. In SV, the system compares the incoming speech data against another speech recording of a claimed identity to confirm whether the claim is true. Essentially, the task is binary - confirming or denying the identity claim - making it particularly suitable for access control and authentication purposes. SV and SI are subtasks of the broader field speaker recognition (SR) [1]. Figure 1 shows the difference between the two tasks.

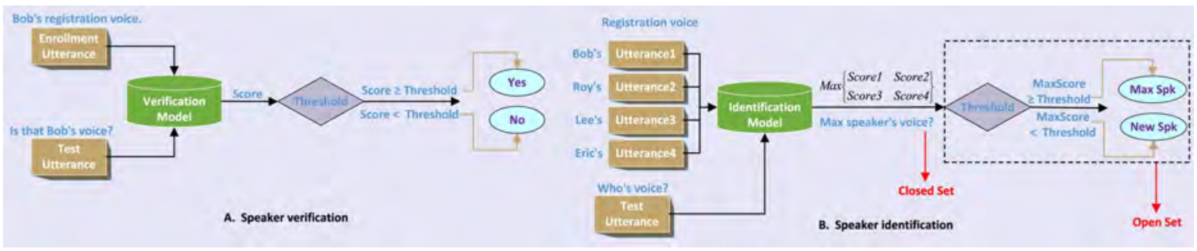

Figure 1: Difference between speaker verification and identification. Image from [2].

With the proliferation of smart devices and voice-activated interfaces such as Siri, Cortana, and Alexa, SV has become an indispensable technology in ensuring secure and personalized user experiences. These systems rely on the unique characteristics of a user's voice to confirm their identity, enabling tasks ranging from sending messages to making purchases. Despite significant advancements, automatic SV systems still face challenges, particularly in noisy or unpredictable environments, where they often struggle to maintain accuracy and reliability [3, 4].

Humans excel in recognizing speakers even in adverse conditions, a skill partly attributable to their adeptness at interpreting prosodic features - subtle variations in rhythm, stress, and intonation that convey important information about the speaker's identity [5, 6]. Similarly, enhancing the capability of machines to understand and utilize these prosodic cues could markedly improve SV systems, especially in challenging scenarios [7]. Henceforth, we refer to this type of information as supra-segmental temporal information (SST), which is distinguished from frame-based acoustic information (FBA), referring to short-term spectral information.

Deep neural networks (DNNs) have become very good at SV [8]. It was therefore often assumed that, similar to humans, they are now able to make use of SST as well [9, 10, 11]. However, recent research shows that at least for some architectures, such as convolutional neural networks (CNNs), recurrent neural networks (RNNs), and residual neural networks (ResNets) this hypothesis needs to be rejected. Specifically it was found that for the aforementioned models, when trained on manipulated data that does not contain any SST anymore, the performance does not decrease as expected, but stays the same or sometimes even increases [12]. This "*deep cheating*" phenomenon is significant as it suggests a potential path to improving the performance, reliability and robustness of SV systems.

This leads us to question whether a different architectural approach could surmount these obstacles. Consequently, this thesis proposes the exploration of transformers [13], renowned for their effectiveness in sequence-based tasks and potential for modeling complex dependencies over time. By investigating these models' ability to utilize SST, we aim to uncover whether they can enhance the accuracy and robustness of SV systems.

1.2. Problem Statement

This thesis aims to investigate the modeling of SST in transformer models, a type of neural network architecture for example associated with large language models (LLMs) like ChatGPT. By pre-training these models to reconstruct hidden frames in speech spectrograms, it is hypothesized that they can better capture the temporal nuances of speech, leading to improved SV accuracy, overall through enhanced capturing of supra-segmental features.

To test these assertions, experiments are conducted with transformers focusing on the capture of SST. Performance comparisons are made between models trained on original speech data and those trained on manipulated data devoid of any temporal information. The outcomes of these experiments will help assess the efficacy of transformers in SV tasks and their ability to robustly model essential speech characteristics.

1.3. Contributions

While not achieving state-of-the-art, we can show that for the transformer architecture (at least when using the configuration described in chapter 3) the same oddity holds true as for the other architectures tested in [12]. Specifically, training with data that includes SST does not improve SV performance; on the contrary, it actually has a detrimental impact on the latter.

1.4. Organization

The rest of this thesis is organized into four chapters and two appendices:

- **Chapter 2: Foundations** covers foundational concepts related to SV on a high level. It also reviews the related work this thesis builds upon.
- **Chapter 3: Methods** contains a detailed description of the methodologies employed in this thesis, focusing on data preprocessing as well as pretraining and finetuning of the model.

- **Chapter 4: Results** presents the results of the conducted experiments as well as a discussion thereof. The pretraining results are presented and the tendency of transformers, when finetuned for SV, to model SST is discussed.

- **Chapter 5: Conclusion** summarizes the findings of the thesis and discusses their implications for future research in SV.

- **Appendix A: Resynthesizing Audio from Spectrograms** describes a problem we encountered when resynthesizing audio from spectrograms and the solution we came up with to mitigate it.

- **Appendix B: More Resynthesized Spectrograms** contains additional resynthesized spectrograms that were developed during our project.

1.5. Implementation

The code developed for this project is publicly available at the following URL:
https://github.com/Ironmomo/SpeakerVerificationBA.

2. Foundations

2.1. Representation of Audio Signals

In SR systems, audio signals are typically represented as sequences of feature vectors. Various methods exist to represent these features, each providing different insights into the audio signal's characteristics and properties.

2.1.1. Raw Waveform

The raw waveform of an audio signal represents the scaled voltage amplitude of the sound wave as it varies over time. This waveform is essentially a direct digital encoding of the air pressure variations caused by sound waves. It is captured through a process of analog-to-digital conversion where the continuous sound wave is sampled at regular intervals (sampling rate) and quantized into digital values.

This representation is one of the most fundamental forms of audio data, providing a time-domain view where each sample point represents the amplitude of the audio signal at a given instant. The sampling rate, typically measured in kilohertz, defines the number of samples captured per second and is crucial in determining the quality and range of frequencies that can be accurately represented in the digital waveform.

More details on the analog-to-digital conversion process can be found in [14] or [15].

A raw waveform can be written as $s \in \mathbb{R}^l$, where $s[n]$ denotes the amplitude of the n-th sample, and l is the total number of samples. The sampling frequency f_s, an important parameter in the analog-to-digital conversion process, dictates the temporal resolution of the waveform. The Nyquist-Shannon sampling theorem states that f_s should be at least twice the maximum frequency component present in the audio signal to adequately capture its frequency content without aliasing [16, 17].

In audio processing and SR systems, the raw waveform often serves as the basis for further transformations and feature extraction methods, which transform the time-domain data into formats more suitable for various analyses and machine learning applications.

2.1.2. Mel Spectogram

Another representation method is a spectrogram, where the frequency composition (y-axis) of the signal is shown as it varies over time (x-axis). A variation of the spectrogram is the Mel spectrogram, in which each frequency of the audio signal is logarithmically scaled to better account for human perception of different frequency bands. The representation as a (Mel) spectrogram offers the advantage of a visual capture of the audio signal. Therefore, image processing models like the neural networks described in Section 1.1 can be used for the algorithmic processing of audio in this representation. For the rest of this thesis, we treat the Mel spectrogram as a matrix X with M rows and L columns, where $X[i]$ denotes the i^{th} column/frame in the spectrogram:

$$X = \begin{bmatrix} | & | & | & | & | & & | \\ X[0] & X[1] & X[2] & X[3] & X[4] & \cdots & X[L-1] \\ | & | & | & | & | & & | \end{bmatrix} \in \mathbb{R}^{M \times L} \tag{1}$$

Figure 2 shows the raw waveform of one sample of the pretraining dataset and its corresponding Mel spectrogram. More details on the conversion of time-domain signals to Mel spectrograms can be found in section 3.2.1.

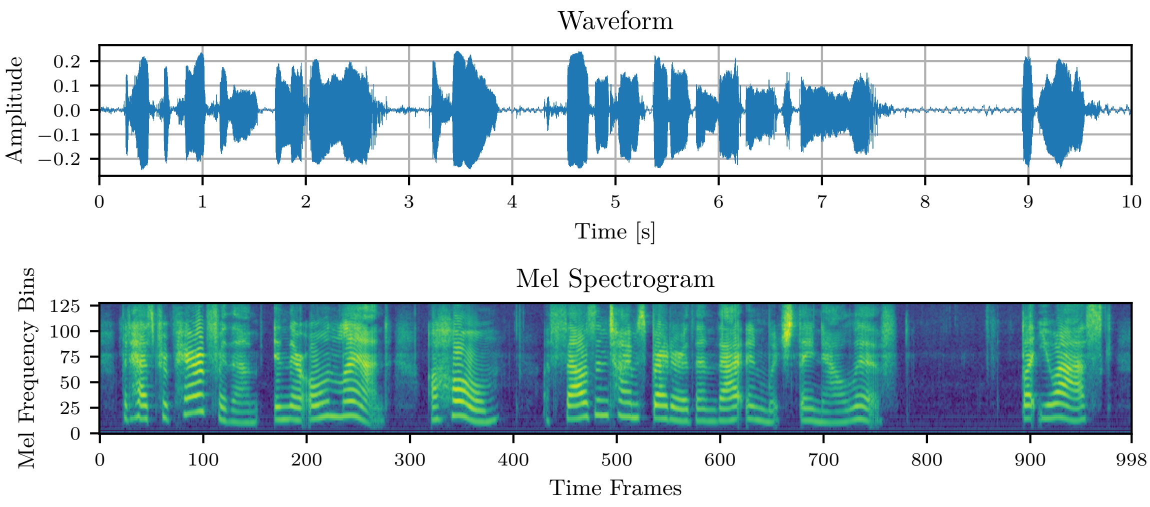

Figure 2: Raw waveform signal (top) and the corresponding mel spectrogram (bottom).

2.2. Linguistic Characteristics in the Context of Speaker Recognition

To better understand the challenges in modeling prosodic features for SR, a detailed analysis of linguistic peculiarities is required. Humans have exceptional abilities in identifying speakers and sub-tasks of SR such as SV. Linguistics has intensively dealt with this topic and has gained important insights. In particular, it has been shown that spectro-temporal acoustic-phonetic information is crucial for the high performance of the human brain in SR [5, 6].

Spectro-temporal acoustic-phonetic information refers to the combination of spectral (frequency-related) and temporal (time-related) properties of speech signals. Spectral information includes the distribution of energy across various frequency bands of the speech signal. This can be used to identify features such as formants, which represent characteristic resonance frequencies in human speech. Formants manifest as peaks in the spectrum and enable the identification of characteristic frequency patterns associated with certain vowels [18]. This is shown in Figure 3.

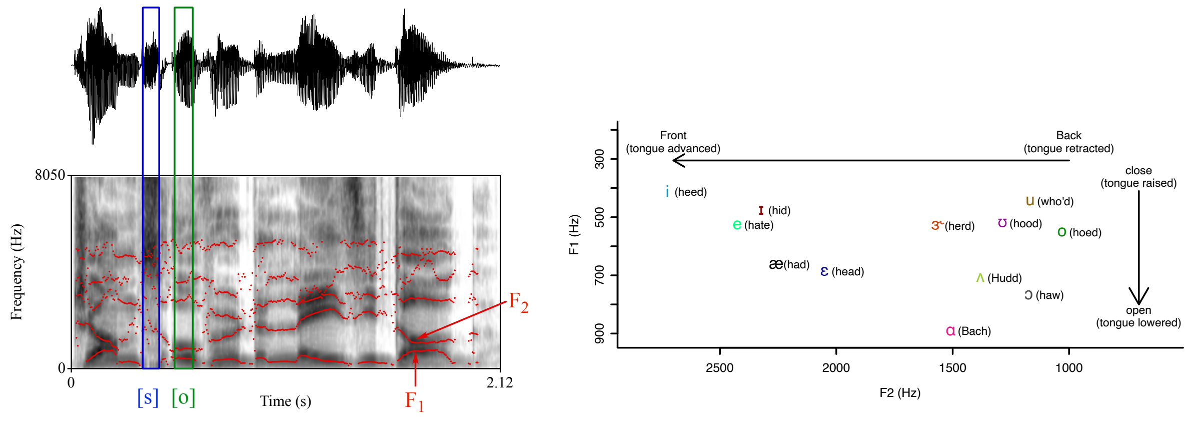

Figure 3: Raw waveform and corresponding spectrogram of a speech recording (left); average vowel formants in the F_1, F_2 vowel space over a set of 139 speakers (right). Image from [19].

Temporal information refers to the sequence of sound events in the speech signal, characterized by features such as stress patterns, speaking rate, and intonation patterns. Stress patterns describe the emphasis of certain syllables, while speaking rate indicates the number of sounds or words per unit of time. Intonation describes the melody and pitch of the language.

The combination of these spectral and temporal information allows for the capture and analysis of complex linguistic features, which is crucial for automatic speech recognition, speech synthesis, and other speech processing applications. In phonetics, it is generally recognized that these features are unique and can thus be assigned to an individual speaker.

