A Deep LSTM-CNN-HMM Neural Network system for Speaker Identification

Andrea Terlizzi[1], Mattia Limone[2], Carmine Iannotti[3], Luca Strefezza[4]

July 21, 2022

Abstract

Deep learning approaches are progressively gaining popularity as alternative to HMM models for speaker identification. Promising results have been obtained with Convolutional Neural Networks (CNNs) fed by raw speech samples or raw spectral features, although this methodology does not fully take into account the temporal sequence in which speech is produced.

DNN-HMM (Deep Neural Network-Hidden Markov Model) is a methodology that combines the statistical modeling power of HMMs with the learning power of deep neural networks. While this technique has seen wide use in speech recognition field, few studies tried to apply it to speaker identification tasks.

This study proposes a novel approach to the DNN-HMM methodology for text-independent speaker identification, involving the use of both convolutional and Long-Short-Term-Memory (LSTM) networks, in order to extract both high-level features from the entire audio and temporal-wise features from each frame, which are then used to predict the emission probabilities of an HMM.

The experiments conducted on the TIMIT dataset showed very promising results, suggesting that the proposed non-sequential architecture may converge faster and perform better than other known methods, if properly tuned.

[1]Andrea Terlizzi, Computer Science Department, Università degli Studi di Salerno

[2]Mattia Limone, Computer Science Department, Università degli Studi di Salerno

[3]Carmine Iannotti, Computer Science Department, Università degli Studi di Salerno

[4]Luca Strefezza, Computer Science Department, Università degli Studi di Salerno

1. Introduction

Speaker identification (SI) tasks can be classified in two big groups: text-dependent and text-independent.

In the first type of task, identification is done with the help of a passphrase that the individual has to pronounce so that the system can recognize him or her, based both on the audio features extracted by the speaker audio utterances and the spoken words.

Text-independent SI, on the other hand, is entirely based on features extracted from audios, and cannot rely on either the phonetic structure of the spoken sentence or its content.

This study, inspired by recent and multiple encouraging results in the application of deep neural networks to text-independent SI [15], sometimes used in conjunction with other types of models such as Hidden Markov Models (HMMs) or Gaussian Mixture Models (GMMs) (in so-called "hybrid approaches" [21]), proposes a new approach to DNN-HMM methodology applied to this task, involving the use of both deep Long-Short-Term-Memory networks (LSTMs) and Convolutional Neural Networks (CNNs).

To our knowledge, DNN-HMM hybrid approaches are still largely untested in text-independent SI, which is one of the reasons why we decided to use LSTM Neural Networks and CNNs,

both of which fit perfectly the SI task, as widely demonstrated in the literature [23] [15].

The study, conducted on the TIMIT dataset, takes advantage of some of the most historically appreciated and used features in the audio and speech processing literature, such as MFCCs, LPCCs, and log-scaled Mel spectrum.

The work is organized as follows: section 2 describes the used dataset in detail, presenting the file structure and its composition; sections 3 on the next page, 4 on page 4, 5 on page 7 explain the preprocessing, feature extraction and HMM acoustic modeling steps made prior to the model training; sections 6 on page 11 and 7 on page 13 describe the applied DNN-HMM technique and the proposed neural network architecture; finally, sections 8 on page 18 and 9 on page 22 describe the experiment carried out in detail, starting with the pre-training phase of the model layers, going through the training of the model and finally arriving at the results obtained and possible future developments.

2. Dataset

In this research we worked on the *DARPA TIMIT - Acoustic-Phonetic Continuous Speech Corpus* [18]. This dataset has been designed to provide speech data for the acquisition of acoustic-phonetic knowledge and for the development and evaluation of automatic speech recognition (ASR) systems. TIMIT has resulted from the joint efforts of several sites under sponsorship of the Defense Advanced Research Projects Agency - Information Science and Technology Office (DARPA-ISTO). Text corpus design was a joint exertion among the Massachusetts Institute of Technology (MIT), Stanford Research Institute (SRI), and Texas Instruments (TI).

2.1. Speaker Distribution

TIMIT contains a total of 6300 sentences, 10 sentences for each of 630 speakers from 8 major dialect regions of the United States. Table 1 shows the number of speakers for the 8 dialect regions, broken down by sex. A speaker's dialect region is the geographical area of the U.S. in which they lived during their childhood years. The geographical areas correspond with recognized dialect regions in U.S., with the exception of the Western region (DR7) in which dialect boundaries are not known with any confidence, and dialect region 8 (DR8), where the speakers moved around a lot during their childhood.

Table 1: Dialect distribution of speakers.

DR	Male	Female	Total
1	31 (63%)	18 (27%)	49 (8%)
2	71 (70%)	31 (30%)	102 (16%)
3	79 (67%)	23 (23%)	102 (16%)
4	69 (69%)	31 (31%)	100 (16%)
5	62 (63%)	36 (37%)	98 (16%)
6	30 (65%)	16 (35%)	46 (7%)
7	74 (74%)	26 (26%)	100 (16%)
8	22 (67%)	11 (33%)	33 (5%)
All	438 (70%)	192 (30%)	630 (100%)

2.2. File Structure

The speech and associated data is organized in the dataset according to the following directory hierarchy:

```
/data/
<USAGE>/
<DIALECT>/
<SEX><SPEAKER_ID>/
<SENTENCE_ID>.<FILE_TYPE>
```

where:

- `usage = train | test`;
- `dialect = DRi`, $i \in \{1,...,8\}$;
- `sex = M | F`;
- `speaker_id = <initials><digit>`:
  - `initials`: speaker initials, 3 capital letters;
  - `digit`: number between 0 and 9 to differentiate speakers with identical initials;

- `sentence_id = = <text_type><sentence_number>`:
  - `text_type = SA | SI | SX`;
  - `sentence_number`: 1...2342;
- `file_type: wav | txt | wrd | phn`

The three values that `text_type` can assume stands for dialect sentences (SA), phonetically-compact sentences (SX) and phonetically-diverse sentences (SI). Even though important in speech recognition, these data were not particularly relevant during execution of the experiments conducted in this study, which is focused on text-independent SI task.

Finally, all the files of type .txt, .wrd and .phn are transcription files, not used in the context of our work.

The following is an example of the structure described above: `/data/train/DR1/FCJF0/SA1.wav`, indicating: training set, dialect region 1, female speaker with ID CJF0, sentence text "SA1", speech waveform file.

2.3. Suggested Training/Test Subdivision

TIMIT comes with texts and speakers already subdivided into suggested training and test sets. This subdivision has evolved over time to meet different criteria (as described by the authors [18, p. 20]). In the context of this research, the proposed subdivision has been ignored in order to create a new one more suitable for its intended purposes, as described in section 3.2 on the next page.

3. Preprocessing

Preprocessing is a critical step in any project related to machine learning, even more so in systems where background-noise or silence in training/testing audio signals is completely undesirable [22]. SI task requires efficient feature extraction approaches from speech signals, because most of the spoken portion includes speaker-related attributes useful to the identification.

TIMIT being a dataset recorded in a controlled environment, doesn't need overly complicated preprocessing operations before moving on to those typical of audio signals processing and that intersect the feature extraction phase, such as framing, windowing, spectrogram computation etc. For this reason, we just applied a limited silence removal, described below, while the operations that intersect the feature extraction phase are described in the appropriate section (4 on the following page).

3.1. Silence Removal

Silence removal is used to eliminate the unvoiced and silent portion of the speech signal. For this purpose, input signal is divided into small segments (frames) and root mean square (RMS) of each individual segment is calculated and compared with a specific threshold value [2].

RMS value of each individual segment can be calculated as:

$$\mathrm{RMS}_{\mathrm{Segment}} = \sqrt{\mathrm{mean}(\mathrm{Segment})^2},$$

while threshold value is computed as:

$$R_{th} = \frac{\mu + v}{2},$$

where v is the minimum RMS value of K voiced signals and μ is the mean RMS value of K unvoiced signals. The formula to compute μ is the following:

$$\mu = \frac{1}{K}\sum_{i=1}^{K}\mathrm{RMS}_{\mathrm{Unvoiced}}$$

If $\mathrm{RMS}_{\mathrm{Segment}}$ of individual segment is less than R_{th} then the segment id deleted. Similarly all the segments are compared with threshold value and system will delete all the unvoiced portion from the input speech signal. That being said, the silence removal function is expressed as:

$$f(x) = \begin{cases} \mathrm{RMS}_{\mathrm{Segment}} > R_{th}, & \text{voiced signal} \\ \mathrm{RMS}_{\mathrm{Segment}} \leq R_{th}, & \text{silent signal} \end{cases}$$

As for our study, we only removed silent frames longer than $0.25s$, since shorter silence time slices could represent discriminating characteristics useful in identifying speakers.

3.2. Dataset Training/Test Subdivision

Another fundamental aspect of the preprocessing phase is the dataset split in training and test set. As extensively described in the previous sections, the TIMIT dataset consists of 10 audios per speaker. Since the purpose of the work is speaker identification, it is necessary that audios of each speaker be present in both the training set and the test set, so we opted for the classic 80/20 split, randomly extracting 2 audios for each speaker to be included in the test set, and inserting the remaining 8 in the test set.

4. Feature Extraction

Feature extraction is one of the key steps in all supervised learning problems. The resulting feature vector (in numeric form) of the feature extraction step is fed as an input into machine learning algorithms (K-NN, SVM, NB, etc.) for the construction and validation of classification/regression model [22].

Short-term spectral features are extracted from short frames (20 - 30 ms) of speech signals, as the speech signal changes continuously due to the articulation of sounds. As a result of these short frames, the extracted features are perceived to be stationary and preserve the local information. Mel Frequency Cepstral Coefficients (MFCCs) and Linear Predictor Cepstral Coefficients (LPCCs) are the most widely employed short-term spectral features in speaker identification [22], indeed we could say that cepstral coefficients derived from either linear prediction analysis or a filter bank approach are today still considered the de-facto standard in the audio feature extraction [33], even if many deep learning-related works [34], [15], [23] have shown how less refined features (raw waveform signals, raw log-spectrograms or Mel-scaled log-spectrum features, ...) can also be just as effective if not better in some cases.

It's important to note that one of the reasons for such success is that spectral features represent phonetic information, since they are derived directly from spectra [33], and these are the ones that allow a better discerning between speakers. In the next two sections we go in more details about the two aforementioned coefficients and how we used them in our study.

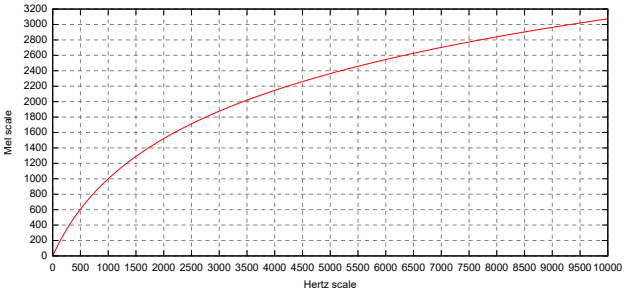

Figure 1: Plot of pitch mel scale versus Hertz scale, source [29].

4.1. MFCCs

Mel Frequency Cepstral Coefficents (MFCCs) are widely used features in ASR and SI. They were introduced by Davis and Mermelstein in the '80s, and have been state-of-the-art ever since.

As the name suggests, MFCCs are based on the Mel-scale (where *Mel* stands for *melody*), a logarithmic scale devised to map the frequency of a sound on a scale that better reflects the human perception of them. Figure 1 shows the plot of pitch Mel scale versus Hertz scale, we can note how 1000 mel equals 1000 Hz.

The MFCC feature extraction technique basically includes windowing the signal, applying the DFT, taking the log of the magnitude, and then warping the frequencies on a Mel scale, followed by applying the inverse DCT. The detailed description of the various steps involved in the MFCC feature extraction is explained below.

4.1.1. Pre-Emphasis. Pre-emphasis refers to filtering that emphasizes the higher frequencies. It is used to balance the spectrum of voiced sounds that have a steep drop in the high frequency range. Usually the glottal source has a slope of about -12 dB/octave. However, when acoustic energy is radiated from the lips, this results in a spectrum rise of about +6 dB/octave. Therefore, pre-distortion removes some of the glottal effects from the vocal tract parameters.

The most commonly used pre-emphasis filter is given by the following transfer function:

$$H(z) = 1 - bz^{-1}$$

4.1.2. Framing. The speech signal is a slowly time-varying or quasi-stationary signal. For stable acoustic characteristics, speech needs to be examined over a sufficiently short period of time. Therefore, speech analysis must always be carried out on short segments across which the speech signal is assumed to be stationary; usually, it is divided into smaller frames each lasting between 20ms and 40ms.

4.1.3. Windowing. Advancing the time window every 10 ms enables the temporal characteristics of individual speech sounds to be tracked, and the 20 ms analysis window is usually sufficient to provide good spectral resolution of these sounds, and at the same time short enough to resolve significant temporal characteristics. The purpose of the overlapping analysis is that each speech sound of the input sequence would be approximately centered at some frame. On each frame, a window is applied to taper the signal towards the frame boundaries. This is done to enhance the harmonics, smooth the edges, and to reduce the edge effect while taking the DFT on the signal. Hamming window has been used as window shape by considering the next block in feature extraction processing chain and integrates all the closest frequency lines. Let:

- X_n = input signal;
- Y_n = output signal;
- W_n = hamming window;

then the resulting windowed signal is:

$$Y_n = X_n \times W_n, \quad n = 0, 1, 2, \ldots, N$$

DFT Spectrum Each windowed frame is converted into magnitude spectrum by applying DFT.

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{\frac{-j2\pi nk}{N}}; \quad 0 \leq k \leq N-1$$

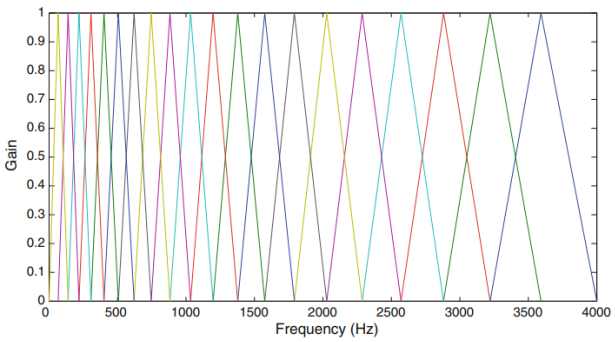

Figure 2: Mel-filter bank, source [33]

.

4.1.4. Mel Spectrum. Mel spectrum is computed by passing the Fourier-transformed signal through a set of band-pass filters known as Mel-filter bank. A Mel is a unit of measure based on the human ears perceived frequency. It does not correspond linearly to the physical frequency of the tone, as the human auditory system apparently does not perceive pitch linearly [33]. We reported the relationship between Mel-scale and Hertz-scale in figure 1 on the preceding page. The approximation of Mel from physical frequency can be expressed as:

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

where f denotes the physical frequency in Hz, and f_{Mel} denotes the perceived frequency [11].

Filter banks can be implemented in either the time domain or the frequency domain, but in MFCC computation, they are usually implemented in the latter one. The center frequencies of the filters are normally evenly spaced on the frequency axis, however, to mimic the perception of the human ear, a warped axis is implemented, according to the nonlinear function given in eq. 1. The most commonly used filter shaper is triangular, the triangular filter banks with Mel frequency warping we used is presented in figure 2.

The Mel spectrum of the magnitude spectrum $X(k)$ is computed by multiplying the magnitude spectrum by each of the triangular Mel weighting filters:

$$s(m) = \sum_{k=0}^{N-1} \left[|X(k)|^2 H_m(k) \right]; \quad 0 \leq m \leq M-1 \quad (2)$$

where M is the total number of triangular Mel weighting filters. $H_m(k)$ is the weight given to the k-th energy spectrum bin contributing to the m-th output band and is expressed as [33]:

$$H_m(k) = \begin{cases} 0, & k < f(m-1) \\ \frac{2(k-f(m-1))}{f(m)-f(m-1)}, & f(m-1) \leq k \leq f(m) \\ \frac{2(f(m+1)-k)}{f(m+1)-f(m)}, & f(m) < k \leq f(m+1) \\ 0, & k > f(m+1) \end{cases} \tag{3}$$

4.1.5. Discrete Cosine Transform. Since the vocal tract is smooth, the energy levels in adjacent bands tend to be correlated. Before computing the DCT, the Mel spectrum is usually represented on a log scale, then DCT is applied to the transformed mel frequency coefficients to produce a set of cepstral coefficients. This results in a signal in the cepstral domain with a quefrency peak corresponding to the pitch of the signal and a number of formants representing low quefrency peaks [33].

Since most of the signal information (corresponding to the vocal tract features) is represented by the first few MFCC coefficients, we only selected the first 13, as usual in most literature. As in [32], extracting only those coefficients ignoring or truncating higher order DCT components is sufficient to obtain a robust system. Finally we report, again from [32], the final formula for calculating MFCCs involving DCT:

$$c(n) = \sum_{m=0}^{M-1} \log_{10}(s(m)) \cos\left(\frac{\pi n(m-0.5)}{M}\right) \tag{4}$$

with $n = 0, 1, 2, \ldots, C-1$, where $c(n)$ are the cepstral coefficients and C is the number of MFCCs, usually between 8 and 13.

4.1.6. First and Second Order Derivatives. First and second derivative of each MFCC (also known as delta and delta-delta features) are also taken into account, since they carry additional information about the coefficient variation over time, ending up with 39 coefficients for each frame, which will become the input of our deep learning model.

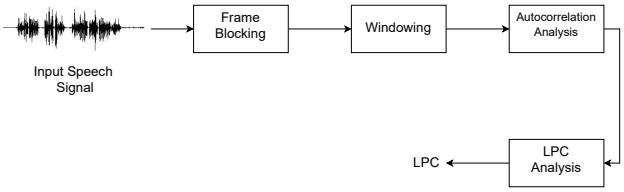

Figure 3: LPC block diagram.

Delta coefficients tell about the speech rate, while delta-delta coefficients provide information similar to acceleration of speech [33]. The commonly used definition for computing these dynamic parameters (delta features) is:

$$\Delta c_m(n) = \frac{\sum_{i=-T}^{T} k_i c_m(n+i)}{\sum_{i=-T}^{T} |i|} \tag{5}$$

where $c_m(n)$ denotes the m-th feature for the n-th time frame, k_i is the i-th weight and T is the number of successive frames used for computation. Generally T is taken as 2. The delta-delta coefficients are computed by taking the first order derivative of the delta coefficients [33].

4.2. LPCCs

LPCCs are another type of cepstral coefficients widely used in speech processing [22]. As mentioned in the section 4 on page 4, spectral features represent phonetic information, as they are derived directly from spectra. The features extracted from spectra, using the energy values of linearly arranged filter banks, equally emphasize the contribution of all frequency components of a speech signal. In this context, LPCCs are used to capture emotion-specific information manifested through vocal tract features [33].

As in the case of MFCCs, the Linear Predictive (LP) analysis requires multiple steps, exemplified in the figure 3, that represents the steps from the speech signal to the LPCs (Linear Predictive Coefficients). The next step is to actually extract the cepstral coefficients, as shown in Figure 4 on the next page.

In this work we extracted from the speech signal 13 LPCCs per speech frame of 16 ms, using an overlap of 8 ms and the *hann* windowing.

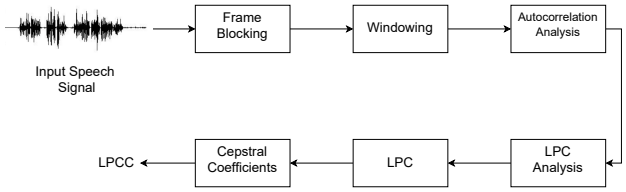

Figure 4: LPCC block diagram.

4.2.1. Cepstrum and LPCCs Calculation. Cepstrum may be obtained using linear prediction analysis of a speech signal. The basic idea behind linear predictive analysis is that the nth speech sample can be estimated by a linear combination of its previous p samples as shown in the following equation [33]:

$$s(n) \approx a_1 s(n-1) + a_2 s(n-2) + \\ + a_3 s(n-3) + \cdots + a_p s(n-p) \quad (6)$$

where $a_1, a_2, a_3, \ldots$ are assumed to be constants over a speech analysis frame. These are known as predictor coefficients or linear predictive coefficients. These coefficients are used to predict the speech samples, then we can calculate the error as the difference of actual and predicted speech samples, by the following formula:

$$e(n) = s(n) - \hat{s}(n) = s(n) - \sum_{k=1}^{p} a_k s(n-k) \quad (7)$$

where $e(n)$ is the error in prediction, $s(n)$ is the original speech signal, $\hat{s}(n)$ is a predicted speech signal and a_ks are the predictor coefficients.