2.3. Audio Signal Processing Using DNNs

The basic approach to SR using DNNs is works as follows. The voice recording is transformed into a suitable format. Often, this is a Mel spectrogram, as information not relevant to human perception is filtered out from the audio signal. This reduction in information content can be justified because humans excel in SR, thus the audio information perceivable by the human auditory organ should also be sufficient for a DNN.

The audio signal is then divided into overlapping time windows, which serve as input for the DNN. These windows typically have a length in the range of several tens of milliseconds. This approach offers several advantages: By using segments of the audio signal, the amount of data is reduced, leading to simpler models with lower dimensionality and shorter training time. Moreover, this method makes the model more robust against environmental noise, as it learns local features. These local features contain, as described in section 2.2, spectral information FBA. However, research points out that SST plays a significant role in human SR [7]. To what extent DNN consider SST is discussed in [12].

2.4. Quantifying the Utilization of SST

The mentioned paper, [12], represents the first systematic approach to investigate the modeling of SST in DNN. This section explains the method developed by the authors, which forms the basis and motivation for the present work.

The goal is to investigate whether state-of-the-art DNN, when removing the SST from the spectrogram inputs, are still able to achieve good performance in SR tasks. If this is the case, it can at least be argued that DNN are capable of compensating for the loss of SST and therefore do not necessarily rely on them, but can instead rely on the remaining FBA.

To remove the SST from the spectrogram while preserving the FBA, two approaches have been pursued:

- **shuffled within segment (SS)**: The spectrogram is divided into equal-sized segments, within which the data is swapped on the time axis.
- **shuffled within utterance (SU)**: The spectrogram is also divided into equal-sized segments, but the data is swapped on the time axis before segmenting.

Figure 4 illustrates the described process of swapping spectrogram information.

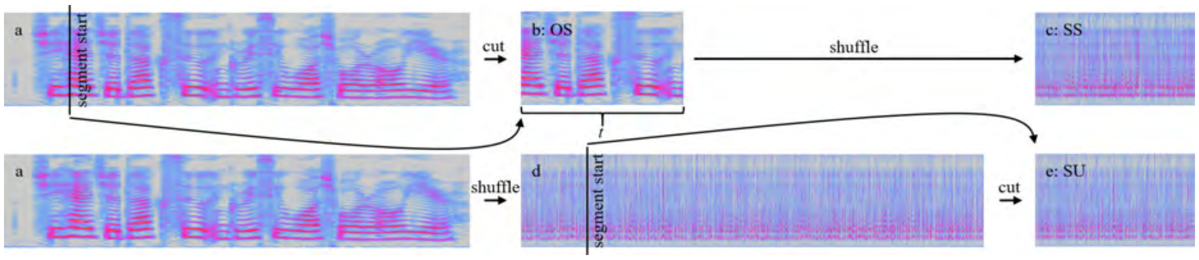

Figure 4: Generation and shuffling of segments (image from [12]).

In [12], various DNN models for SR are tested on different training and evaluation datasets. This includes training on original segment (OS) and evaluation on OS, SS, and SU as well as training on SS and evaluation on OS, SS, and SU, and training on SU and evaluation on OS, SS, and SU. The evaluated architectures include a CNN, a RNN, a ResNet and the F-ResNet. The results of these experiments are shown in Table 1.

In particular, the results show that training and evaluation on swapped data (SS/SS and SU/SU) deliver state-of-the-art performance in speaker clustering (SC) and SV, although the model does not utilize SST. These results cast significant doubts on whether DNNs actually learn SST or whether they rely exclusively on FBA features [12].

If the performance of training and evaluation on the shuffled data were significantly lower than the performance of training and evaluation on the original data, it could be stated that the model relies on both FBA and SST.

The present thesis therefore investigates the performance of Transformers in SV tasks and tries to determine whether they rely on SST.

Table 1: Test results as reported in [12] using original and shuffled spectrograms from the TIMIT dataset using four different DNNs. SC performance (left) is measured using the misclassification rate (MR) (μ/σ) and SV performance (right) is measured using the equal error rate (EER) (μ/σ). Bold font indicates best results per model, cell coloring scales with quality per model.

task $\rightarrow$		Speaker Clustering			Speaker Verification		
training $\downarrow$ / test $\rightarrow$		OS	SU	SS	OS	SU	SS
CNN [20]	OS	**0.00 σ0.00**	9.75 σ0.94	9.00 σ2.15	6.38 σ0.12	12.02 σ0.51	11.90 σ0.46
	SU	8.50 σ2.42	0.50 σ0.61	1.75 σ0.61	8.55 σ0.49	5.55 σ0.06	6.12 σ0.12
	SS	9.00 σ1.66	1.00 σ0.50	1.25 σ0.00	8.16 σ0.42	**5.33 σ0.18**	5.78 σ0.16
RNN [9]	OS	1.25 σ1.12	2.75 σ0.94	2.75 σ0.50	**3.53 σ0.07**	4.19 σ0.09	3.90 σ0.12
	SU	3.75 σ1.37	**0.00 σ0.00**	2.50 σ1.58	3.99 σ0.16	3.78 σ0.10	3.66 σ0.13
	SS	2.00 σ1.00	1.25 σ0.79	0.25 σ0.50	4.00 σ0.07	3.89 σ0.06	3.54 σ0.05
ResNet [21]	OS	**1.00 σ0.94**	8.25 σ4.78	11.50 σ4.29	**4.96 σ0.19**	10.34 σ1.56	9.21 σ1.15
	SU	2.50 σ1.77	**1.00 σ0.50**	3.00 σ1.27	6.59 σ0.25	6.25 σ0.23	6.37 σ0.35
	SS	2.75 σ0.94	1.25 σ1.12	**1.00 σ0.94**	5.89 σ0.25	6.11 σ0.31	5.80 σ0.11
F-ResNet [22]	OS	11.50 σ2.15	37.50 σ4.18	33.75 σ4.18	12.20 σ0.25	23.41 σ1.73	20.47 σ1.50
	SU	16.50 σ2.42	5.75 σ1.70	4.25 σ1.00	15.12 σ0.84	10.46 σ0.28	9.69 σ0.20
	SS	15.50 σ2.57	6.75 σ1.50	**3.75 σ0.79**	15.91 σ0.90	9.86 σ0.11	**8.95 σ0.13**

2.5. Contrastive Loss

Contrastive loss is a metric learning technique widely used in machine learning to measure the similarity between pairs of data points. The idea of contrastive loss is to ensure that similar data points (positive pairs) are closer in an embedding space, while dissimilar data points (negative pairs) are farther apart.

Contrastive loss was introduced in the context of Siamese networks, where the goal is to learn a function that maps input data into an embedding space where the similarity between data points can be easily measured [23]. For SV, it means that intra-speaker embeddings are closer in the embedding space while inter-speaker embeddings are farther apart. The contrastive loss function is described in 3.4.4.

3. Methods

3.1. Overview

Our experimental framework utilizes the self-supervised audio spectrogram transformer (SSAST) model developed by [24]. The architecture of this model is depicted in Figure 5.

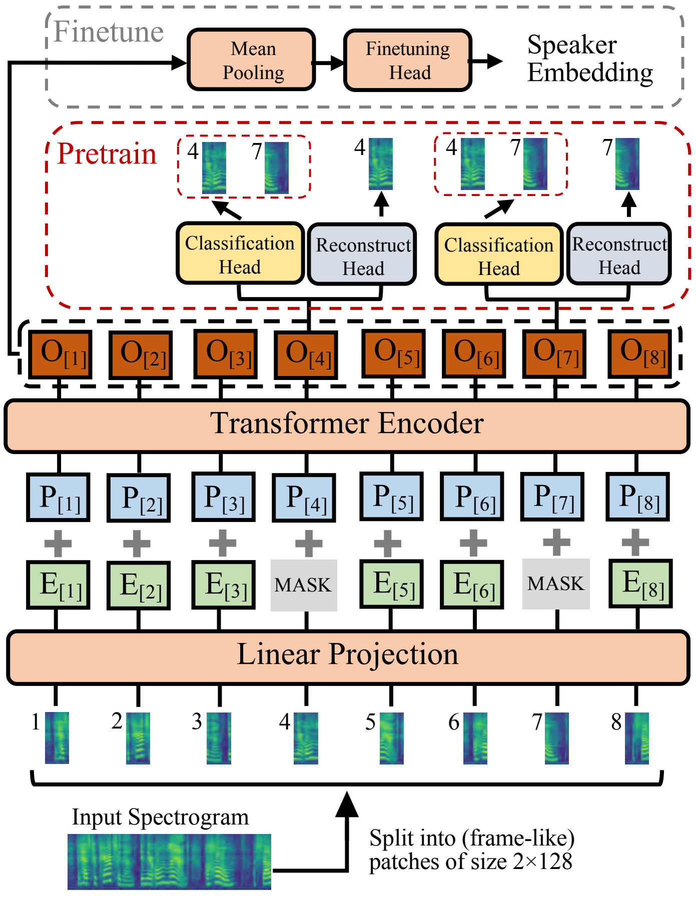

Figure 5: Architecture of the (frame-based) SSAST (adapted from [24]).

To assess the impact of temporal structure in audio spectrograms for SV tasks, we compare the performance of models trained on unaltered spectrograms with those trained on temporally shuffled spectrograms.

3.2. Data

3.2.1. Conversion to Mel Spectrogram

Given a discrete audio waveform $s \in \mathbb{R}^l$ sampled at a sampling frequency f_s, where $s[n]$ denotes the n-th sample in the audio signal and l is the number of samples in the audio signal, the Mel spectrogram can be computed[1] as described in the following paragraphs [25, 26]. Figure 6 illustrates the main steps of the conversion on a high level.

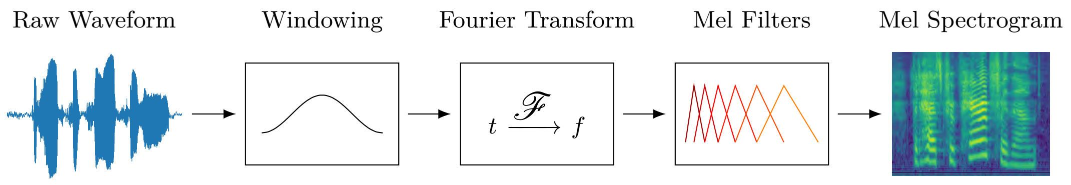

Figure 6: Main steps for converting an acoustic signal into a Mel spectrogram.

Windowing and Frame Striding

The signal s is divided into overlapping frames using a stride of S and a window length of W. This results in L frames where

$$L = \left\lfloor \frac{l - W}{S} + 1 \right\rfloor. \tag{2}$$

The floor function, denoted by $\lfloor \cdot \rfloor$, ensures that only complete frames are used. Each frame s_i, where $i \in \{0, 1, \dots, L-1\}$, is centered as follows:

$$s_i = (s[h : h + W - 1] - \mu_{s_i}), \quad \text{for } h = i \cdot S, \tag{3}$$

where μ_{s_i} is the mean of the frame.

Pre-emphasis Filtering

After centering, a pre-emphasis filter is applied to each frame to accentuate higher frequencies:

$$s_i'[n] = s_i[n] - \alpha \cdot s_i[n-1] \tag{4}$$

where $\alpha = 0.97$ is the pre-emphasis coefficient.

Window Function

To minimize spectral leakage, each frame s_i' is then multiplied element-wise with a window:

$$s_i'' = s_i' \odot w \tag{5}$$

where w is obtained using a windowing function (e.g. Hanning).

[1]See https://pytorch.org/audio/main/_modules/torchaudio/compliance/kaldi.html#fbank for more details.

Fourier Transform and Power Spectrum

Each frame is zero-padded to the nearest power of two, W_{pad}, and transformed into the frequency domain using the fast Fourier transform (FFT) [27]. The power spectrum P_i of each frame is computed as:

$$P_i = \left|\mathscr{F}(s_i'')\right|^2 \in \mathbb{R}^{\frac{W_{\text{pad}}}{2}+1} \tag{6}$$

where $\mathscr{F}$ denotes the FFT for a real-valued input signal. These power spectra are then concatenated as columns to create P, a matrix of size $\left(\frac{W_{\text{pad}}}{2}+1\right) \times L$.

Mel Filter Banks

The Mel scale filter banks are applied to the power spectrum to extract frequency bands that correspond to human auditory perception. This is achieved by mapping frequencies onto the Mel scale, which is calculated using the following conversions:

$$m(f) = 1127 \log_e \left(1 + \frac{f}{700}\right) \tag{7}$$

$$f(m) = 700 \left(e^{m/1127} - 1\right) \tag{8}$$

The filter banks H, a matrix of size $M \times \left(\frac{W_{\text{pad}}}{2}+1\right)$, transform the power spectra to the Mel scale. The calculation of H_{ij} in the Mel filter banks involves constructing triangular filters between specific points on the frequency scale. For each filter i the coefficients are computed as follows:

$$H_{ij} = \begin{cases} \frac{m(f_j)-m_{\text{left}}}{m_{\text{center}}-m_{\text{left}}}, & \text{if } m_{\text{left}} \leq m(f_j) < m_{\text{center}}, \\ \frac{m_{\text{right}}-m(f_j)}{m_{\text{right}}-m_{\text{center}}}, & \text{if } m_{\text{center}} \leq m(f_j) < m_{\text{right}}, \\ 0, & \text{otherwise.} \end{cases} \tag{9}$$

where f_j is the center frequency of the j-th bin in the Fourier transformed frame, m_{left}, m_{center}, and m_{right} are the mel frequencies corresponding to the left, center, and right edges of the i-th triangular filter, respectively.