To compute a unique set of predictor coefficients, the sum of squared differences between the actual and predicted speech samples has to be minimized (error minimization) as shown in the next equation [33]:

$$\min_{a_1,\ldots,a_p} E_n = \sum_{m} \left[s_n(m) - \sum_{k=1}^{p} a_k s_n(m-k) \right]^2 \quad (8)$$

where m is the number of samples in an analysis frame. To obtain the LPCs from the equation above, E_n is differentiated with respect to each a_k and the result is equated to zero:

$$\frac{\partial E_n}{\partial a_k} = 0, \quad \text{for} \quad k = 1, 2, 3, \ldots, p \quad (9)$$

After finding each a_k for $k \in \{1, 2, ..., p\}$, cepstral coefficients (LPCCs) can be computed using the following recurrence relationship [33]:

$$C_0 = \log_e p \quad (10)$$

$$C_m = a_m + \sum_{k=1}^{m-1} \frac{k}{m} C_k a_{m-k}, \quad \text{for} \quad 1 < m < p \quad \text{and} \quad (11)$$

$$C_m = \sum_{k=m-p}^{m-1} \frac{k}{m} C_k a_{m-k}, \quad \text{for} \quad m > p \quad (12)$$

5. Acoustic modeling

The first step in the DNN-HMM methodology for SI is speaker acoustic modeling, which has been approached with a historically appreciated technique: GMM-HMM (Gaussian-Mixture Model-Hidden Markov Model).

Other acoustic modeling approaches include segmental models [40], maximum entropy models [8], and (hidden) conditional random fields [17].

Before diving into acoustic modeling for DNN-HMM, let's introduce the statistical concepts GMM-HMM is based on.

5.1. GMM

A Gaussian Mixture Model (GMM) is a parametric probability density distribution represented by a weighted sum of M multivariate Gaussian densities, as originally proposed by Zolfaghari and Robinson [44].
Each n-variate Gaussian component $\mathcal{N}(x \mid \mu_k, \Sigma_k)$ resembles the distribution of a cluster in the dataset, and can be defined by two parameters:

Mean vector: $\mu_k = [\mu_{k,1}, \mu_{k,2}, ..., \mu_{k,n}]$, correspondig to the k-th cluster's centroid $\frac{1}{|G_k|} \sum_{x \in G_k} x$;

Covariance matrix: $\Sigma_k = ||\sigma_{i,j}^{(k)}||_{n\times n}$ such that $\sigma_{i,j}^{(k)} = Cov(X_{k,i}, X_{k,j})$ is the covariance between the i-th and the j-th feature of the samples belonging to the k-cluster G_k;

and thus it can be expressed as:

$$\mathcal{N}(x \mid \mu_k, \Sigma_k) = \frac{1}{(2\pi)^{\frac{D}{2}} |\Sigma_k|^{\frac{1}{2}}} e^{-\frac{1}{2}(x-\mu_k)^T \Sigma_k^{-1}(x-\mu_k)}$$

That being said, a GMM is defined as a weighted combination of M n-variate Gaussian densities, called components:

$$P(x \mid \lambda) = \sum_{k=1}^{M} w_k \mathcal{N}(x \mid \mu_k, \Sigma_k),$$

where:

$$\sum_{k=1}^{M} w_k = 1$$

In GMM training, dataset clusters (and thus parameters of Gaussian components) are initially computed using a clustering algorithm (e.g. k-means), and then adjusted using an E-M algorithm (such as MLE [10] or MAP [7]), which works even if the clusters overlap with each other, since this type of algorithm uses the probability of an instance x to belong to a cluster G_j, rather than the distance from its centroid.
Concerning the component weights $w_1, w_2, ..., w_M$, they are initialized randomly and then also tuned with the E-M algorithm alongside the mean vectors and covariance matrices.

Based on the structure and constraints of covariance matrices, GMMs can be divided in some different groups:

Full: each Gaussian component has its own general covariance matrix, thus the generated clusters can adopt any position/shape;

Tied: each Gaussian component has the same general covariance matrix, hence the generated clusters can have any position (means), but must have the same shape;

Diagonal: each Gaussian component has its own diagonal covariance matrix (i.e. the only non-zero entries are variances $\sigma_{i,i}^{(k)} = Var(X_{k,i})$), thus each cluster can have any shape, but their contour axes must be oriented along the coordinate axes;

Tied diagonal: each Gaussian component has the same diagonal covariance matrix, hence each cluster must have the same shape and their contour axes must be oriented along the coordinate axes;

Spherical: identical to the diagonal case, but the variances within the same covariance matrix are equal (i.e. $\sigma_{i,i}^{(k)} = \sigma_{j,j}^{(k)}, \forall i, j \in \{1, 2, ..., n\}$), thus each cluster must have a circular shape (although not necessarily the same circular shape), and their contour axes be oriented along the coordinate axes.

5.2. HMM

Hidden Markov Model (HMM) is a statistical modeling technique with an underlying couple of stochastic processes, an hidden (i.e. not observable) one, and an observable one.

The hidden stochastic process is represented by finite set of possible states, called **hidden state space** $S = \{s_1, s_2, ..., s_K\}$, transitioning to each other overtime with fixed probabilities, generating a sequence of states $q_1, q_2, ..., q_T$.
The main assumption on which HMM are built is that the probability of transiting into a state s_i at the time t depends solely on the previous state q_{t-1} (this is also called Markov Hypothesis [28]), hence:

$$P(q_t = s_i \,|\, q_1, q_2, ..., q_{t-1}) = P(q_t = s_i \,|\, q_{t-1})$$

this makes possible to collect these probabilities **transition matrix**:

$$A = \begin{bmatrix} t_{1,1} & t_{1,2} & \dots & t_{1,K} \\ t_{2,1} & t_{2,2} & \dots & t_{2,K} \\ \vdots & \vdots & \ddots & \vdots \\ t_{K,1} & t_{K,2} & \dots & t_{K,K} \end{bmatrix}, \text{ where:}$$

$$t_{i,j} = P(q_t = s_i \,|\, q_{t-1} = s_j);$$

this transition matrix can be viewed as one of the three fundamental parameters that define an HMM. Based on the structure of the transition matrix, several HMM *topologies* are defined (as seen in Figure 5, 6, 7, 8):

Ergodic: there is no 0 probability in the transition matrix, $t_{i,j} > 0, \forall i, j \in \{1, 2, ..., K\}$, so each state can be reached from each state with a non-zero probability;

Left-to-right: there exists a topological order between the states $q_1, q_2, ..., q_K$ such that for every non-zero transition $t_{i,j} > 0 \implies i < j$;

Left-to-right cyclic: there exists an ordering between the states $q_1, q_2, ..., q_K$ such that for every non-zero transition $t_{i,j} > 0 \implies i \leq j$ (there can also be self-loops);

Bidirectional: the HMM transitions can be split in two sets A, B such that each transition $t_{i,j} > 0 \in A \implies i \leq j$ and $t_{i,j} > 0 \in B \implies i \geq j$.

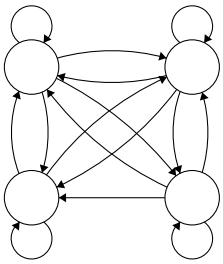

Figure 5: Ergodic

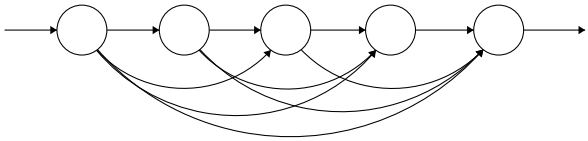

Figure 6: Left-to-right

The second fundamental parameter of an HMM is the **prior distribution** vector, which defines the probability that a sequence of states $q_1, q_2, ..., q_T$ has of starting with each state $s_1, s_2, ..., s_K$:

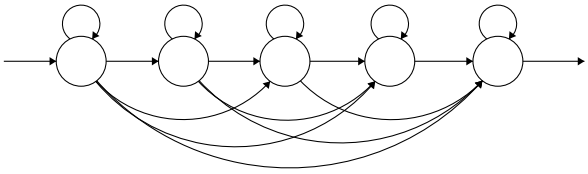

Figure 7: Left-to-right cyclic

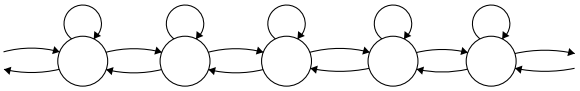

Figure 8: Bidirectional

$$\pi = [\pi_1, \pi_2, ..., \pi_K], \text{ where:}$$
$$\pi_i = P(q_1 = s_i), \forall i \in \{1, 2, ..., K\}.$$

The third and last parameter of HMM is the so-called **emission distribution**. As said before, an HMM has two underlying stochastic processes: an hidden and an observable one; similar to the former, the latter also produces a sequence over time, called *observation sequence* $o_1, o_2, ..., o_T$. At each time t, the **emission distribution** defines the probability density function (or discrete density function if the possible observation are countable) of an observation o_t, given the current state s_t:

$$e(o_t \,|\, s_t) = f_D(o_t \,|\, s_t) \text{ (continuous case)}$$

$$e(o_t \,|\, s_t) = P(o_t \,|\, s_t) \text{ (discrete case).}$$

Given a set of observations:

$$X = \{x^{(1)}, x^{(2)}, ..., x^{(m)}\},$$

an HMM is trained with an E-M algorithm to maximize the likelihood of the observations:

$$A^*, B^*, \pi^* = \arg\max_{A,B,\pi} P(X \,|\, A, B, \pi)$$

A particularly useful application of the E-M approach to HMMs is the Viterbi algorithm, a dynamic programming algorithm originally proposed by Andrew Viterbi [42], which makes possible to compute in polynomial time the most likely sequence of states, given a sequence of observations:

$$q_1^*, ..., q_T^* = \arg\max_{q_1,...,q_T} P(q_1, ..., q_T \,|\, o_1, ..., o_T).$$

The idea of the algorithm is to express this probability through a recurrence relationship, and then use classic dynamic programming with memoization to compute its value faster (and in polynomial time, which wouldn't be possible otherwise).
The recurrence relationship used to express the probability of the most likely sequence of state, given the observations $o_1, o_2, ..., o_T$ is the following:

$$V_{1,k} = P(o_1 \,|\, k)\, \pi_k, \forall k \in S$$
$$V_{t,k} = \max_{q \in S} P(o_t \,|\, q) A_{q,s_k} V_{t-1,q},$$

where A is the transition matrix and π is the prior distribution. The algorithm operates as follows:

(i) computes the base case probabilities $V_{1,k}$ for each state k, storing it into a $T \times K$ matrix V;

(ii) computes each remaining entry $V_{t,k}$ of the matrix according to the recurrent relationship, taking advantage of the already stored values in the matrix, and storing into another matrix Ptr the state used to compute $V_{k,t}$ (representing the most likely state to transition from, into state s_k at time t):

$$\text{Ptr}_{t,k} = \arg\max_q P(o_t \,|\, q) \cdot A_{q,s_k} \cdot V_{t-1,q}$$

(iii) computes the most likely sequence of states using the computed values in the V matrix, retrieving the sequence backwards:

(a) $q_T^* = \arg\max_q V_{T,q}$;

(b) $q_{t-1}^* = \text{Ptr}_{t,q_t^*}$

5.3. GMM-HMM

Given these definitions, a GMM-HMM is a HMM where emission distribution of each state is defined by a GMM with M Gaussians:

$$e(o_t \,|\, s) = P(o_t \mid \lambda) = \sum_{k=1}^{M} w_k \mathcal{N}(x \mid \mu_k, \Sigma_k),$$

Often, GMM-HMM are used to build acoustic models that represent the feature distribution of a specific class of audios, which is useful in a plenty of different tasks. In speech recognition for example, GMM-HMM acoustic models are built to represent the phoneme distribution [41], [14], and each state corresponds to a phoneme, while in scene classification each GMM-HMM acoustic model is built to represent the audios recorded in a particular scene/context [5].

Not too differently, in SI task, which is the main subject of this study, GMM-HMM acoustic models are built to represent the audios recorded by a specific speaker. Therefore, for p speakers there will be p GMM-HMM models, each trained with an E-M algorithm on utterances of the corresponding speaker:

$$\text{HMM}_1, \text{HMM}_2, ..., \text{HMM}_p$$

At identification time, an unknow audio $o = o_1, o_2, ..., o_T$ is scored against each GMM-HMM acoustic model using Viterbi algoritm [26], and the speaker is identified by the most likely path among all the speaker acoustic models:

$$s = \arg\max_i \text{Vit}_P(o, \text{HMM}_i),$$

where $\text{Vit}_P(o, \text{HMM}_i)$ is the posterior probability of the most likely HMM_i state path given o, according to the Viterbi algorithm:

$$\text{Vit}_P(o, \text{HMM}_i) = \max_{q_1,...,q_T} P_i(q_1, ..., q_T \,|\, o_1, ..., o_T)$$

This methodology has been successfully applied in multiple past studies with good results [26], alongside with the GMM-standalone acoustic modeling, which was applied to the TIMIT dataset too [37].

As for the choice of the state number K for each acoustic model, it is either chosen application domain-wise, in tasks like speech recognition where each state can represent a phoneme (e.g. triphone [9]), or can be chosen empirically through grid search methods, using instance likelihood as score.
The same applies to the mixture number M, whose optimal value can vary considerably depending on the dataset, as shown in multiple studies [13], [12], [31].

Finally, it is important to note that each HMM topology suits different types of tasks; in ASR, for example, letf-to-right topologies are often used, since it fits the phonetic structure of speech, which involves some phonemes before others overtime. In SI or scene classification, on the other hand, the most suitable topology turns out to be the ergodic [21], [5], since the structure of audio recognition patterns for these two types of task may not necessarily be ordered, as they are not related to phonemes.

GMM-HMM in DNN-HMM In DNN-HMM approach to SI, GMM-HMM acoustic models are built using feature extracted from the utterances of the training set (often MFCCs/MFCCs & deltas [21]) to represent the audios recorded by each speaker, and then used with Viterbi algorithm to generate frame-level audio labels that will be later used to train the DNN model.

More specifically, for each audio $o = o_1, o_2, ..., o_T$ from the speaker j, where each o_t is an audio frame containing some kind of feature (raw audio, power/log-scaled STFT spectrum, power/log-scaled Mel spectrum, MFCCs/ MFCCs & deltas, LPCCs, ...), Viterbi algorithm is applied to generate the most likely state sequence (this step is sometimes also called "forced alignment" [21], [5]):

$$q^* = q_1^*, q_2^*, ..., q_T^* = \text{Vit}(o, \text{HMM}_j),$$

where $\text{Vit}(o, \text{HMM}_j)$ is the most likely HMM_j state path given o, according to the Viterbi algorithm:

$$\text{Vit}(o, \text{HMM}_j) = \arg \max_{q_1,...,q_T} P_j(q_1, ..., q_T | o_1, ..., o_T)$$

Table 2: GMM-HMM grid-search Results

Feature	States	Mixtures
MFCCs & deltas	8	3
LPCCs	4	2
Log-scaled Mel Spectrum features	3	2

The generated states $q^* = q_1^*, q_2^*, ..., q_T^*$ are used as frame-level labels for the audio frames $o = o_1, o_2, ..., o_T$, so that each frame o_t corresponds to the state label q_t^*.
As will be explained shortly, these labels will later be used to train the underlying DNN model used in the DNN-HMM approach.

In this study, grid search has been performed to optimize the mixture M and state number K of our ergodic GMM-HMM acoustic models with diagonal covariance type. Results have been varying depending on the type of features used, as shown in table 5.3.

It's noteworthy that acoustic models generated using log-scaled Mel-spectrum and LPCC features were not used in later stages of the study (e.g. training of the neural network model), but were generated for comparative purposes only (although this could be subject of future studies).

6. DNN-HMM

Let's now dive into the core elements of DNN-HMM approach for SI, describing the general idea and architecture, which will be described in detail in section 7 on page 13.

The basic idea behind DNN-HMM starts from the observation that Deep Neural Networks (DNNs) are more easily capable of learning advanced, non-linear statistical relationships from input data, require a lot less preprocessing compared to GMMs, and can even work with non-independent feature frames. That being said, it seems logical and worth trying to use a DNN as an emission function of an HMM, instead of a GMM.

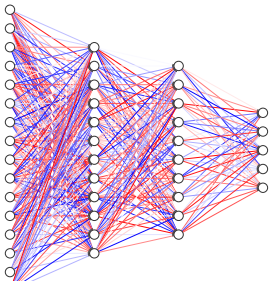

Figure 9: Deep neural network for classification with 5 output classes.

6.1. Deep Neural Network (DNN)

A Deep Neural Network (DNN) is an artificial neural network that employs more than one hidden layer between the input and output layers to classify an input pattern or, more generally, learn to approximate a target function f. In the presence of a classification problem, such as the SI problem, an attempt is made to discriminate a pattern x based on a feature y, which can take value among Q categories (also called classes), and the DNN estimates the class probabilities $p_j, j \in \{1,...,Q\}$. The input features to a DNN do not require special attention to preprocessing, and in fact in audio processing one can range from raw frames of audio, to MFCCs/MFCCs & deltas, to Mel's log-scaled spectrogram features. Figure 9 shows an example neural network for classification tasks.

More formally, a feed-forward DNN with H layers, weight matrices $W_1,...,W_H$, bias vectors $b_1,...,b_H$ and activation functions $f_1,...,f_H$ computes a non-linear function [21]:

$$g_{W,b}(x) := a_H(x) \ ,$$

where, for $h = 1,2,...,H$:

$$a_h(x) = f_h((a_{h-1}(x))^T \cdot W_h + b_h), \text{ and:}$$

$$a_0(x) = x$$

In classification tasks, the last layer activation function is often a softmax function σ, which outputs a probability $\hat{p}_j$ for each class j, and then the class with the highest probability is considered the predicted one:

$$q = \arg\max_j \hat{p}_j, \text{ where:}$$

$$\hat{p}_j = f_H(x)_j = \sigma(x)_j := \frac{e^{x_j}}{\sum_{i=1}^{Q} e^{x_i}}$$

DNN training Generally speaking, DNN are trained to minimize a loss function $L(W,b)$ (e.g. mean squared error, mean absolute error or binary cross-entropy), and at each training cycle (so-called epochs), weights W and biases b are updated according to their gradient:

$$\nabla_{W,b}(L) = \left[\frac{\partial L}{\partial b}, \frac{\partial L}{\partial W}\right]^T;$$

while gradients are computed using the backpropagation algorithm [39], the update rule for the weight based on their value can be done in many different ways, and it's in general considered a model hyperparameter to tune. Two notable and popular examples of optimization algorithms to update the network parameters are Stochastic Gradient Descent (SGD) [38] and Adadelta [43], which were also used to train the models for this work.

In multi-class classification tasks such as SI, the most used loss function is the categorical cross-entropy [21]:

$$L = \sum_{i=1}^{Q} p_j \log \hat{p}_j,$$

where p_j is the target probability for class j (usually 1 for right class, 0 otherwise) and $\hat{p}_j$ is the network-predicted probability for target class j.

6.2. How DNN-HMM works

DNN-HMM training In contrast to traditional DNN systems for IS, in those based on DNN-HMM, the network output classes do not correspond to the speakers to be identified, but to the states of each GMM-HMM acoustic model speaker generated [21]. As previously said, the state sequences computed from the speaker utterances with the Viterbi

algorithm are used as labels for audio frames, therefore each neural network output corresponds to a GMM-HMM acoustic model state. Hence, by minimizing the categorical cross-entropy loss function C on the utterances of the training set and the corresponding labels, the network is trained to predict the posterior probabilities of each HMM state k, given the input audio feature frame o_t:

$$P(s_t = k \,|\, o_t)$$

In this way, the network can learn meaningful features that allow it to assign to an audio the states computed from the acoustic model of its speaker (and consequently discriminate from one speaker to another).

DNN-HMM decoding In testing stage, a new audio $o = o_1, ..., o_T$ from the test set is given as input to the neural network, which outputs a posterior probability $P(s_t = k \,|\, o_t)$ for each frame o_t. As briefly mentioned earlier, the idea behind DNN-HMM is that the neural network defines the emission probabilities of the HMM, so each posterior is converted into an emission probability (likelihood) using Bayes' theorem [21]:

$$e(o_t \,|\, s_t = k) = P(o_t | \, s_t = k) = \frac{P(s_t = k \,|\, o_t) P(o_t)}{P(s_t = k)}$$

In the previous formula, the prior probability $P(s_t = k)$ can be approximated by occurrences of state k in the training utterances of the corresponding speaker, as relative frequency:

$$P(s_t = k) \approx \frac{\#\text{Occurences}(k)}{\#\text{STATES\_SPKR} \cdot \#\text{AUDIO\_SPKR}}$$

On the other hand, the (audio frame) prior probability $P(o_t)$ distribution is very hard to model without strong assumptions, but since all input frames are assumed to be independent each other, it can be considered as a constant scaling/normalization factor for the emission probability, and therefore ignored completely:

$$e(o_t \,|\, s_t = k) = P(o_t | \, s_t = k) = \frac{P(s_t = k \,|\, o_t)}{P(s_t = k)}$$

The last step in speaker identification task with DNN-HMM is the execution of the Viterbi decoding, where:

(i) $o = o_1, ..., o_T$ features against each speaker DNN-HMM model using the Viterbi algorithm, which uses the transition matrices T_j and prior distributions π_j of each speaker (from the GMM-HMM acoustic models), which gives in ouput a sequence of states $q_1, ..., q_T$ and its posterior probability:

$$P_{\pi_j, T_j}(q_1, ..., q_T \,|\, o_1, ..., o_T) = \text{Vit}_P^{\pi_j, T_j}(o)$$

(ii) the audio is identified as belonging to the speaker DNN-HMM model which maximizes this probability:

$$z^* = \arg\max_j P_{\pi_j, T_j}(q_1, ..., q_T \,|\, o_1, ..., o_T)$$

7. Proposed architecture

To our knowledge, the only few studies that applied DNN-HMM technique to text-independent SI tasks used deep feed-forward neural networks [21], which are hardly able to capture temporal relationships between feature frames due to their sequential structure unlike, for example, Recurrent Neural Networks (RNNs).