Figure 7 shows the structure of the matrix H in the case of $f_s = 16\text{kHz}$, $M = 128$ and $W_{\text{pad}} = 512$.

Application of the Filter Banks

Finally, the obtained Filterbanks H are applied to P and the logarithm is computed:

$$X = \log_e(HP + \epsilon) \in \mathbb{R}^{M \times L} \tag{10}$$

where ϵ is a small constant to avoid logarithm of zero.

This results in X, the Mel spectrogram, where M is the number of Mel bins and L is the number of frames. Each frame of the Mel spectrogram is denoted as $X[i]$, for $i = 0, 1, \ldots, L-1$.

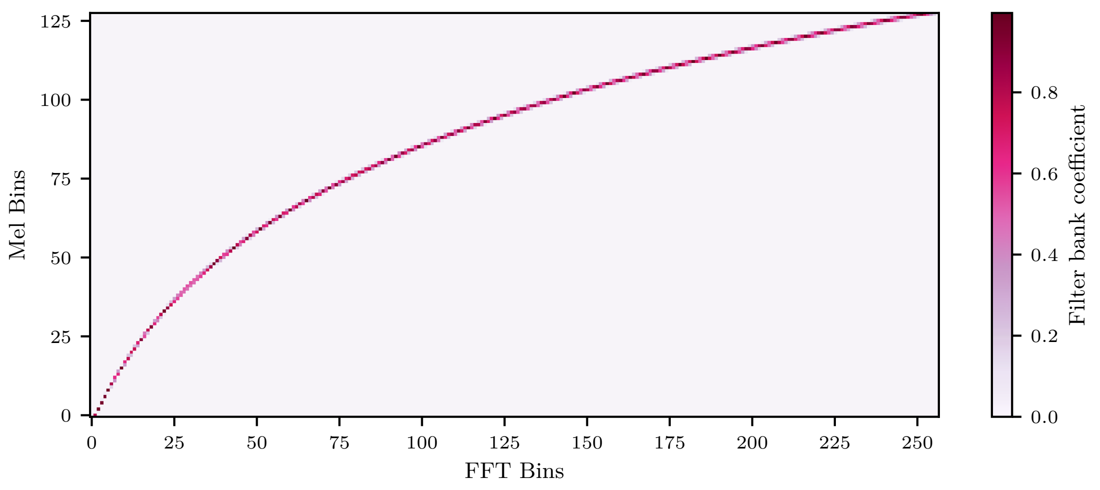

Figure 7: Mel filter bank matrix H in the case of $f_s = 16\text{kHz}$, $M = 128$ and $W_{\text{pad}} = 512$.

3.2.2. Datasets

For the pretraining task, we use AudioSet-2M [28] and Librispeech [29], enabling comparison with the results reported in [24] and thus facilitating validation of our experimental setup and codebase. Combined, these two datasets consist of 5,734 hours of audio.

For fine-tuning on SV, in addition to the Librispeech dataset the VoxCeleb dataset [30] is used. It comprises a diverse collection of speakers recorded in various settings, providing a rich landscape for assessing SV efficacy. Combined, these two datasets consist of 3,368 hours of audio from 9,593 distinct speakers.

All audio files are converted to $f_s = 16\text{kHz}$ mono channel audio signals of 10 seconds, i.e. one-dimensional arrays of $l = 160{,}000$ values. If the original audio file is longer than that, it is split into pieces. For audio files shorter than 10s, they are combined with other files.

As a preprocessing step, audio files are converted into mel spectrograms, using the procedure outlined in 3.2.1. Each spectrogram X consists of L frames with M mel bins. A single frame covers a duration of 25 milliseconds ($\Rightarrow W = 16\text{kHz} \cdot 25\text{ms} = 400$), and consecutive frames are shifted by 10 milliseconds ($\Rightarrow S = 16\text{kHz} \cdot 10\text{ms} = 160$) relative to each other. The number of frames in a 10-second audio clip can be calculated using Equation 2:

$$L = \left\lfloor \frac{l - W}{S} + 1 \right\rfloor = \left\lfloor \frac{160000 - 400}{160} + 1 \right\rfloor = 998$$

This calculates the total number of 25ms frames that can fit into 10 seconds, taking into account that each subsequent frame starts 10ms after the previous one. The number of Mel bins M is set to 128 and a Hanning window is used. Consequently, the preprocessed data samples are all of size 998×128:

$$X = \begin{bmatrix} | & | & | & | & | & & | \\ X[0] & X[1] & X[2] & X[3] & X[4] & \cdots & X[997] \\ | & | & | & | & | & & | \end{bmatrix} \in \mathbb{R}^{128 \times 998}$$

3.3. Pretraining

In this section, we describe the self-supervised pretraining algorithm developed by [24]. As opposed to frameworks that either used a discriminative (e.g., wav2vec [31]) or a generative (e.g., APC [32]) loss function, the SSAST architecure employs a joint discriminative and generative objective for pretraining, which is summarized in Algorithm 1.

Algorithm 1 Joint Discriminative and Generative Masked Spectrogram Patch Modeling

```
Require: Unlabeled Audio Dataset $\mathcal{D}$, SSAST Model $\mathcal{M}$, Number of Masked Patches $N$
 1: for every epoch do
 2:     for $X \in \mathcal{D}$ do
 3:         split $X$ into $T$ patches $x = \{x_0, x_1, ..., x_{T-1}\}$
 4:         $E = \mathcal{M}_{\texttt{patch\_embedding}}(x)$
 5:         $I = \text{RandomSample}(\{0, 1, \dots, T-1\}, N)$          ▷ Select $N$ unique indices
 6:         $E_I = E_{\texttt{mask}}$                                    ▷ Mask the Patch Embeddings
 7:         $O = \mathcal{M}_{\texttt{Transformer}}(E + P)$
 8:         $\mathcal{L}_d = 0, \mathcal{L}_g = 0$
 9:         for $i \in I$ do
10:             $r_i = \mathcal{M}_{\texttt{reconstruction\_head}}(O_i)$
11:             $c_i = \mathcal{M}_{\texttt{classification\_head}}(O_i)$
12:             $\mathcal{L} \mathrel{+}= \mathcal{L}_d(x_i, c_i, x_I) + \lambda \mathcal{L}_g(x_i, r_i)$
13:         $\mathcal{L} = \mathcal{L} \ / \ N$
14:         update $\mathcal{M}$ to minimize $\mathcal{L}$
    return $\mathcal{M}$
```

As mentioned above, the spectrograms X are of size 998×128. They are then split into $T = L/2 = 499$ patches $x = \{x_0, x_1, ..., x_{498}\}$ of size 2×128 (2 frames in the time dimension with 128 mel frequency bins). These *frame-like* patches x henceforth serve as the basic processing unit of the model, similar to the subword tokens in an LLM:

$$\underbrace{\begin{matrix} | & | \\ X[0] & X[1] \\ | & | \end{matrix}}_{x_0} \ \underbrace{\begin{matrix} | & | \\ X[2] & X[3] \\ | & | \end{matrix}}_{x_1} \cdots \ \underbrace{\begin{matrix} | & | \\ X[996] & X[997] \\ | & | \end{matrix}}_{x_{498}}$$

Each patch x_i is converted to a corresponding 768-dimensional patch embedding E_i using a linear projection layer, which is called `patch_embedding`-layer. Then a set of indices I is generated by randomly sampling N unique numbers from $\{0, 1, ..., 498\}$. For each patch x_i with an index $i \in I$, its patch embedding is replaced with a learnable mask embedding $E_{\texttt{mask}}$. Then, positional embeddings P (also learnable) are added to the patch embeddings E and the result is input to the (encoder-only) Transformer. Thus, the Transformer output O (499×768) is obtained. For each of the masked patches x_i, its corresponding O_i is input to a classification head and a reconstruction head to get an output c_i and r_i, respectively.

The Transformer has an embedding dimension of 768, 12 layers, and 12 heads. Both the classification and reconstruction heads are two-layer multilayer perceptrons (MLPs) that map O_i (768) to the same dimension as the (flattened) input patch x_i (256). r_i is expected to be close to x_i, and the model should learn to match correct (x_i, c_i) pairs. Therefore, the mean squared error (MSE) loss $\mathcal{L}_g$ is used for the generative objective and the information noise-contrastive estimation (InfoNCE) loss [33] $\mathcal{L}_d$ for the discriminative objective:

$$\mathcal{L}_g = \frac{1}{N} \sum_{i \in I} \text{mean} \left((r_i - x_i)^2 \right) \tag{11}$$

$$\mathcal{L}_d = \frac{-1}{N} \sum_{i \in I} \log_e \left(\frac{\exp(c_i * x_i)}{\sum_{j \in I} \exp(c_i * x_j)} \right) \tag{12}$$

where N is the number of masked patches and $*$ denotes the dot product. $\mathcal{L}_d$ and $\mathcal{L}_g$ are then added with a weight λ:

$$\mathcal{L} = \mathcal{L}_d + \lambda \mathcal{L}_g \tag{13}$$

Akin to [24], we set $\lambda = 10$. Finally, the parameters are updated to minimize $\mathcal{L}$ using the adaptive moment estimation (Adam) optimizer [34].

The authors of [24] found out that frame-based SSAST results in superior performance compared to patch-based SSAST for speech-related downstream tasks. Furthermore, they also found that masking $N = 400$ of the $T = 512$ frame-like patches surpasses masking $N = 250$ patches. Consequently, we adopt this masking approach during pretraining, masking $N = 390$ of the $T = 499$ frame-like 2×128 patches for the pretraining, to obtain a similar ratio of number of masked patches N / total number of patches T. During evaluation, we accidently used $N = 400$, making the task a bit harder during evaluation[2].

Two distinct pretraining instances are initiated using the SSAST architecture:

- Pretraining on original spectrograms for 10 epochs ($\mathcal{M}_\text{o}$).
- Pretraining on spectrograms with their frames shuffled randomly in time, disrupting the temporal structure ($\mathcal{M}_\text{s}$). Since the losses quickly stabilize at the expected values (Algorithm 2 and Equation 18), 5 epochs are enough for this configuration.

This dual approach is designed to yield insights into the influence of the temporal spectrogram arrangement during pretraining on the model's learning process. While shuffling the frames eliminates the model's ability to leverage temporal features, the model can still make some predictions, such as assigning an equal probability to all masked patches in the classification task and predicting the average of the unmasked patches in the reconstruct head.

We set the batch size to 48, which is twice as much as the original SSAST. We use an initial learning rate of 10^{-4}, and cut it into half if the patch prediction accuracy on the validation set stops improving for 8k iterations. We pretrained the SSAST on 3 NVIDIA A100 GPUs, which took about 4.4 days in the case of the model trained on original spectrograms ($\mathcal{M}_\text{o}$) and 2.2 days in the case of the model trained on shuffled spectrograms ($\mathcal{M}_\text{s}$).

3.4. Finetuning

Both models, $\mathcal{M}_\text{o}$ and $\mathcal{M}_\text{s}$, having undergone pretraining as described in 3.3, are then finetuned for SV on the VoxCeleb 1/2 and Librispeech dataset. The dataset for finetuning consists of 3368

[2]Our code built on [24], and they always used $N = 400$ for evaluation, independent of the number of masked patches used for training.

hours of speaker utterances from 9593 different speakers. The fine-tuning protocol, including mean-pooling on the outputs of the transformer encoder followed by a linear layer to derive speaker embeddings, is executed as delineated in [24]. Corresponding to the pretraining phase, fine-tuning is done on both the original ($\Rightarrow \mathcal{M}_{\cdot/\mathrm{o}}$) and the shuffled ($\Rightarrow \mathcal{M}_{\cdot/\mathrm{s}}$) spectrograms for each of the two pretraining conditions, which yields four distinct models, referred to as $\mathcal{M}_{\mathrm{o/o}}$, $\mathcal{M}_{\mathrm{o/s}}$, $\mathcal{M}_{\mathrm{s/o}}$ and $\mathcal{M}_{\mathrm{s/s}}$.

To finetune the two pretrained models on the downstream task of SV the main idea is to project the transformer embeddings into a new vector space where it is possible to differentiate between two speakers using a vector similarity metric [35, 36]. The paper [36] showed a proof of concept for this strategy. It used the same dataset (VoxCeleb) for training and as the speaker embedding dimension 512 was chosen.

This thesis investigates three different approaches to finetune the SSAST model; experiment 1 (3.4.1), experiment 2 (3.4.2) and experiment 3 (3.4.3).