On the other hand, as multiple studies in the SI field suggested, the use of 1D/2D Convolutional Neural Networks (CNNs) in combination with log-scaled spectrum/Mel spectrum features can greatly improve the performance of a text-independent SI system, due to the high-level feature extraction power of this type of network [15].

This simple observation led us to believe that the combination of the features extracted by both these typologies of networks could lead to even better identification performances, especially if combined to the acoustic modeling power of HMMs.

Thus, the proposed architecture is non-sequential, and combines both convolutional and recurrent layers, trying to combine both kind of features to achieve better performance.

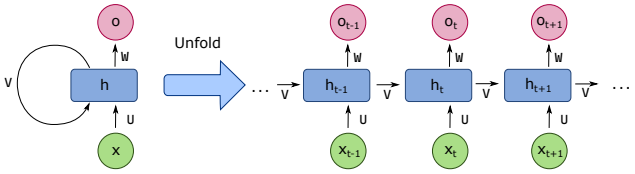

Figure 10: Unrolling of a basic recurrent neural network, compressed on the left and unfolded on the right, source [35].

Before describing the mentioned architecture, an overview of the most important network designs that make up its layers will be given.

7.1. Recurrent Neural Networks (RNN)

A recurrent neural network (RNN) is a neural network that is specialized in processing a temporal sequence of data $x_1, \cdots, x_T$.

The term "recurrent" in their name refers to the execution of the same operations on each element x_t of the sequence performed by this type of network, applying the same matrix of weights V.

This causes each output o_t to depend on the previous ones, and for the network to maintain a hidden state h_t directly dependent on the previous one h_{t-1}.

The image 10 shows an RNN unfolding over time (on the sequence) in a complete network.

We are going to analyze each element of the architecture of an RNN, explaining step by step how the information is processed within this type of network:

input : x_t represents the input of the network at instant t;

hidden state : h_t represents the hidden state at time t, which acts as the "memory" of the network, being computed based on x_t and the previous state h_{t-1} by applying weight matrices, and an activation function f (any nonlinear transformation such as tanh, ReLU, ...) and adding a bias vector c:

$$h_t = f(U \cdot x_t + V \cdot h_{t-1} + c)$$

weights : the connections between neurons of input-to-hidden type (x_t to h_t) are parameterized by a matrix of weights U, while those of hidden-to-hidden type (h_{t-1} to h_t) and hidden-to-output type (h_t to o_t) are respectively parameterized by two weight matrices V and W;

output : o_t represents the output of the network, which is also often subject to a nonlinear activation function g, especially when the network contains other layers.

$$o_t = g(b_t + W \cdot h_t)$$

where b_t is the bias vector at time t

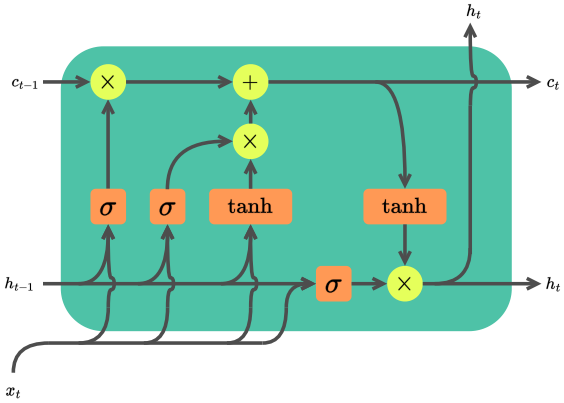

Figure 11: LSTM unit, source [27].

7.2. Long-Short-Term-Memory (LSTM)

An LSTM is a special type of RNN. RNNs use recurrent units to learn temporal features from sequence data, as LSTMs do. However, what happens inside the recurrent unit is very different between the two, as shown in image 11.

Assuming we know the basic operation of RNNs, we are going to describe the operation of an LSTM layer at time t, to which comes the time sequence $x_1, x_2, \cdots, x_T$:

Hidden state and input : the hidden state of a previous time step h_{t-1} and the input of the current time step x_t are combined before running copies through various operational gates using concatenation between vectors.

$$[h_{t-1} \mid x_t]$$

Cell state: c_{t-1} is a recurrent input representing the long-term memory of the network. In fact,

it will be modified only slightly by the current time step with a sequence of linear additions and Hadamard products (component-wise), resulting in the next cell state c_t.

Forget gate: this gate controls what saved information should be forgotten, and consists of a simple sigmoid neural network layer that multiplies the previous result with a weight matrix W_f, adds a bias vector b_f and gives the result as input to a sigmoid activation function. As the sigmoid function varies between 0 and 1, it determines which cell state values should be discarded (multiplied by 0), remembered (multiplied by 1) or partially remembered (multiplied by a value between 0 and 1).

$$f_t = \sigma\left(W_f \cdot [h_{t-1} \mid x_t] + b_f\right)$$

Input gate: consists of a sigmoid neural layer, which multiplies $[h_{t-1}|x_t]$ with a weight matrix W_i, adds a bias b_i and gives the result as input to a sigmoid function. Outputs between 0 and 1 identify the important elements of $[h_{t-1}|x_t]$ that are to be added to the cell state c_{t-1}.

Proposed update cell state : consists of a tanh neural layer, which multiplies $[h_{t-1}|x_t]$ with a weight a matrix W_c, adds a bias vector b_c and gives the result as input to the activation function tanh. The output $\tilde{c}_t$ determines the candidate values for storage within the cell state.

$$\tilde{c}_t = \tanh\left(W_c \cdot [h_{t-1} \mid x_t] + b_c\right)$$

Update cell state: the previous cell state c_{t-1} is multiplied by the result of forget gate f_t to remove irrelevant information, and the result is added component by component to the vector of candidate values $\tilde{c}_t$, weighted by the output of input gate i_t, to finally obtain the next cell state c_t.

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$

Output gate: consists of a simple sigmoid neural layer that, again, multiplies $[h_{t-1}|x_t]$ with a weight matrix W_o, adds a bias vector b_o and gives the result in ingress to a sigmoid activation function.

$$o_t = \sigma\left(W_o \cdot [h_{t-1} \mid x_t] + b_o\right)$$

Update hidden state: the last step is to update the hidden-state h_{t-1}. The current cell state c_t is passed through the activation function tanh and multiplied component by component with the output gate result.

$$o_t = \tanh\left(c_t\right) \odot o_t$$

Finally, the current cell state c_t and hidden state h_t return as input to the recurrent unit, and the process repeats until the sequence ends.

7.3. Convolutional Neural Network (CNN)

Convolutional neural network (CNN) is a class of artificial neural networks that has become dominant in various computer vision and signal processing tasks and is attracting interest across a variety of domains. CNNs are capable to automatically and adaptively learn hierarchies of high-level features through layers of small weight filters representing the features. Usually, CNN architectures are composed of multiple building blocks, such as convolution layers, batch normalization and pooling layers.

Convolutional Layer A convolutional layer l_m receives as input a signal $x^{(m-1)}$ with K_m channels (either raw input or the output of $(m-1)$-th layer) and computes as output a new signal $\mathbf{x}^{(m)}$ composed of O_m channels. The output at each channel is known as a **feature map**, and is computed as [6]:

$$x_o^{(m)} = g_m\left(\sum_k w_{o,k}^{(m)} * x_k^{(m-1)} + b_o^{(m)}\right)$$

where $*$ denotes the 1D convolution operation:

$$w_{o,k} * x_k[t] = \sum_p x_k[t-p] \cdot w_{o,k}[p],$$

where $W_{o,k}^{(m)} \in R^{P_m}$ is vector called **convolutional kernel**, $b_o^{(m)} \in \mathbb{R}$ is a bias vector and g_m is the activation function applied to the final result. The convolutional kernel $w_{o,k}^{(m)}$ act as the trainable parameters of a filter that the layer can use to detect or enhance some feature in the incoming signal frames. The weights and the consequent feature detected/enhanced by the filter are learned during the training process just like the weight matrices of fully-connected neural networks or RNN.

Batch Normalization Layer Batch normalization is a technique that aims to accelerate the training of deep neural networks by reducing the shift of internal covariates. This is done through a normalization operation that fixes the means and variances of layer inputs at the batch level. This operation also has a number of extremely beneficial "side" effects:

- strong reduction in the dependence of gradients on parameter scaling or their initial values, which allows much higher learning rates to be used without the risk of divergence or gradient exploding;
- regularization of the activation values of model units, which reduces the need for dropout or other hard-regularization techniques.

Batch normalization is applied as follows to a batch $\mathscr{B}$ with m instances [20]:

(i) computes the mean and variance of the batch $\mathscr{B}$:

$$\mu_{\mathscr{B}} = \frac{1}{m}\sum_{i=1}^{m} x_i$$
$$\sigma_{\mathscr{B}}^2 = \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathscr{B}})^2$$

(ii) computes the normalized instance $\hat{x}_i$ for each $i \in \{1,2,...,m\}$:

$$\hat{x}_i = \frac{x_i - \mu_{\mathscr{B}}}{\sqrt{\sigma_{\mathscr{B}}^2 + \varepsilon}},$$

where ε is a very small value (e.g. machine epsilon) added to prevent division by zero.

(iii) computes the layer output for the instance $\hat{x}_i$:

$$y_i = \gamma\hat{x}_i + \beta = \mathrm{BN}_{\gamma,\beta}(x_i),$$

where γ and β are learnable parameters tuned during back-propagation.

Pooling Layer The pooling layers of a CNN implement a dimensionality reduction transformation designed to lower the number of trainable parameters for subsequent layers, allowing them to focus on larger areas of the input patterns while retaining most of the information they contain.

Given an input signal $x^{(m-1)}$ with K_m, a 1D pooling layer with pool size $P_m \in \mathbb{N}$ and strides $\alpha_m \in \mathbb{N}$ is a channel-wise operation like the following [6]:

$$x_o^{(m)}[t] = \kappa \cdot \left(\sum_p \left(x_o^{(m-1)}[\alpha_m \cdot t + p]\right)^{\rho}\right)^{1/\rho}$$

where $\kappa, \rho \in \mathbb{N}$ are fixed parameters depending on the type of pooling we are applying to the data.

It's worth noting that the use of $P_m = \alpha_m$ corresponds to dividing each channel of the input signal into non-overlapping P_m patches (regions) and replacing the values in each region with a single value calculated on the basis of ρ and κ.