3.4.1. Experiment 1

For the first experiment the same MLP structure is used as suggested in the finetuning part in [24]. It consists of a LayerNormalization and a fully connected layer with an input dimension of 768 and an output dimension of 527, which is very close to the approach described in [36]. As described above, both pretrained models are finetuned once on original spectrograms and once on shuffled spectograms. The performance of these four different models, $\mathcal{M}_{\mathrm{o/o}}$, $\mathcal{M}_{\mathrm{o/s}}$, $\mathcal{M}_{\mathrm{s/o}}$ and $\mathcal{M}_{\mathrm{s/s}}$, is evaluated on original spectrograms and shuffled spectrograms using 100 different speakers from the dataset used for finetuning.

3.4.2. Experiment 2

In the second experiment, an MLP is used with an output dimension of 256 and the same architecture as deployed in the classification head during the pretraining stage. The idea behind it is that the pretraining loss forces the classification head to create correct matches of masked patches and the corresponding embeddings. Therefore, this architecture should also have the potential to accurately generate inter- and intra-speaker embeddings. For performance evaluation 100 different speakers from the finetuning dataset have been used.

3.4.3. Experiment 3

During evaluation and after consultations with our supervisor, Prof. Dr. Thilo Stadelmann, we have reasons to believe that the training time in Experiment 1 (Section 3.4.1) was not sufficient. Therefore a third experiment has been set up. It uses the same head as described in Experiment 1 (Section 3.4.1) because, as discussed in 4.2, this head yields the most promising performance. This experiment has been run for 14 Epochs on the same dataset as described in 3.4. Unfortunatly, it was not possible for us to let the experiment run for more epochs due to time limitations. For performance evaluation 100 different speakers from the finetuning dataset have been used.

3.4.4. Loss

For contrastive learning there are different loss functions that have been widely used in state-of-the-art SR [35, 37]. The loss functions relevant for this thesis are discussed in the following subsections.

The first approach was to use the triplet loss. This decision has been made because of the wide use of the triplet loss function in this domain. The downside of the triplet loss was that the implementation could not be made very efficient as we wanted it to be, because our code used a for-loop to iterate over the different output embeddings to calculate the loss. When using the cosine embedding loss, we were able to vectorize the loss calculation, which lead to faster training. The required time per sample for the triplet loss was 0.017 seconds on average. With the cosine embedding loss the time per sample decreased by almost 20% to 0.014 seconds.

To finalize our decision, we compared the performance of the two approaches by finetuning with both setup for one day. We have achieved similar results in terms of performance and therefore the more efficient implementation has been used.

Triplet Loss

The triplet loss [38] was our first approach. It is computed using the following equation:

$$\mathcal{L} = \max\left(\|\mathcal{M}(X_A) - \mathcal{M}(X_P)\|_p - \|\mathcal{M}(X_A) - \mathcal{M}(X_N)\|_p + \alpha, 0\right) \tag{14}$$

We set $p = 2$ and $\alpha = 1$, which is also the default for PyTorch [39, 40]. In Equation 14, $\mathcal{M}(X_A)$ denotes the output of the model (i.e. the speaker embedding) for the audio sample of an *anchor* speaker, $\mathcal{M}(X_P)$ represents the embedding of an audio sample of the same speaker (a *positive* sample) and $\mathcal{M}(X_N)$ is the embedding of a different speaker (a *negative* sample). During the training process $\mathcal{M}(X_A)$ and $\mathcal{M}(X_P)$ are pulled closer to each other, while $\mathcal{M}(X_A)$ and $\mathcal{M}(X_N)$ are pulled apart, as illustrated in Figure 8.

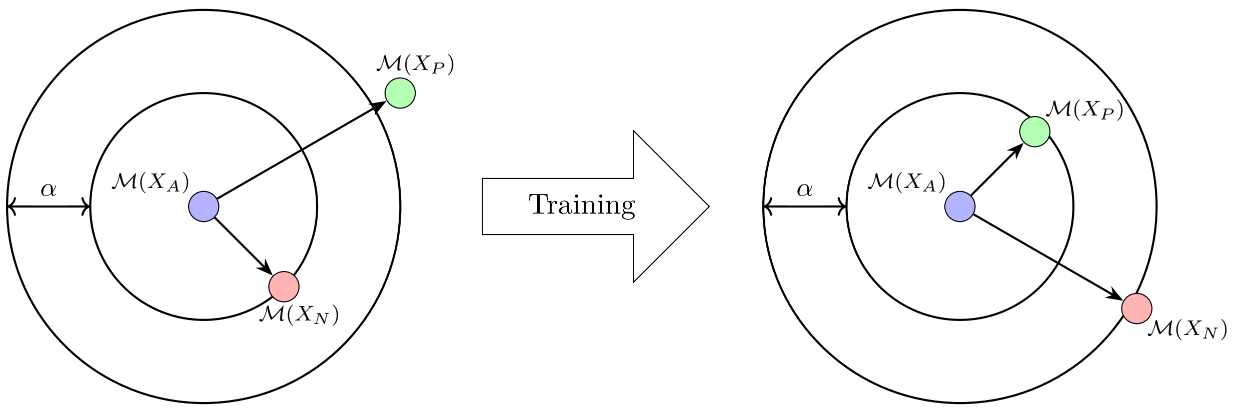

Figure 8: Visualization of the Triplet loss (adapted from [41]).

Cosine embedding loss

The cosine embedding loss was the final loss function used for further evaluation. For two speakers with identity labels a and b, it is computed using the following equation:

$$\mathcal{L} = \begin{cases} 1 - \text{sim}\left(\mathcal{M}(X_a), \mathcal{M}(X_b)\right), & \text{if } a = b \\ \max\left(0, \text{sim}\left(\mathcal{M}(X_a), \mathcal{M}(X_b)\right) - \alpha\right), & \text{if } a \neq b \end{cases}, \tag{15}$$

where $\mathcal{M}(X)$ is the output of the model for a Mel spectrogram X and $\text{sim}(\cdot,\cdot)$ denotes the cosine similarity of two vectors:

$$\text{sim}(u, v) := \cos(\theta) = \frac{u * v}{\|u\|\|v\|} = \frac{\sum_{i=1}^{n} u_i v_i}{\sqrt{\sum_{i=1}^{n} u_i^2}\sqrt{\sum_{i=1}^{n} v_i^2}}. \tag{16}$$

We set $\alpha = 0$, which is also the default in Pytorch.

3.4.5. Testing

Each finetuned model is scrutinized under the same two testing conditions; on shuffled as well as on orinial spectrograms. This evaluation strategy is pivotal to discern whether models leverage temporal features. For performance evaluation, EER and cosine similarity are calculated.

EER - Equal Error Rate

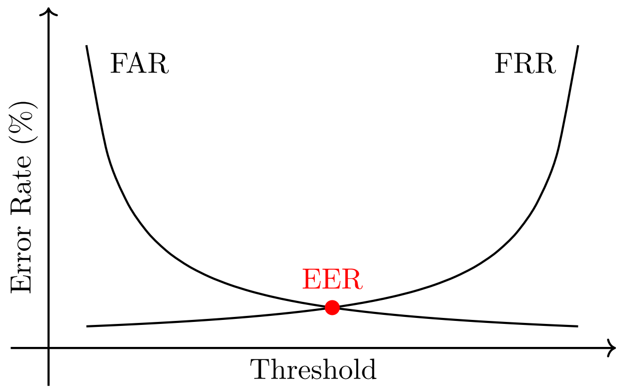

Figure 9: Illustration of FAR and FRR curves with EER point.

The EER is a popular benchmark in SV for comparing model performance. It is well suited for the performance evaluation of binary classification models. It provides a single value that summarizes the system's performance by balancing two types of errors: false acceptances and false rejections. The EER is defined as the point at which the false acceptance rate (FAR) and false rejection rate (FRR) are equal, as illustrated in Figure 9.

Using the EER as a performance metric makes it possible to conduct the NEURURER examination [12] as described in 2.4 and answer the main research question of this thesis.

Cosine similarity

In data analysis, cosine similarity is a measure of similarity between two non-zero vectors defined in an inner product space. Cosine similarity is the cosine of the angle between the vectors; that is, it is the dot product of the vectors divided by the product of their lengths. It follows that the cosine similarity does not depend on the magnitude of the vectors, but only on their angle [42, 43].

Evaluating this metric can underline the statement beeing made by the EER and directly verifies if the finetuned model has learned to differentiate between intra- and inter-speaker embeddings.

4. Results

4.1. Pretraining

4.1.1. Original Spectrograms

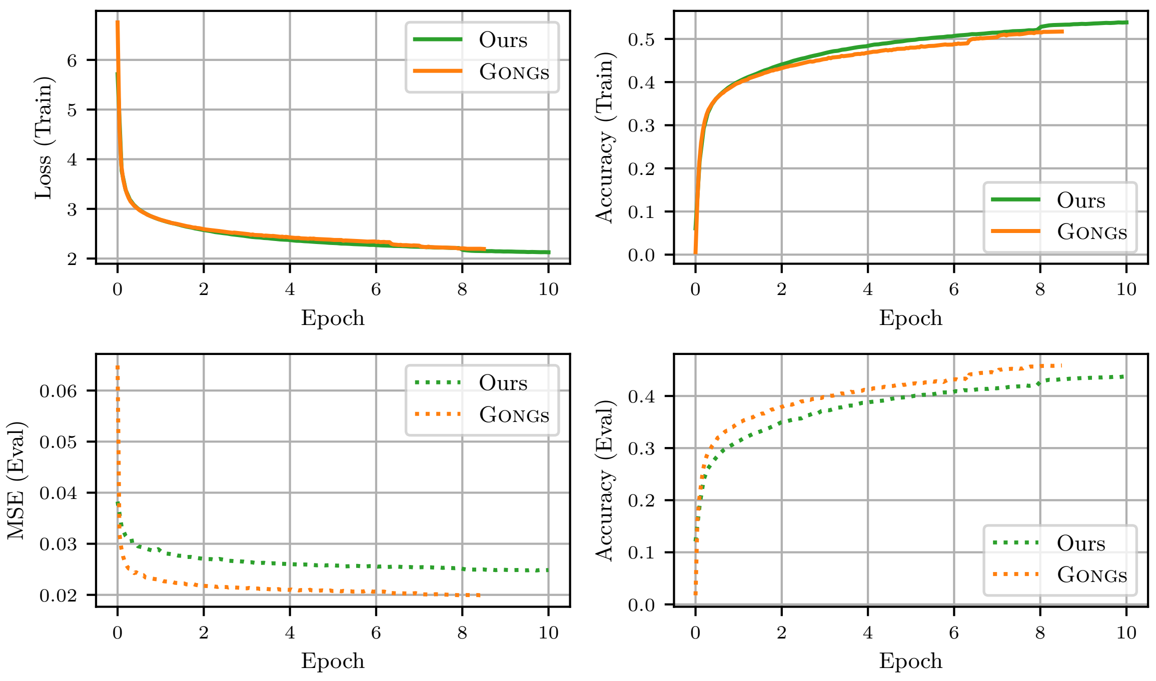

Figure 10: Pretraining result of the model trained on original spectrograms ($\mathcal{M}_o$).

Figure 10 shows the pretraining results of our model trained on original spectrograms (green) compared to the results reported in [24] (orange). The two plots on the top show the performance on the train split, the bottom two plots contain the performance on the test split. The top left plot shows the train loss, calculated using Equation 13. The bottom left plot shows the MSE on the test split[1]. On the right side, we report the accuracy on the train split (top) and the test split (bottom), where accuracy refers to the number of correctly classified patches by the classification head, divided by the total number of masked patches.

As mentioned in 3.3, we accidently masked $N = 400$ out of $T = 499$ patches during evaluation rather than $N = 390$, which results in a more challenging task (compared to predicting 400 of 512 patches) and explains the slightly worse performance on the evaluation dataset compared to [24].

[1]In [24], they did not keep track of the InfoNCE on the evaluation dataset.

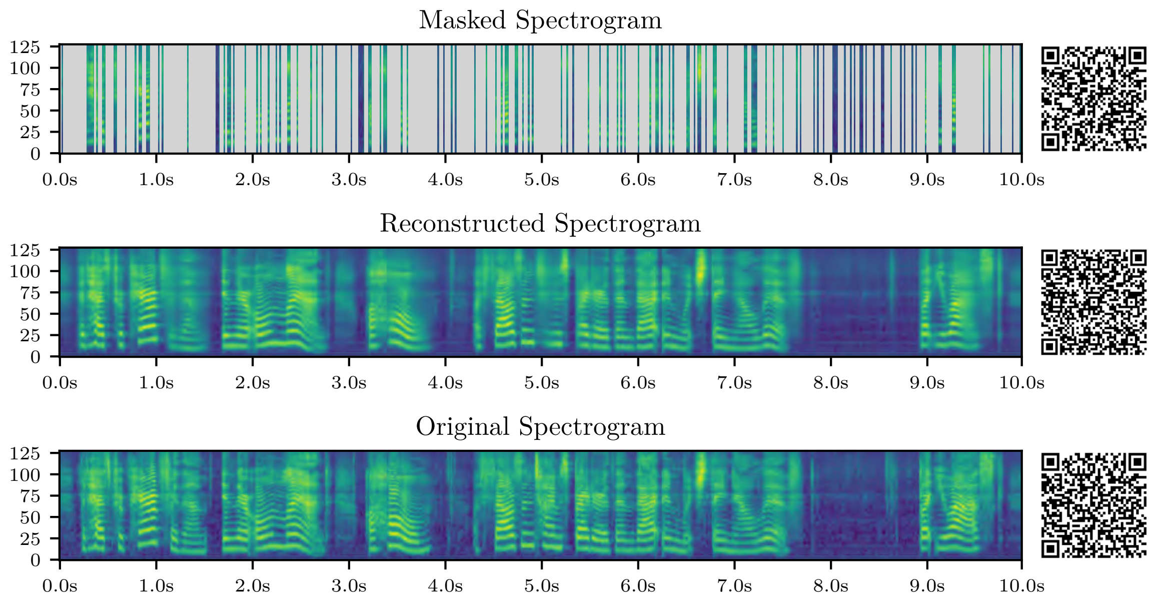

Figure 11: Spectrogram with randomly sampled mask indices (top), reconstructed spectrogram from the model trained on original spectrograms (middle), original spectrogram (bottom). Resynthesized audio files from each spectrogram can be downloaded by clicking or scanning the corresponding QR-Code.

Figure 11 shows the capabilities of the model pretrained on original spectrograms with an example. In the top part of the image, the masked spectrogram is shown. It is masked at 370 randomly[2] sampled indices I in the range $\{0, 1, \ldots, 498\}$, i.e., $I = \text{RandomSample}(\{0, 1, \ldots, 498\}, 370)$. This corresponds to 7.4 seconds of masked audio. As seen in the middle spectrogram, the reconstruction is very accurate.

4.1.2. Shuffled Spectrograms

As seen in Figure 12, the loss decreases a bit after the first epoch and then remains more or less at a constant value. The constant value can be calculated for both the InfoNCE as well as the MSE loss. For the train split, this can be done as follows. The procedure for the eval split is analogous.

For the MSE, the expected loss $\mathcal{L}_{g,\text{expected}}$ can be estimated with Algorithm 2. The best prediction of a spectrogram with randomly shuffled frames should be to take the average of the unmasked frames as a prediction for the masked ones. To calculate $\mathcal{L}_{g,\text{expected}}$, we can therefore iterate over the dataset, randomly mask $\tilde{N}_m = 2N = 2 \cdot 390 = 780$ frames, calculate the mean in each frequency bin of the remaining $\tilde{N}_u = L - \tilde{N}_m = 998 - 780 = 218$ frames[3], and use this as the prediction for the masked frames. Then we calculate the MSE using Equation

[2]A seed of 15 is used for reproducibility.

[3]The notation $\tilde{N}_m$ and $\tilde{N}_u$ are used to denote the number of masked and unmasked *frames* respectively, to avoid confusion with N, which represents the number of masked *patches*.

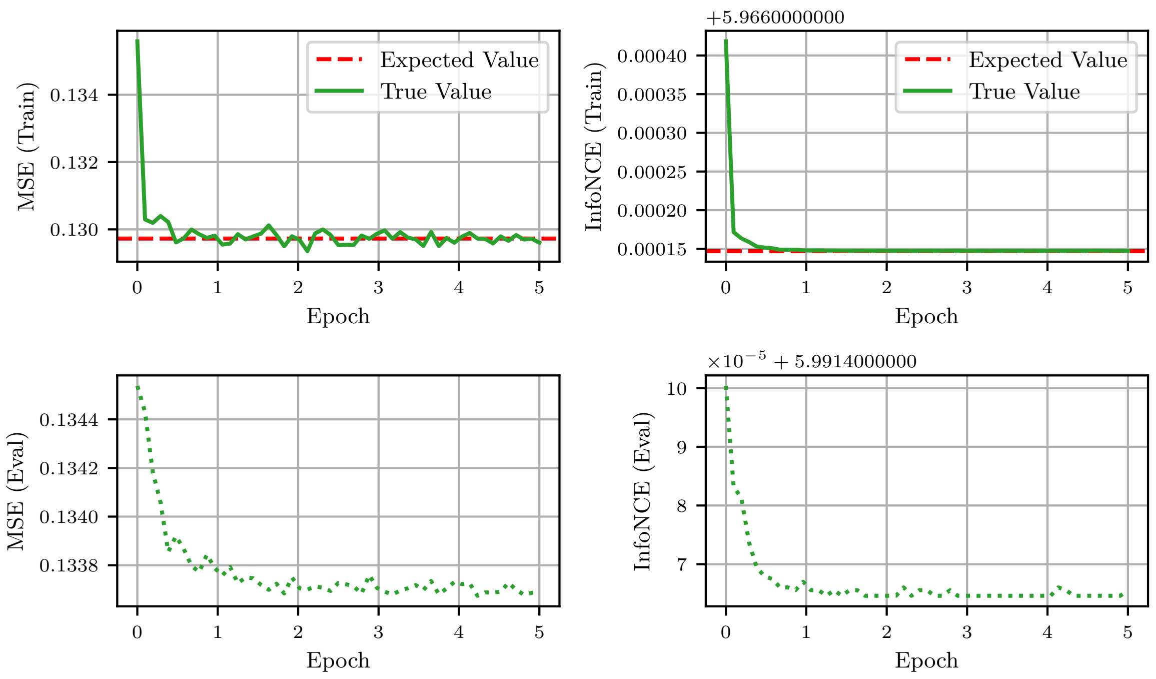

Figure 12: Pretraining result of the model trained on shuffled spectrograms ($\mathcal{M}_\text{s}$).

Algorithm 2 Calculation of Expected MSE Loss $\mathcal{L}_{g,\text{expected}}$ for the $\mathcal{M}_\text{s}$

```
Require: Unlabeled Audio Dataset $\mathcal{D}$, Number of Masked Patches $N$
 1: $\tilde{N}_m = 2N$                                         ▷ Number of masked frames
 2: $\tilde{N}_u = L - \tilde{N}_m$                            ▷ Number of unmasked frames
 3: mse_list = []
 4: for $X \in \mathcal{D}$ do
 5:     $I_u = \text{RandomSample}(\{0, 1, \dots, L-1\}, \tilde{N}_u)$      ▷ Indices of unmasked frames
 6:     $\hat{X} = \frac{1}{\tilde{N}_u} \sum_{i \in I_u} X[i]$            ▷ Mean vector of unmasked frames
 7:     $I_m = \{0, 1, \dots, L-1\} \setminus I_u$                       ▷ Indices of masked frames
 8:     $\text{MSE} = \frac{1}{\tilde{N}_m} \sum_{i \in I_m} \text{mean}\left((X[i] - \hat{X})^2\right)$   ▷ MSE for remaining frames
 9:     Append MSE to mse_list
10: $\mathcal{L}_{g,\text{expected}} = \text{mean}(\texttt{mse\_list})$
11: return $\mathcal{L}_{g,\text{expected}}$
```

11 and take the average over all samples in the dataset, yielding

$$\mathcal{L}_{g,\text{expected}} \approx 0.129726,$$

which seems to be the asymptotic constant[4] the MSE is approaching.We can also visualize this property by taking a sample from the dataset, calculating the average of the unmasked frames and plotting a certain number of predicted frames (Figure 13). The predictions are very close to the average of the unmasked frames. This confirms that the model trained on

[4]To be precise, this value is not exactly the same for each run of Algorithm 2, since it slightly depends on the (random) selection of the frames used to calculate the average. For a sufficiently large and coherent dataset, this is not a concern.

shuffled spectrograms merely learns to calculate the average of the unmasked frames in the spectrogram. The complete masked spectrogram, along with the reconstructed as well as the original spectrogram are shown in Figure 14. As can be seen from the middle spectrogram, the reconstruction is very poor, because the model only learned to calculate the average of the unmasked frames in the spectrogram.

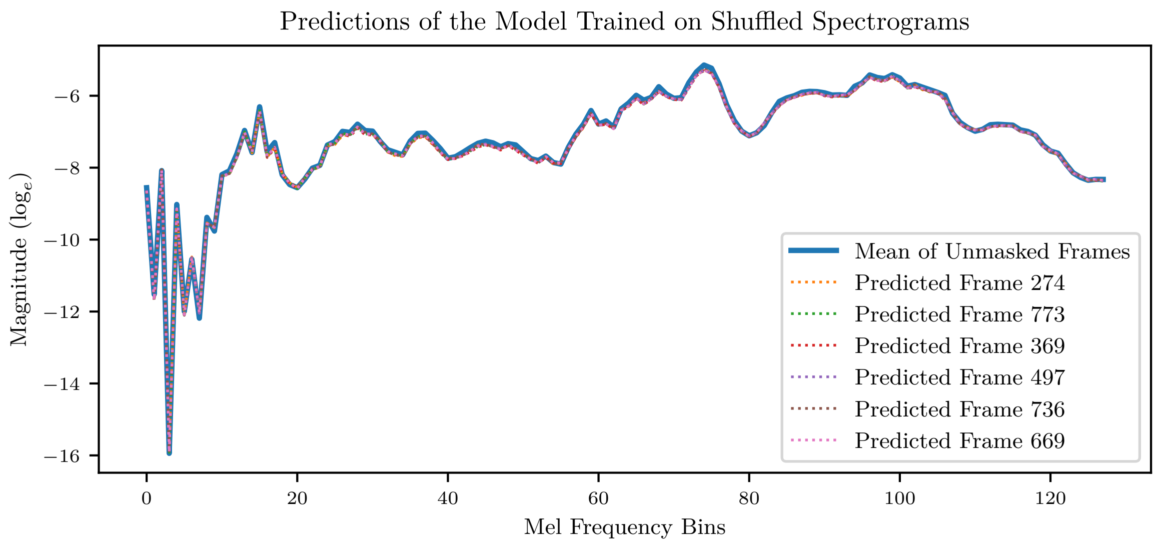

Figure 13: Mean of the unmasked frames of the same spectrogram as in Figure 14 as well the prediction of the model at six randomly chosen frames from the masked areas.

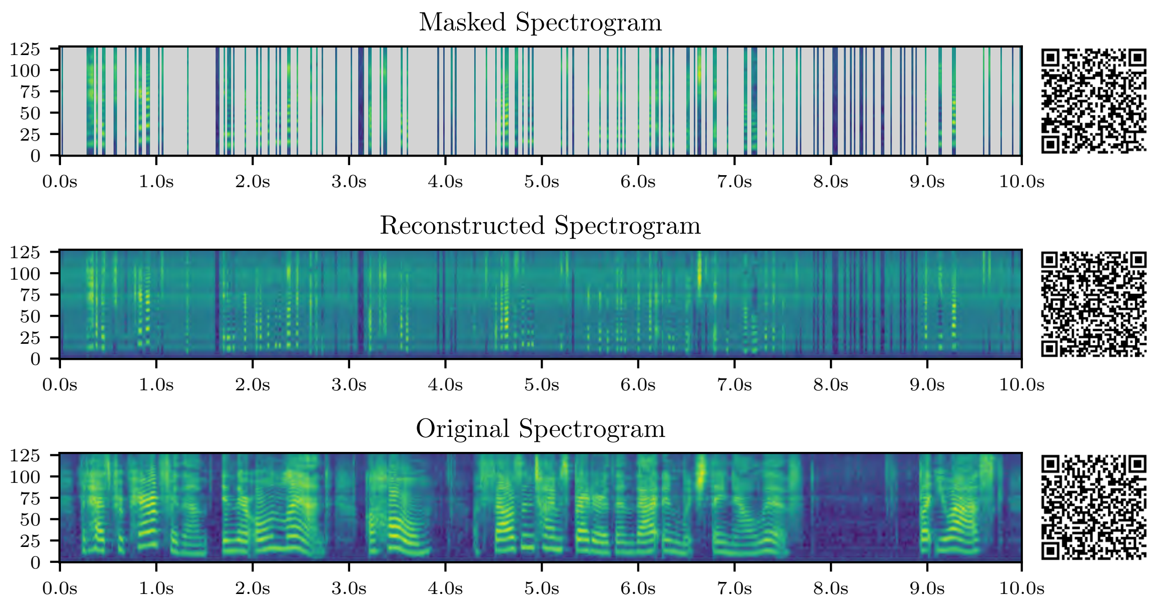

Figure 14: Spectrogram with randomly sampled mask indices (top), reconstructed spectrogram from the model trained on shuffled spectrograms (middle), original spectrogram (bottom). Resynthesized audio files from each spectrogram can be downloaded by clicking or scanning the corresponding QR-Code.