In max pooling layers ($\rho = \infty, \kappa = 1$), the output of the above formula value is the maximum of the values found in the patch:

$$\lim_{\rho\to\infty}\left(\sum_p \left(x_o^{(m-1)}[\alpha_m \cdot t + p]\right)^{\rho}\right)^{1/\rho} = \max\left\{x_o^{(m-1)}[\alpha_m \cdot t + p] : p \in \{1,2,...,P_m\}\right\}$$

In average pooling layers on the other hand, which were also used in this study, ($\rho = 1, \kappa = 1/P_m$), the result is the average of the values:

$$x_o^{(m)}[t] = \frac{1}{P_m} \cdot \sum_p x_o^{(m-1)}[\alpha_m \cdot t + p]$$

7.4. RecConvSiameseNet

Let us now go into a detailed description of the neural network architecture used in the performed experiments.

Branches The proposed architecture (Figure 12), named RecConvSiameseNet, involves two information processing branches:

- one characterized by LSTM recurrent layers (Rec);
- the other featuring several 1D convolutional blocks (Conv), each containing a set of increasing 1D convolutional filters, a batch normalization layer and a 1D average pooling layer, followed by a final dense layer which flattens the feature maps extracted by the convolutional blocks.

These two branches receive as input two different types of audio features, extracted upstream during preprocessing, which are respectively:

- the more refined (243×39) frames containing MFCCs & deltas;
- the coarsest log-scaled Mel spectrum features contained in (243×128) frames.

The "Siamese" part in the name refers to the non-sequential nature of the network, which is typical of Siamese networks [24], although the proposed architecture does not have the usual characteristics of Siamese networks, the two branches being completely different from each other.

The basic idea, as mentioned earlier, is to combine the features that these two types of layers are capable to extract, in particular:

- LSTM branch being fed with MFCCs & deltas one (1×39) frame at time, and processing them in the same way, it is able to extract temporal-wise features from the cepstrum, capturing short-term and long-term relationship between frames;
- convolutional branch being fed with the (243×128) log-scaled Mel spectrum features in their entirety, it is capable of excerpting high-level features that span the entire audio.

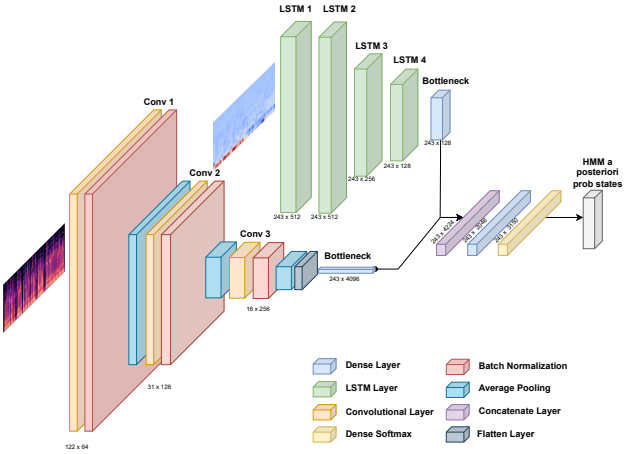

Figure 12: DNN-HMM Architecture

Further details on architecture and implementation of the two branches will be provided in the sections 8.1.1 on page 19 and 8.1.2 on page 20, where the autoencoder models used during the pre-training phase will be described, as the two RecConvSiameseNet branches were extracted from them prior to the training.

Network tail As shown in the Figure 12, after the feature extraction process done by the the two branches is completed, the LSTM branch result vector u_t is, at each timestep t, concatenated with the flattened feature maps extracted by the convolutional branch v (through repeat vector and concatenation layers), resulting in a new feature vector $w_t = [u_t|v]$. This feature vector is fed into a single network tail, composing of two time-distributed dense layers.

The first one has $\#\text{SPKR} \cdot \#\text{STATES\_SPKR} = 5040$ neuron units, and uses units Leaky ReLU activation function [36], with $\alpha = 0.1$:

$$\text{LeakyReLU}(x) := \begin{cases} x, & x \geq 0 \\ \alpha x, & x < 0 \end{cases}$$

The second one has $\#\text{SPKR} \cdot \#\text{STATES\_SPKR} = 5040$ neuron units too, but since it's the final output layer, it aims to predict the posterior probabilities

Table 3: Parameters of RecConvSiameseNet tail

Layer Type	Parameters	Shape
Concatenate layer		(1, 243, 4.224)
Dense		(1, 243, 5040)
LeakyReLU		(1, 243, 5040)
Dense		(1, 243, 5040)
Softmax		(1, 243, 5040)

$P(s_t = k \mid o_t)$ of each HMM state k given the current frame o_t, and thus uses the softmax activation function (like the one we described in previous section 6.1 on page 12).

Denoting with W_0, b_0 and W_1, b_1 the weights and biases of the first and second tail dense layers, respectively, the network output on each state can be expressed as:

$$P(s_t = k \mid o_t) = \sigma(a_t^T \cdot W_1 + b_1)_k, \text{ where:}$$

$$a_t = \text{LeakyReLU}(w^T \cdot W_0 + b_0)$$

Details about RecConvSiameseNet training procedure and will be described in section 9.1 on page 22, while the architecture of the tail can be seen in Table 3.

8. Pre-training

The concept of pre-training is inspired by human beings. Thanks to an innate ability, we don't have to learn everything from scratch. Instead, we transfer and reuse our old knowledge of what we have learned in the past to understand new knowledge and handle a variety of new tasks.

In Deep Learning, pre-training imitates the way human beings process new knowledge. That is: using model parameters of tasks that have been learned before to initialize the model parameters of new tasks. In this way, the old knowledge helps new models successfully perform new tasks from old experience instead of from scratch.

For all these reasons, we performed pre-training on both the two branches of the previously described (7 on page 13) neural network architecture.

8.1. Autoencoder

In this section, we are going to define the architecture of the CNN and LSTM autoencoders we used to pre-train the two branches of the network used in this experiment.

Autoencoders are an unsupervised artificial neural network class that learn how to efficiently encode an input vector, creating a compressed but meaningful "latent space" representation of it, in order to reconstruct it back obtaining a vector that is as close as possible to the original one. Furthermore, by design, autoencoders reduce data dimensions by learning how to ignore the noise in the data, thus they serve as very good de-noising tools.

They generally consists of four components:

Encoder: one or more stacked neural network layers $e_0, ..., e_h$ composing a function f, in which the model learns how to reduce the input dimensions and compress the input pattern x into an encoded representation $f(x)$.

Bottleneck: : the last encoder layer e_h, containing the compressed representation $f(x)$.

Decoder: one or more stacked neural network layers $d_1, ..., d_r$ (usually, the decoder is built in a mirror-like fashion with respect to the encoder, so $r = h$, d_1 has the same parameters as l_h, d_2 has the same as l_{h-1} and so on). These layers compose a decoder function g, in which the model learns how to reconstruct the original input data from the encoded representation $f(x)$ in such a way that $g(f(x))$ is as close to the original input x as possible.

Reconstruction Loss: exactly as in regular neural networks, autoencoders are trained with an optimization algorithm in order to minimize a loss function J; the only difference here is that in autoencoders the loss represents the distance between input and reconstructed output:

$$J = \frac{1}{|D|} \sum_{x \in D} L(x, \hat{x}),$$

where $\hat{x}$ is the reconstruction of the network on input x.

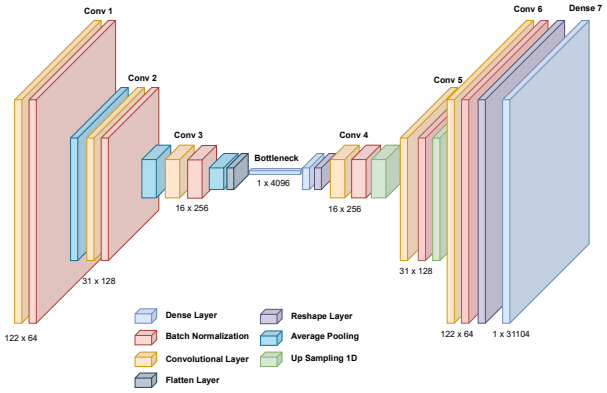

Figure 13: CNN Autoencoder

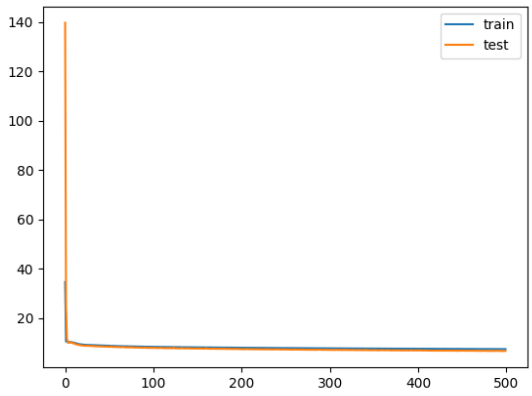

Figure 14: CNN Autoencoder Train/Validation Loss over the first 500 epochs.

Often, regularization techniques such as dropout [19] or regularization terms added to the autoencoder loss functions are used, since this has the double advantage of reducing overfitting and make the learned representation sparse, which is more suitable for different tasks other than input reconstruction, which is exactly the aim of pre-training. Another way to make the learned representation sparse is to add penalization term to the loss, based on the activation value of the bottleneck neurons:

$$J = \frac{1}{|D|} \sum_{x \in D} L(x, \hat{x}) + \lambda \sum_{i} a_i^{(h)},$$

where $a_i^{(h)}$ is the activation value of the i-th neuron of the bottleneck.

By virtue of the above, dropout and activation penalization techniques were applied to the autoencoders created for the pre-training phase of this study to make the learned audio representation as sparse as possible.

8.1.1. CNN Autoencoder. The autoencoder used to pre-train the parameters of the convolutional branch of the RecConvSiameseNet uses log-scaled Mel spectrum features extracted from each speakers'audio as input. For SI, it is important to derive features that contain the vocal characteristics of the speaker, and indeed the aim here is to provide the network with a global overview of high-level features of the audio, in contrast to the LSTM branch that aims to extract temporal-wise features.

SGD (Stochastic Gradient Descent) algorithm has been used with a learning rate of $\eta = 0.05$ to train the autoencoder for 1500 epochs with a batch size of 100 (14), clipping the L_2 norm of the gradient to 1 and the value to of weight gradient to 0.5, in order to reduce overfitting and avoid gradient exploding.

Furthermore, as briefly mentioned before, a dropout rate of 0.5 has been applied alongside bottleneck activity regularization to prevent overfitting and make the learned data representations as sparse as possible.

Looking at the structure in detail, we will explore layer by layer the structure of the encoder and decoder (excluding dropout layers).

Our model's encoder consists of 3 convolutional layers with an increasing number of filters, always followed by a batch normalization layer and by an average pooling layer that downsamples the input representation by taking the average value over the window defined by the pool size. The complete set of the encoder parameters is reported in table 4 on the following page.