For the InfoNCE, each masked frame has an equal probability of being the correct one at any given location. This scenario occurs if

$$\frac{\exp(c_i * x_i)}{\sum_{j\in I}\exp(c_i * x_j)} = \frac{1}{N}, \forall i \in I. \tag{17}$$

Equation 12 can then be simplified as follows:

$$\mathcal{L}_{d,\text{expected}} = \frac{-1}{N}\sum_{i\in I}\log_e\left(\frac{1}{N}\right) = \frac{-1}{N}\cdot N\cdot(-\log_e(N)) = \log_e(N). \tag{18}$$

For $N = 390$, we obtain

$$\mathcal{L}_{d,\text{expected}} \approx 5.966147,$$

which is exactly the asymptotic value the InfoNCE loss approaches, as can be seen in Figure 12.

4.2. Finetuning

In this section, the results of the finetuning as described in 3.4 are shown and interpreted. This section is structured as follows. First the results of the second finetuning approach (Experiment 2, described in section 3.4.2) using a MLP with an output dimension of 256 are discussed. The following sections have been achieved using the first attempt. Exceptions are clearly mentioned. Then the results of the EER are shown and interpreted. Afterwards the results of the cosine similarity are shown and interpreted. The general performance is then evaluated and further interpreted. At the end the results are compared with the Neururer test as described in 2.4.

During the last few days before the delivery date of this thesis a 3rd experiment as described in 3.4.3 has been run. The reason for that was because experiment 2 is considered to be a failure. Experiment 3 shows very interesting results leading to another conclusion for this thesis. Due to time concerns the conclusion made from experiment 1 and 2 remains in this thesis. The results of this experiment can be found at the end of this section 4.2.5.

4.2.1. Experiment 2

As described in 3.4.2, the second attempt uses a different head to project the (mean pooled) transformer output to speaker embeddings. Due to time limitations only the SSAST pretrained on the original data, $\mathcal{M}_\text{o}$, has been finetuned, and only on the original data. No further training runs have been done using this head. The model has been trained for 30 epochs. Training it 6x longer than the models trained using approach one (Experiment 1, described in section 3.4.1) should give more insights about the model performance when applying a longer training time.

The EER is shown in table 2. The performance has decreased. The reasons for that cannot be fully explained. One supposition is that the embedding dimension is too small. To underline this presumption, further research would need to be done. It is suggested to apply different optimizer or hyperparameter to minimize the risk of converting to a local minimum.

Table 2: Test results of the finetuning using original spectrograms from the Librispeech and Voxceleb 1/2 dataset. SR performance is measured using the EER. Bold font indicates best results per model, cell coloring scales with quality per model.

model	pretraining ↓ / finetuning ↓ / test →		original	shuffled
$\mathcal{M}_{o/o}$	original	original	13.75	11.26

Table 3: Test results of the finetuning using original spectrograms from the Librispeech and Voxceleb 1/2 dataset. SR performance is measured using the cosine similarity for positive pairs and for negative pairs.

model	pretraining ↓ / finetuning ↓ / test →		original	shuffled
$\mathcal{M}_{o/o}$	original	original	Pos.: 0.979 Neg.: 0.723	Pos.: 0.927 Neg.: 0.717

When evaluating the cosine similarity in Table 3, the cosine similarity for positive pairs approaches the optimal value of 1. However, the cosine similarity for negative pairs, which ideally should be near 0, is relatively high. This discrepancy indicates suboptimal performance.

4.2.2. Experiment 1

The results of experiment 1 are shown in table 4. While not being state-of-the-art, they are within the same order of magnitude as the models tested in [12] (Table 1). Interestingly, $\mathcal{M}_{s/s}$ achieves the best performance. $\mathcal{M}_{o/s}$ also achieves better performance than $\mathcal{M}_{o/o}$ when tested against shuffled spectrograms. This evidence suggests that model performance for SR benefits from shuffled spectrograms / achieves better performance when forced to rely solely on FBA. This hypothesis has been discussed and affirmed in recent research. Shuffling the frames within each block introduces variability in the feature extraction process. This helps the model to generalize better, as it learns to identify speakers based on a more varied set of inputs, rather than being dependent on a specific sequence of frames [44].

When evaluating the cosine similarity in Table 5, it does not necessarily mean that a high discrepancy between the positive and negative similarities reflect a better EER.

Table 4: Test results of the finetuning using original and shuffled spectrograms from the Librispeech and Voxceleb 1/2 dataset. SR performance is measured using the EER. Bold font indicates best results per model, cell coloring scales with quality per model.

model	pretraining ↓ / finetuning ↓ / test →		original	shuffled
$\mathcal{M}_{o/o}$	original	original	10.63	12.13
$\mathcal{M}_{o/s}$		shuffled	15.61	**10.47**
$\mathcal{M}_{s/o}$	shuffled	original	11.87	11.69
$\mathcal{M}_{s/s}$		shuffled	8.41	**7.84**

4.2.3. Neururer Test

Comparing the obtained EERs with the results reported in [12] reveals a similar pattern. The performance of $\mathcal{M}_{o/s}$ and $\mathcal{M}_{s/s}$ relative to $\mathcal{M}_{o/o}$ and $\mathcal{M}_{s/o}$ does not result in a deterioration

Table 5: Test results of the finetuning using original and shuffled spectrograms from the Librispeech and Voxceleb 1/2 dataset. SR performance is measured using the cosine similarity for positive pairs and for negative pairs.

model	pretraining ↓ / finetuning ↓ / test →		original	shuffled
$\mathcal{M}_{o/o}$	original	original	Pos.: 0.913 Neg.: 0.163	Pos.: 0.798 Neg.: 0.167
$\mathcal{M}_{o/s}$		shuffled	Pos.: 0.918 Neg.: 0.526	Pos.: 0.864 Neg.: 0.105
$\mathcal{M}_{s/o}$	shuffled	original	Pos.: 0.786 Neg.: 0.238	Pos.: 0.81 Neg.: 0.148
$\mathcal{M}_{s/s}$		shuffled	Pos.: 0.809 Neg.: 0.418	Pos.: 0.767 Neg.: 0.326

of SR performance; rather, an improvement in SR performance is observed. Consequently, this thesis provides evidence that transformers do not possess a significant inductive bias that makes them more reliant on SST compared to other architectures.

4.2.4. General Evaluation

The findings of this thesis provide evidence that transformer networks do not inherently possess an inductive bias to learn SST. However, this does not refute the hypothesis that transformer networks can be directed to learn SST, thereby achieving more robust performance in SR.

Model Improvements

We postulate that the performance of the SSAST for SR can be enhanced, potentially yielding results that support the thesis of an inductive bias to learn SST. To improve performance, we recommend increasing the batch size and extending the training duration. Additionally, exploring various hyperparameters should be considered.

Research indicates that contrastive learning benefits from larger batch sizes [45]. Increased batch sizes in contrastive learning facilitate better exploration of the data space, resulting in more diverse negative samples and more accurate gradient estimates.

Figure 15 illustrates the train and test loss over five epochs during fine-tuning of $\mathcal{M}_o$ on original spectrograms, producing $\mathcal{M}_{o/o}$. Based on these loss functions, we assumed that a minimum in the loss landscape had been reached, rendering five epochs sufficient. However, consultations with Prof. Dr. Thilo Stadelmann, our supervisor, suggest that this assumption may be incorrect. Phenomena such as *grokking* can lead to improvements in generalization well beyond the point of overfitting [46].

4.2.5. Experiment 3

As delineated in 3.4.3, a third experimental iteration has been conducted over the past few days preceding the submission of this thesis. The results derived from this experiment may provide additional substantiation for the hypothesis articulated in 4.2.4, specifically that transformer networks indeed possess an inductive bias towards learning SST. Table 6 corroborates that extended training duration enhances performance for SR, yielding optimal outcomes.

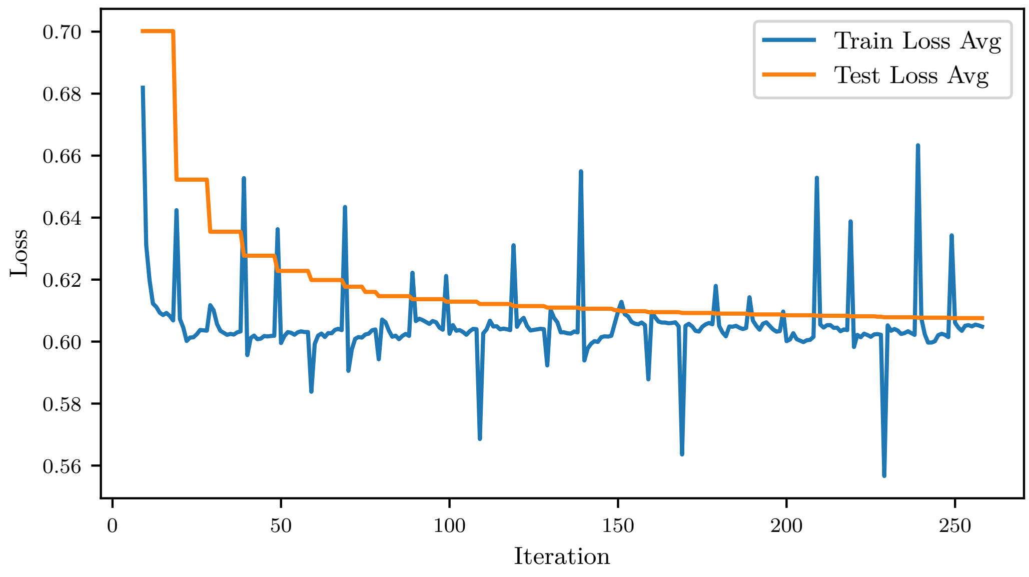

Figure 15: Training and test loss of $\mathcal{M}_{o/o}$ during 5 epochs.

Additionally, the observed performance decline between original and shuffled spectrograms is significantly more pronounced than in previous results.

When evaluating the cosine similarity in table 7, a significant discrepancy between the positive and negative similarities indicates an improved EER. However, the poor performance on the shuffled data is evidenced by cosine similarity values that are very close to each other.

Validity of Neururer's Hypothesis

To ascertain the validity of Neururer's hypothesis, it is imperative to train an additional model using shuffled spectrograms and assess its performance. Should it be demonstrated that a model trained on shuffled spectrograms fails to surpass the performance of the current model, this would constitute definitive evidence that transformer networks harbor an inductive bias for learning SST. However, within the scope of this thesis, the results presented in tables 2 and 6 are not directly comparable.

Table 6: Test results of the finetuning using original spectrograms from the Librispeech and Voxceleb 1/2 dataset. SR performance is measured using the EER. Bold font indicates best results per model, cell coloring scales with quality per model.

model	pretraining ↓ / finetuning ↓ / test →		original	shuffled
$\mathcal{M}_{o/o}$	original	original	**6.54**	19.61

Table 7: Test results of the finetuning using original spectrograms from the Librispeech and Voxceleb 1/2 dataset. SR performance is measured using the cosine similarity for positive pairs and for negative pairs.

model	pretraining ↓ / finetuning ↓ / test →		original	shuffled
$\mathcal{M}_{o/o}$	original	original	Pos.: 0.929 Neg.: 0.425	Pos.: 0.972 Neg.: 0.913

5. Conclusion

5.1. Summary

In this thesis, we successfully showed that the SSAST can be used for SR. Therefor the model described in [24] has successfully been rebuilt, achieving the same loss and performance on the pretraining task. Additionally, a model has been pretrained on shuffled spectrograms.

These models have then been finetuned for the downstream task of SR on the Librispeech + VoxCeleb 1/2 dataset containing speech utterances of 10 seconds from 9593 individual speakers. With these finetuned models, it was possible to further investigate the potential of transformers to learn SST.

The finetuned models developed for this thesis did not reach state-of-the-art performance in SR. Several factors have been identified that could improve the performance.

Nevertheless, the results of this work do show the potential of using shuffled audio data to train SR systems. In this work, such models outperformed the models trained on original data.

5.2. Future Work

Future research should focus on several factors to improve the performance of transformers for SR as well as the understanding of the degree to which they are able to model SST in SR related tasks.

5.2.1. Architecture Enhancement

Future work could benefit from experimenting with different transformer network architectures to capture temporal dependencies more effectively. Variants such as the Vision Transformer (ViT) or Swin Transformer could be explored.

5.2.2. Hyperparameter Tuning

Systematic tuning of hyperparameters, including learning rate, batch size, and network architecture, is essential. Advanced techniques such as Bayesian optimization or grid search could be employed to identify optimal configurations.

5.2.3. Extended Training Duration

Increasing the training duration may allow models to reach better performance by avoiding convergence to local minima. Long-term training with regular evaluation checkpoints can provide insights into performance trends and potential improvements.

5.2.4. Advanced Pooling Strategies

Investigating different pooling strategies and dimensions for the finetuning head might yield better speaker representation embeddings. Techniques such as attention pooling or learnable pooling parameters could be tested.

5.2.5. Enhanced Contrastive Learning

Refining the contrastive learning approach, possibly through the use of larger batch sizes or more sophisticated contrastive loss functions, might lead to improved speaker representations. Hard negative mining could further improve the learning process.

By addressing these areas, future research can build upon the findings of this thesis and might find an answer to falsifiable the potential of transformer networks to learn SST.

Bibliography

[1] Tomi Kinnunen and Evgeny Karpov. “Real-time speaker identification and verification”. In: *Audio, Speech, and Language Processing, IEEE Transactions on* 14 (Feb. 2006), pp. 277–288. DOI: 10.1109/TSA.2005.853206.

[2] Zhongxin Bai and Xiao-Lei Zhang. “Speaker recognition based on deep learning: An overview”. In: *Neural Netw.* 140 (2021), pp. 65–99. ISSN: 0893-6080. DOI: https://doi.org/10.1016/j.neunet.2021.03.004.

[3] Omar Ratib Khazaleh and Leen Ahmed Khrais. “An investigation into the reliability of speaker recognition schemes: analysing the impact of environmental factors utilising deep learning techniques”. In: *Journal of Engineering and Applied Science* 71.1 (2024), p. 13.

[4] Yao Sun, Hanyi Zhang, Longbiao Wang, Kong Aik Lee, Meng Liu, and Jianwu Dang. “Noise-Disentanglement Metric Learning for Robust Speaker Verification”. In: *ICASSP 2023 - 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).* 2023, pp. 1–5. DOI: 10.1109/ICASSP49357.2023.10096848.

[5] Philip Rose. *Forensic speaker identification.* London and New York: Taylor & Francis, 2002.

[6] John HL Hansen and Taufiq Hasan. “Speaker recognition by machines and humans: A tutorial review”. In: *IEEE Signal processing magazine* 32.6 (2015), pp. 74–99.

[7] Thilo Stadelmann and Bernd Freisleben. “Unfolding speaker clustering potential: a biomimetic approach”. In: *Proc. ACM MM.* 2009, pp. 185–194.

[8] Jaesung Huh et al. *VoxSRC 2022: The Fourth VoxCeleb Speaker Recognition Challenge.* 2023. arXiv: 2302.10248 [cs.SD].

[9] Thilo Stadelmann, Sebastian Glinski-Haefeli, Patrick Gerber, and Oliver Dürr. “Capturing suprasegmental features of a voice with rnns for improved speaker clustering”. In: *Proc. ANNPR.* 2018, pp. 333–345.

[10] Yong Zhao, Tianyan Zhou, Zhuo Chen, and Jian Wu. “Improving deep CNN networks with long temporal context for text-independent speaker verification”. In: *Proc. ICASSP.* 2020, pp. 6834–6838.

[11] Feng Ye and Jun Yang. “A Deep Neural Network Model for Speaker Identification”. In: *Applied Sciences* 11.8 (2021), p. 3603.

[12] Daniel Neururer, Volker Dellwo, and Thilo Stadelmann. “Deep neural networks for automatic speaker recognition do not learn supra-segmental temporal features”. In: *Pattern Recognition Letters* 181 (2024), pp. 64–69. ISSN: 0167-8655. DOI: https://doi.org/10.1016/j.patrec.2024.03.016.

[13] Ashish Vaswani et al. “Attention is all you need”. In: *NIPS.* 2017.

[14] Richard Lyons. *Understanding Digital Signal Processing.* 3rd ed. Pearson Education, 2010. ISBN: 9780137028528. URL: https://books.google.ch/books?id=UBU7Y2tpwWUC.

[15] Alan Oppenheim, Alan Willsky, and Hamid Nawab. *Signals & Systems.* 2nd ed. USA: Prentice Hall, 1996. ISBN: 0138147574.

[16] Harry Nyquist. “Certain Topics in Telegraph Transmission Theory”. In: *Transactions of the American Institute of Electrical Engineers* 47 (1928), pp. 617–644. URL: `https://api.semanticscholar.org/CorpusID:8632488`.

[17] Claude Elwood Shannon. “Communication in the Presence of Noise”. In: *Proceedings of the IRE* 37.1 (1949), pp. 10–21. DOI: `10.1109/jrproc.1949.232969`.

[18] Larry Small. *Fundamentals of Phonetics: A Practical Guide for Students.* USA: Pearson Academic, 1998.

[19] Shahin Tavakoli et al. *Statistics in Phonetics.* 2024. arXiv: `2404.07567 [stat.AP]`.

[20] Yanick X Lukic, Carlo Vogt, Oliver Dürr, and Thilo Stadelmann. “Learning embeddings for speaker clustering based on voice equality”. In: *Proc. MLSP.* 2017, pp. 1–6.

[21] Weidi Xie, Arsha Nagrani, Joon Son Chung, and Andrew Zisserman. “Utterance-level aggregation for speaker recognition in the wild”. In: *Proc. ICASSP.* 2019, pp. 5791–5795.

[22] Joon Son Chung et al. “In defence of metric learning for speaker recognition”. In: *Proc. Interspeech.* 2020, pp. 2977–2981.

[23] Raia Hadsell, Sumit Chopra, and Yann LeCun. “Dimensionality Reduction by Learning an Invariant Mapping”. In: *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’06)* 2 (2006), pp. 1735–1742. URL: `https://api.semanticscholar.org/CorpusID:8281592`.

[24] Yuan Gong, Cheng-I Lai, Yu-An Chung, and James Glass. “SSAST: Self-Supervised Audio Spectrogram Transformer”. In: *Proceedings of the AAAI Conference on Artificial Intelligence.* Vol. 36. 10. 2022, pp. 10699–10709.

[25] Yao-Yuan Yang et al. “TorchAudio: Building Blocks for Audio and Speech Processing”. In: *arXiv preprint arXiv:2110.15018* (2021).

[26] Jeff Hwang et al. *TorchAudio 2.1: Advancing speech recognition, self-supervised learning, and audio processing components for PyTorch.* 2023. arXiv: `2310.17864 [eess.AS]`.

[27] James William Cooley and John Wilder Tukey. “An algorithm for the machine calculation of complex Fourier series”. In: *Mathematics of Computation* 19 (1965), pp. 297–301. URL: `https://api.semanticscholar.org/CorpusID:121744946`.

[28] Jort Gemmeke et al. “Audio Set: An ontology and human-labeled dataset for audio events”. In: *ICASSP.* 2017.

[29] Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur. “Librispeech: an asr corpus based on public domain audio books”. In: *ICASSP.* 2015.

[30] Arsha Nagrani, Joon Son Chung, and Andrew Zisserman. “VoxCeleb: a large-scale speaker identification dataset”. In: *Proc. Interspeech.* 2017, pp. 2616–2620.

[31] Steffen Schneider, Alexei Baevski, Ronan Collobert, and Michael Auli. “wav2vec: Unsupervised pre-training for speech recognition”. In: *arXiv preprint arXiv:1904.05862* (2019).

[32] Yu-An Chung and James Glass. *Generative Pre-Training for Speech with Autoregressive Predictive Coding.* 2020. arXiv: `1910.12607 [eess.AS]`.

[33] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. “Representation learning with contrastive predictive coding”. In: *arXiv preprint arXiv:1807.03748* (2018).

[34] Diederik Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *ICLR*. 2015.

[35] Joon Son Chung et al. *In Defence of Metric Learning for Speaker Recognition*. 2020. arXiv: 2003.11982 [cs.LG].

[36] Joon Son Chung, Arsha Nagrani, and Andrew Zisserman. “VoxCeleb2: Deep speaker recognition”. In: *Proc. Interspeech*. 2018, pp. 1086–1090.

[37] Joachim Denzler Bjorn Barz. *Deep Learning on Small Datasets without Pre-Training using Cosine Loss*. 2019. arXiv: 1901.09054 [cs.LG].

[38] Florian Schroff, Dmitry Kalenichenko, and James Philbin. “FaceNet: A unified embedding for face recognition and clustering”. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2015), pp. 815–823. URL: https://api.semanticscholar.org/CorpusID:206592766.

[39] Adam Paszke et al. *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. 2019. arXiv: 1912.01703 [cs.LG].

[40] PyTorch. *TripletMarginLoss*. https://pytorch.org/docs/stable/generated/torch.nn.TripletMarginLoss.html#torch.nn.TripletMarginLoss. Accessed: [10 May 2024]. 2023.

[41] Mark Alan Matties. *Vector Embeddings with Subvector Permutation Invariance using a Triplet Enhanced Autoencoder*. 2020. arXiv: 2011.09550 [cs.LG].

[42] Jiawei Han, Micheline Kamber, and Jian Pei. “2 - Getting to Know Your Data”. In: *Data Mining (Third Edition)*. Ed. by Jiawei Han, Micheline Kamber, and Jian Pei. Third Edition. The Morgan Kaufmann Series in Data Management Systems. Boston: Morgan Kaufmann, 2012, pp. 39–82. DOI: https://doi.org/10.1016/B978-0-12-381479-1.00002-2.

[43] Amit Singhal. “Modern Information Retrieval : A Brief Overview”. In: 2001. URL: https://api.semanticscholar.org/CorpusID:260972090.

[44] Anke Schmeink Mahdi Barhoush Ahmed Hallawa. *Robust Automatic Speaker Identification System Using Shuffled MFCC Features*. 2021. ieeexplore: 978-1-6654-4950-2.

[45] Changyou Chen et al. *Why do We Need Large Batchsizes in Contrastive Learning? A Gradient-Bias Perspective*. 2022. NeurIPS: 9781713871088.

[46] Alethea Power, Yuri Burda, Harri Edwards, Igor Babuschkin, and Vedant Misra. “Grokking: Generalization beyond overfitting on small algorithmic datasets”. In: *arXiv preprint arXiv:2201.02177* (2022).

[47] Daniel Griffin and Jae Lim. “Signal estimation from modified short-time Fourier transform”. In: *IEEE International Conference on Acoustics, Speech, and Signal Processing*. 1983. URL: https://api.semanticscholar.org/CorpusID:53067.

[48] Brian McFee et al. “librosa: Audio and Music Signal Analysis in Python”. In: *SciPy*. 2015. URL: https://api.semanticscholar.org/CorpusID:33504.

A. Resynthesizing Audio from Spectrograms

The SSAST is trained on Mel spectrograms, which are 2-dimensional representations of the frequencies present in an audio sample. The columns in a spectrogram are derived from the amplitude response, i.e. the absolute value of the Fourier transform (Equation 6). This process results in the loss of phase information. To resynthesize the raw audio signal from its spectrogram, the phase must be estimated, which can be done using the Griffin-Lim algorithm [47]. Although there is no implementation for *Mel* spectrograms in torchaudio, the librosa library offers this functionality in a feature inversion function called `mel_to_audio` [48].

However, there are notable differences in the Mel spectrograms calculated by the two libraries for the same audio file. Mel spectrograms from torchaudio default to a $\log_e$ scale, which is the scale used in the original training of the SSAST. In contrast, the feature inversion function of librosa expects the spectrogram to be on a linear scale. Additionally, as described in Section 3.2.1, torchaudio applies a pre-emphasis filter (Equation 4), but librosa does not. To mitigate these differences, a custom function, described below, was implemented.

Given a Mel spectrogram to be resynthesized, $X \in \mathbb{R}^{M \times L}$, and a reference audio signal $s_{\text{reference}} \in \mathbb{R}^{l}$, the function `synthesize_audio_from_spectrogram` performs the following operations:

First, it computes the Mel spectrogram of the reference signal $s_{\text{reference}}$ using both librosa's `melspectrogram` function ($\rightarrow X_{\text{librosa}}$) and torchaudio's Kaldi-compliant `fbank` function ($\rightarrow X_{\text{torchaudio}}$). The resulting spectrograms are then transformed to the dB scale.

Next, an additive transformation aligns the spectrogram computed by torchaudio, $X_{\text{torchaudio}}$, with that computed by librosa, X_{librosa}. This is achieved by adding a polynomial function $p(f)$,

$$p(f) := \sum_{i=0}^{n} c_i f^i, \tag{19}$$

to each frame of $X_{\text{torchaudio}}$ to minimize the difference:

$$\Delta X(f,t) := X_{\text{librosa}}(f,t) - X_{\text{torchaudio}}(f,t). \tag{20}$$

The polynomial $p(f)$ is fitted by solving a least squares problem:

$$c = \text{argmin}_c ||\Delta X(f,t) - p(f)||^2, \tag{21}$$

where f is the Mel frequency bin index, t is the time frame index, n is the order of the polynomial, and c_i are the coefficients of the polynomial.

The coefficient vector $c \in \mathbb{R}^{n+1}$ can be calculated by solving the normal equation:

$$A^T A c = A^T b \Rightarrow c = (A^T A)^{-1} A^T b \tag{22}$$

where $A \in \mathbb{R}^{(M \cdot L) \times (n+1)}$ is the matrix of polynomial features and $b \in \mathbb{R}^{M \cdot L}$ is the (flattened) vector of differences between the two Mel spectrograms.

The polynomial $p(f)$ is then applied to the Mel spectrogram to be synthesized, $X(f,t)$, to account for the differences between the two libraries:

$$X_{\text{adjusted}}(f,t) = X(f,t) + p(f) \tag{23}$$

Figure 16 shows this process for a cross section (one frame/column) of a spectrogram.