The decoder is very similar to the encoder, but in reverse order, as it reconstructs the encoder's input based on encoder's output.

In order to match the dimension of the input, three transposed convolutional layers are needed in the decoder in combination with upsampling layers that repeats each temporal step r times along

Table 4: Parameters of CNN Encoder

Layer Type	Parameters	Shape
Input		(1, 243, 128)
Conv 1D	Filters, kernel, stride (64, 7, 2)	(122, 64)
Batch Normalization	Features = 64	(122, 64)
Average Pooling	Pool size = 2	(61, 64)
Conv 1D	Filters, kernel, stride (128, 5, 2)	(31, 128)
Batch Normalization	Features = 128	(31, 128)
Average Pooling	Pool size = 2	(16, 128)
Conv 1D	Filters, kernel, stride (256, 5, 1)	(16, 256)
Batch Normalization	Features = 128	(16, 256)
Average Pooling	Pool size = 2	(8, 256)
Flatten Dense		(1, 4096)

Table 5: Parameters of CNN Decoder

Layer Type	Parameters	Shape
Dense		(1, 4096)
Reshape		(16, 256)
Conv 1D Transpose	Filters, kernel, stride (256, 5, 1)	(16, 256)
Batch Normalization	Features = 256	(16, 256)
Up Sampling 1D	Size = 2	(32, 256)
Conv 1D Transpose	Filters, kernel, stride (128, 5, 2)	(64, 128)
Batch Normalization	Features = 128	(64, 128)
Up Sampling 1D	Size = 2	(128, 128)
Conv 1D Transpose	Filters, kernel, stride (64, 7, 2)	(256, 64)
Batch Normalization	Pool size = 2	(256, 64)
Reshape		(1, 16384)
Dense		(1, 31104)
Reshape		(1, 243, 128)

the time axis, upscaling the dimension of the reconstructed pattern, and a final triplet composed of reshape, dense and reshape layers. Decoder parameters are reported in Table 5, while complete model architecture can be visualized in Figure 13 on the preceding page.

8.1.2. LSTM Autoencoder. The autoencoder used to pre-train the parameters of the recurrent branch of the RecConvSiameseNet takes MFCCs & deltas features extracted from each speakers'audio as input, processing the frames one-by-one. The aim here is to extract temporal-wise features from the long-term and short-term relationships between the MFCC & deltas frames, thanks to the memory capabilities of LSTM layers.

Adadelta algorithm has been used with a learning rate of $\eta = 1$ (as suggested by the original paper [43]), $\rho = 0.95$ and $\varepsilon = 10^{-8}$ to train the LSTM autoencoder for 1000 epochs with a batch size of 200 (16 on the next page).

Similar to what we have seen with the convo-

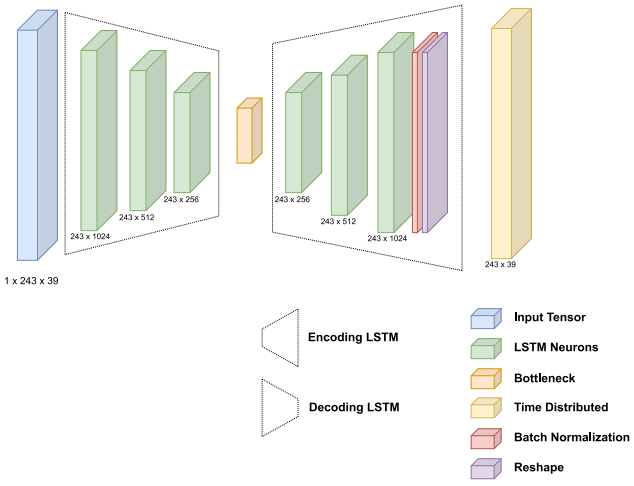

Figure 15: LSTM Autoencoder

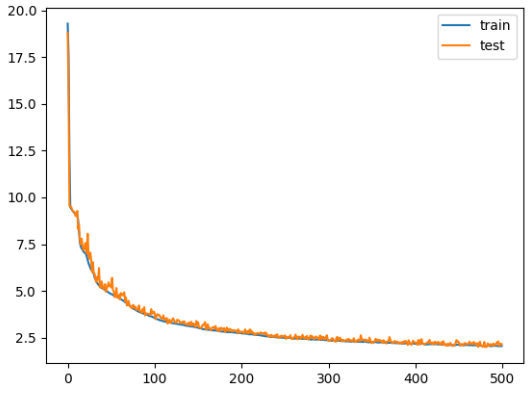

Figure 16: LSTM Autoencoder Train/Validation Loss over the first 500 epochs.

Table 6: Parameters of LSTM Encoder

Layer Type	Parameters	Shape
LSTM	Units = 512	(243, 512)
LSTM	Units = 512	(243, 512)
LSTM	Units = 256	(243, 256)
LSTM	Units = 128	(243, 128)
Bottleneck	Dimension = 128	

lutional autoencoder, bottleneck activity regularization has been applied, in order to reduce overfitting and make the learned data representations as sparse as possible.

Diving into the architecture, encoder consists of four LSTM layers with an increasing number of units and a bottleneck of 128 units, as seen in Table 6.

As in the convolutional autoencoder, the decoder is built to a mirrored fashion with respect to the encoder, as we can see in the Table 7 showing the decoder parameters. A reshape and a time distributed dense layer have been added at the end of the architecture in order to reconstruct the same shape as the fed input. The full LSTM autoencoder architecture is seen in Figure 15 on the preceding page.

Table 7: Parameters of LSTM Decoder

Layer Type	Parameters	Shape
LSTM	Units = 128	(243, 128)
LSTM	Units = 256	(243, 256)
LSTM	Units = 512	(243, 512)
LSTM	Units = 512	(243, 512)
Batch Normalization	Features = 1024	(243, 1024)
Time Distributed (Dense)	Units = 39	(243, 39)

8.1.3. Evaluation Metrics. Table 8 on page 23 illustrates the error rate analysis of the proposed autoencoders compared with some existing algorithms, such as the multiobjective evolutionary optimization algorithm [4], other deep convolutional and recurrent autoencoder neural networks [16], continual learning algorithms [30], enhancement parameter with a genetic algorithm [3], DTW [25] and HHSAE-ASR [1].

There are many possible metrics to evaluate the performances of autoencoders, but the general idea is to evaluate the reconstruction error like a distance between input and reconstructed input.

Mean Absolute Error MSE evaluates the absolute distance of the observations (entries of the dataset) to the predictions on a regression, taking the average over all observations. We use the absolute value of the distances so that negative errors are accounted properly:

$$\mathrm{MAE} = \frac{1}{n}\sum_{i=1}^{n}\left| y_i^{\mathrm{real}} - y_i^{\mathrm{pred}} \right|$$

Mean Squared Error Another way to deal with negative values is by squaring the distance, so that the results are positive. This is done by the MSE, and higher errors (or distances) weight more in the metric than lower ones, due to the nature of the power function:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}\left(y_i^{\text{real}} - y_i^{\text{pred}}\right)^2$$

Root Mean Squared Error A backlash in MSE is the fact that the unit of the metric is also squared, so if the model tries to predict price in $US\$$, the MSE will yield a number with unit $(US\$)^2$ which does not make sense. RMSE is used then to return the MSE error to the original unit by taking the square root of it, while maintaining the property of penalizing higher errors:

$$\text{RMSE} = \sqrt{\text{MSE}} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(y_i^{\text{real}} - y_i^{pred}\right)^2}$$

R^2 It is a statistical measure in a regression model that determines the proportion of variance in the dependent variable that can be explained by the independent variable. In other words, r-squared shows how well the the regression model fits the data. Denoting with $\bar{y}$ the mean of the real label values, and with y_i, f_i the real label value and model prediction on i-th instance, respectively:

$$\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i,$$

then the variability of the data set can be measured with two sums of squares formulas:

- the sum of squares of residuals, also called the residual sum of squares:

$$SS_{\text{res}} = \sum_i (y_i - f_i)^2 = \sum_i e_i^2$$

- The total sum of squares (proportional to the variance of the data):

$$SS_{\text{tot}} = \sum_i (y_i - \bar{y})^2$$

That being said, the most general definition of the coefficient of determination is:

$$R^2 = 1 - \frac{SS_{\text{res}}}{SS_{\text{tot}}}$$

In the best case, the modeled values exactly match the observed values, which results in $SS_{res} = 0$ and $R^2 = 1$. A baseline model, which always predicts $\bar{y}$, will have $R^2 = 0$. Models that have worse predictions than this baseline will have a negative R^2.

Analysis MSE and RMSE are very useful in understanding whether outliers generate noise in the prediction, while MAE is slightly less affected by them, so using the one or the other is a fair trade-off.

Optimising for MSE for example, means that the generated output values are symmetrically close to the input values, meaning that an higher-than-real value is penalised by the same amount as an equally lowered one.

As for the obtained results shown in 8 on the following page, it's worth noting that although the MAE and RMSE values of our autoencoders are particularly high if compared to the others mentioned above, this is fairly intentional and due both to the choice of not normalizing the input data, and to the applying of heavy regularization in order to make the learned weights and representations as sparse as possible and suited for usage in classification task. Nevertheless, it's also interesting to note how both autoencoders achieve good R^2 values, which are satisfactorily high in the case of the convolutional one and very high in the case of the recurrent one.

9. Experiment and Analysis

Below we report the experiments performed during model training and the obtained results in the final identification phase.

9.1. RecConvSiameseNetTraining

Having described all previous stages, we delve now into details of the training step for the model we presented in section 7.4 on page 17. RecConvSiameseNet was trained to minimize categorical cross entropy loss function for 800 epochs with Adadelta algorithm, learning rate $\eta = 1$, $\rho = 0.95$ and $\varepsilon = 10^{-8}$, and a batch size of 200. Graph of training/test loss during the epochs is shown in Fig-

Table 8: training values Error rate in Speech Processing.

Methods	MAE	RMSE	R^2
Multiobjective evolutionary optimisation algorithm [4]	N/A	1.43	N/A
Deep convolution encoder and LSTM-RNN [16]	N/A	1.38	N/A
Continual learning algorithms [30]	N/A	1.25	N/A
Genetic algorithm [3]	N/A	1.14	N/A
MFCC and DTW [25]	N/A	1.12	N/A
HHSAE-ASR [1]	N/A	1.087	N/A
Convolutional Autoencoder with Mel Spectrogram	4.570	6.227	0.8464
LSTM Autoencoder with MFCC	1.796	2.885	0.9983

ure 17, with the final resulting loss being 1.2096 on the training set and 1.2071 on the test set.