Finally, the adjusted Mel spectrogram is transformed back to a linear scale and synthesized into audio using the feature inversion function `mel_to_audio` from the librosa library.

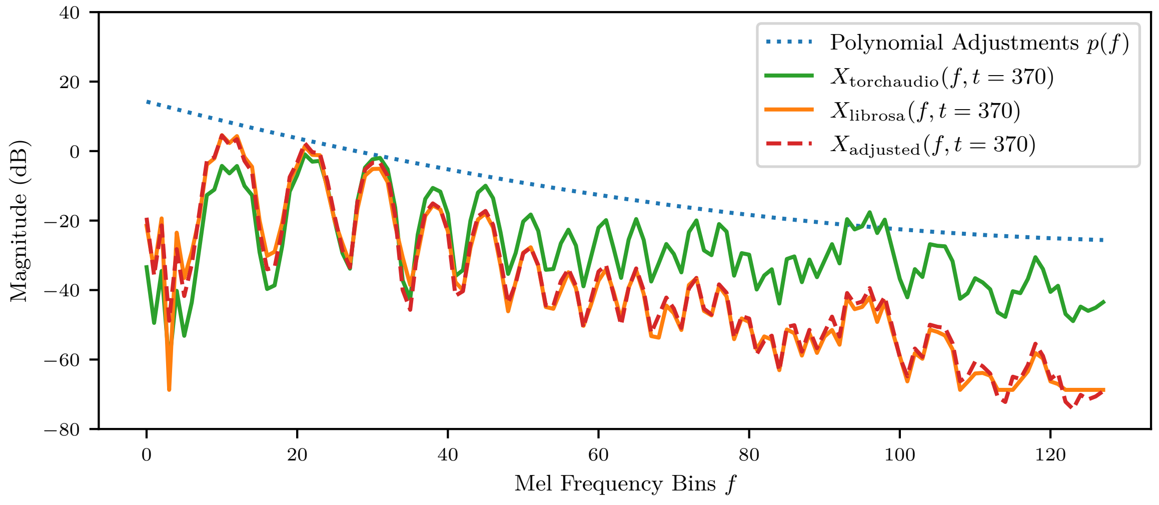

Figure 16: Illustration of the polynomial adjustments $p(f)$ calculated from the difference between the Mel spectrograms of the reference audio signal (blue, dotted). Additionally, a cross-section of the Mel spectrograms at the 370th time frame is shown.

B. More Resynthesized Spectrograms

This appendix contains further examples of the pretrained model (trained on original data) reconstructing spectrograms. Figure 17 and 18 contain $N = 180$ manually constructed mask indices which translate to a time duration of 3.6 seconds in total. Each mask has an extent/width of 20 frame-like patches, which corresponds to 0.4 seconds of audio.

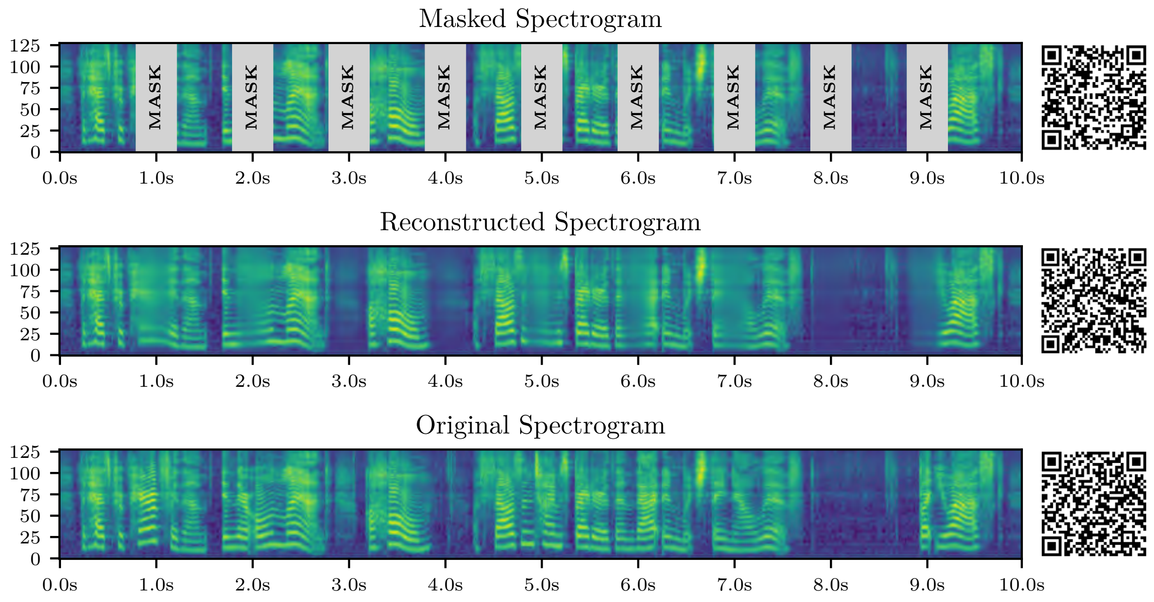

Figure 17: Reconstructing the same spectrogram as in Figure 11 and 14, but using a manually chosen set I of masked indices. Resynthesized audio files from each spectrogram can be downloaded by clicking or scanning the corresponding QR-Code.

The sample that was used for Figure 17 is the same that was already used in Figure 11 and 14. Figure 18 shows the reconstruction of a spectrogram of music. The longest trained models (trained for 10 epochs / 428k Iterations), seems to have a quite accurate sense of melody and rhythm.

Figure 19 shows a masked spectrogram with one large mask of 1.6 seconds at the center of the spectrogram. As can be seen in the *Reconstructed Spectrogram from Original Model*, even the model trained on original spectrograms tends to make something resembling the average, if a masked patch has a large width.

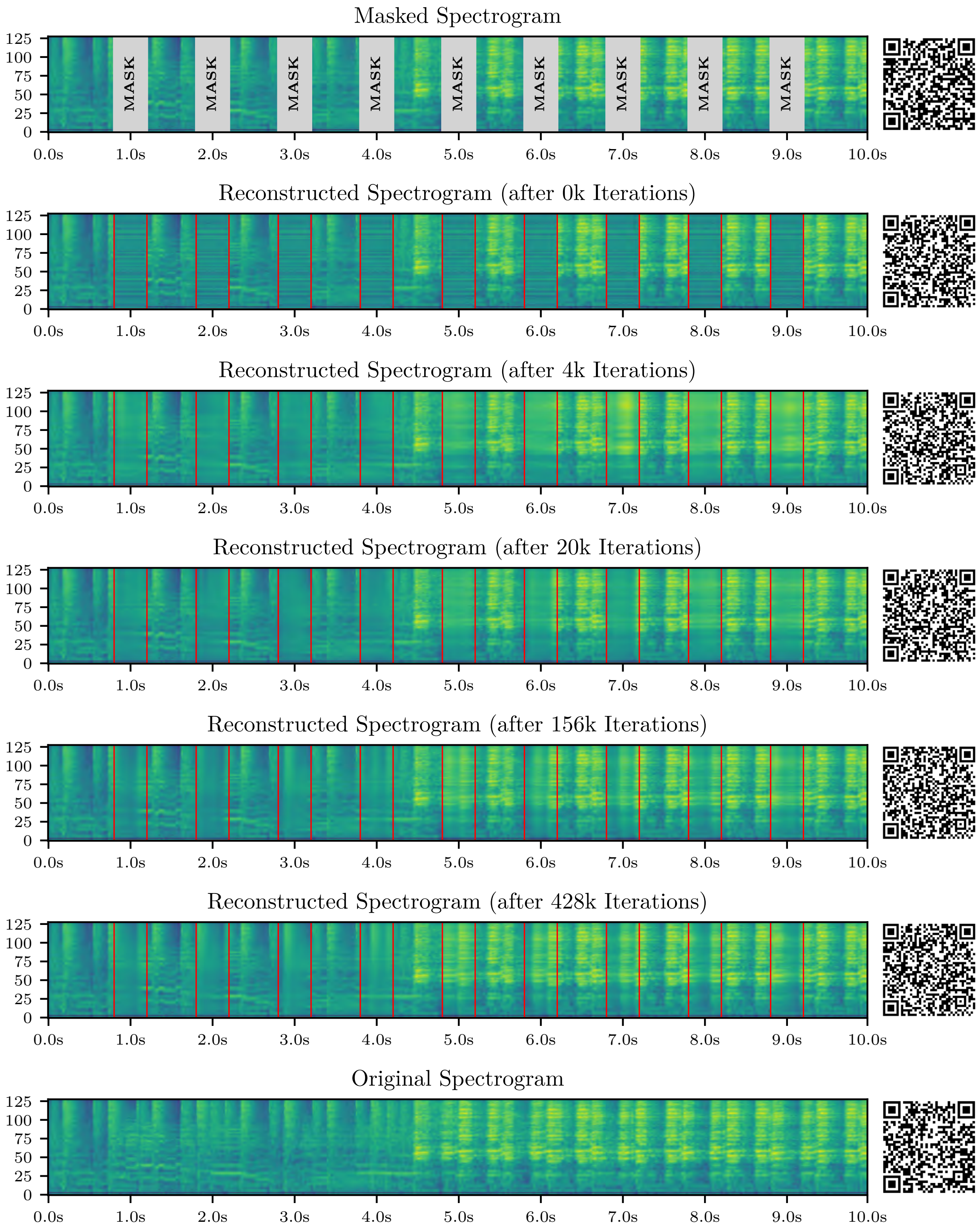

Figure 18: Reconstructing a spectrogram of music with the model at different stages of pre-training. Resynthesized audio files from each spectrogram can be downloaded by clicking or scanning the corresponding QR-Code.

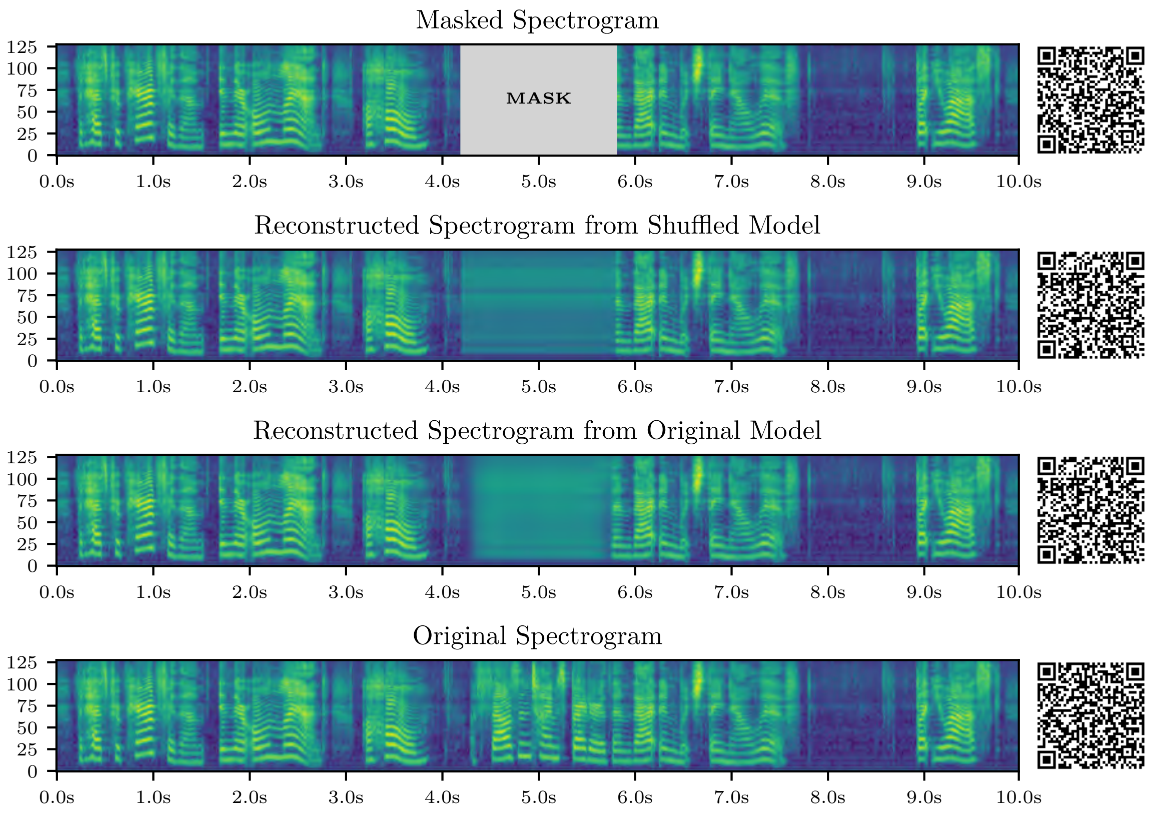

Figure 19: Manually masking $N = 80$ of the frame-like 2×128 patches, but aligning the masks all at the center and next to each other, forming one large mask of 1.6 seconds. Resynthesized audio files from each spectrogram can be downloaded by clicking or scanning the corresponding QR-Code.