As the network architecture is particularly complex and deep, several regularization techniques were applied to reduce overfitting and increase the generalization capabilities of the model:

- for the convolutional branch: dropout rate = 0.5;
- for the LSTM branch: dropout rate = 0.7;
- in the first dense layer of the tail: dropout rate = 0.5;
- L_1 loss penalty on the final activation values with $\lambda = 10^{-5}$;
- L_1 loss penalty on the weight values of the first dense level of the tail, with $\lambda = 10^{-5}$.

Evaluation metrics As discussed before, the network is trained using the labels generated by audio

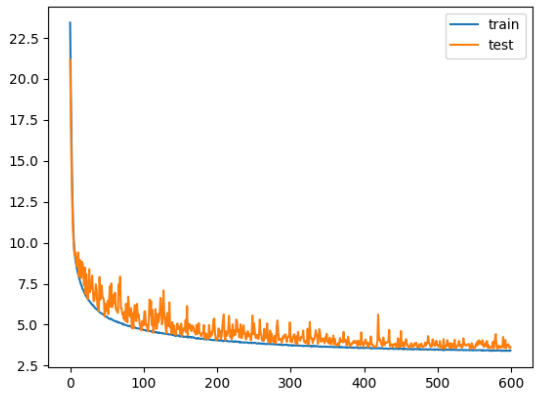

Figure 17: RecConvSiameseNet train/validation loss over the first 600 epochs.

forced alignment on the GMM-HMM acoustic models with Viterbi algorithm, so it learns to estimate the posterior probabilities:

$$P(s_t = k \,|\, o_t)$$

for each state k, given the frame o_t at each time step t. The training labels being artificially generated by GMM-HMM acoustic models, we surely not want the network to (hypothetically) perfectly or almost perfectly match the labels with its predictions, since this would make them completely useless, as they would be no more than an approximation of an already known model predictions (the GMM-HMM model). It follows from this reasoning that simple accuracy cannot be used as a metric to evaluate the goodness of a neural model underlying a DNN-HMM system, since we are not interested into evaluating the exact match between network-predicted most likely state and the Viterbi-predicted ones.

What we are really interested in quantifying in this context is the model's ability to predict as most likely states those from the acoustic model of a certain speaker, given an audio $o = o_1, o_2, ..., o_T$ belonging to the latter.

The number of states of each speaker GMM-HMM acoustic model being #STATES_SPKR = 8, we used three different metrics to evaluate the performances of RecConvSiameseNet (prior to the SI step), one being widely-used and two custom

Table 9: RecConvSiameseNet's metrics results

Top-8	Top-8 speaker	# States in top-K
0.9698	0.9968	0.9690

Table 10: Full DNN-HMM SI results

Speaker identification accuracy
0.9852

ones. Each metric is calculated frame-wise, having 0 as a minimun score and 1 maximum score on each of them.

Top-K accuracy: the widely-used Top-K accuracy rate with $k = 8$, for each training instance frame o_t, it scores 1 if the real state label y_t is in the top-8 highest probability $P(s_t = q \mid o_t)$ values predicted by the network, and 0 otherwise;

Top-K speaker accuracy: a slightly modified Top-K accuracy rate with $k = 8$, for each training instance frame o_t, it scores 1 if the real state label y_t or any state label from the same speaker is in the top-8 highest probability $P(s_t = q \mid o_t)$ values predicted by the network, and 0 otherwise;

Number of speaker states in Top-K: a custom metric that counts, for each training instance frame o_t, how much states of the corresponding speaker are in the top-K most likely ones predicted by the network (with $k = 8$), divided by the number of states for each speaker #STATES_SPKR $= 8$ (so we have 0 in case no right state is in the top-8, 0.5 if exactly 4 are, and so on).

Among all the above metrics, *number of speaker states in top-K* is arguably the most meaningful of the three, since it carries the most information, taking into account all the states and giving all frames a different weight between 0 and 1, based on how the network performed on that specific frame. On the other hand, *top-K accuracy* doesn't takes into account every state, and *top-K speaker accuracy* doesn't give different weight if model predicts more than 1 speaker state in the top-K.

Table 9.1 shows values of each presented metric for the final model, which appear to be quite high.

9.2. Final results

This brief section presents the final results of the DNN-HMM system, which appear to be comparable if not slightly better than other DNN-HMM SI systems [21] or LSTM/CNN-based ones [15], although the differences between the used dataset in terms of composition, dimension and distribution make comparisons very difficult. Aforementioned results are shown in Table 9.2.

9.3. Considerations and future works

To the best of our knowledge, this is the first study that goes to apply DNN-HMM for text independent SI using a non-sequential network architecture or using both convolutional and LSTM layers.

As promising as the performances achieved are, the scope for improvement of the network and the variety of experiments that can still be carried out remains very high:

Fine-tuning: re-training the model that has already been trained on different datasets to evaluate performances of transfer learning in DNN-HMM context;

Parameter tweaking: additional parameter and hyperparameter tweaking on the proposed architecture and speaker acoustic models to try improving performance even further;

Data augmentation: training the model with additional augmented data to try improving the model generalization capability even more;

Variations of the architecture: trying out variations of the architecture, for example replacing the current convolutional branch with more complex architectures like SincNet one [34] or other convolutional architectures;

Alternative/Additional features: training the model using other typologies of features like LPCCs, or expanding the existing architectures to take advantage of them (e.g. adding a new branch);

Application to other datasets: applying the same methodology and network architecture to other datasets, especially in-the-wild ones, and to the noise-added TIMIT.

10. Acknowledgements

This research was supported by the Computer Science Department of University of Salerno, by Prof. Michele Nappi's Biometric and Image Processing Laboratory and by Ph.D. Student Chiara Pero.

References

[1] Mohammed Hasan Ali et al. "Harris Hawks Sparse Auto-Encoder Networks for Automatic Speech Recognition System". In: *Applied Sciences* 12.3 (2022). ISSN: 2076-3417. DOI: 10.3390/app12031091. URL: https://www.mdpi.com/2076-3417/12/3/1091.

[2] Muhammad Asadullah and Shibli Nisar. "A Silence Removal and Endpoint Detection Approach for Speech Processing". In: Sept. 2016.

[3] Mazhar Awan et al. "Real-Time DDoS Attack Detection System Using Big Data Approach". In: *Sustainability* 13 (Sept. 2021), p. 10743. DOI: 10.3390/su131910743.

[4] Mazhar Javed Awan et al. "Image-Based Malware Classification Using VGG19 Network and Spatial Convolutional Attention". In: *Electronics* (2021). ISSN: 2079-9292. DOI: 10.3390/electronics10192444.

[5] Xiao Bao et al. "An investigation of high-resolution modeling units of deep neural networks for acoustic scene classification". In: *2017 International Joint Conference on Neural Networks (IJCNN)* (2017). DOI: 10.1109/IJCNN.2017.7966232.

[6] Gustavo Carneiro, Jacinto Nascimento, and Andrew P. Bradley. "Chapter 14 - Deep Learning Models for Classifying Mammogram Exams Containing Unregistered Multi-View Images and Segmentation Maps of Lesions". In: *Deep Learning for Medical Image Analysis*. Ed. by S.Kevin Zhou, Hayit Greenspan, and Dinggang Shen. Academic Press, 2017, pp. 321–339. ISBN: 978-0-12-810408-8. DOI: https://doi.org/10.1016/B978-0-12-810408-8.00019-5. URL: https://www.sciencedirect.com/science/article/pii/B9780128104088000195.

[7] Aleksej Chinaev and Reinhold Haeb-Umbach. "Map-based estimation of the parameters of a Gaussian Mixture Model in the presence of noisy observations". In: (2013), pp. 3352–3356. DOI: 10.1109/ICASSP.2013.6638279.

[8] C. Chueh and Jen-Tzung Chien. "Maximum Entropy Modeling of Acoustic and Linguistic Features". In: (2006). DOI: 10.1109/ICASSP.2006.1660207.

[9] Sakhia Darjaa et al. "Effective Triphone Mapping for Acoustic Modeling in Speech Recognition." In: *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH* (2011). DOI: 10.21437/Interspeech.2011-190.

[10] N. E. Day. "Estimating the Components of a Mixture of Normal Distributions". In: *Biometrika* 56.3 (1969), pp. 463–474. ISSN: 00063444. URL: http://www.jstor.org/stable/2334652 (visited on 07/11/2022).

[11] John Robert Deller, John G. Proakis, and John H. L. Hansen. "Discrete-Time Processing of Speech Signals". In: (1993).

[12] Mangesh S. Deshpande and Raghunath S. Holambe. “Text-Independent Speaker Identification Using Hidden Markov Models”. In: (2008). DOI: 10.1109/ICETET.2008.46.

[13] Nikos Fakotakis, Kallirroi Georgila, and Anastasios Tsopanoglou. “A continuous HMM text-independent speaker recognition system based on vowel spotting”. In: (1997). URL: http://www.isca-speech.org/archive/eurospeech_1997/e97_2347.html.

[14] Farheen Fauziya and Geeta Nijhawan. “A Comparative Study of Phoneme Recognition using GMM-HMM and ANN based Acoustic Modeling”. In: *International Journal of Computer Applications* 98 (July 2014), pp. 12–16. DOI: 10.5120/17186-7366.

[15] Ye Feng and Yang Jun. “A Deep Neural Network Model for Speaker Identification”. In: *Applied Sciences* (2021). DOI: 10.1007/s00521-021-06226-w.

[16] Faria Ferooz et al. “Suicide Bomb Attack Identification and Analytics through Data Mining Techniques”. In: *Electronics* 10.19 (2021). ISSN: 2079-9292. DOI: 10.3390/electronics10192398. URL: https://www.mdpi.com/2079-9292/10/19/2398.

[17] Eric Fosler-Lussier and Jeremy Morris. “Crandem systems: Conditional random field acoustic models for hidden Markov models”. In: (2008). DOI: 10.1109/ICASSP.2008.4518543.

[18] Lori F. Garofolo John S. and Lamel et al. *DARPA TIMIT - Acoustic-Phonetic Continuous Speech Corpus CD-ROM*. Ed. by Nist. 1993.

[19] Geoffrey E. Hinton et al. “Improving neural networks by preventing co-adaptation of feature detectors”. In: *ArXiv* abs/1207.0580 (2012).

[20] Sergey Ioffe and Christian Szegedy. “Batch normalization: Accelerating deep network training by reducing internal covariate shift”. In: (2015).

[21] Shahin Ismail et al. “Speaker identification and verification using Gaussian mixture speaker models”. In: *Neural Computing and Applications* (2021). DOI: 10.1007/s00521-021-06226-w.

[22] Rashid Jahangir et al. “Speaker identification through artificial intelligence techniques: A comprehensive review and research challenges”. In: *Expert Systems with Applications* (). DOI: https://doi.org/10.1016/j.eswa.2021.114591.

[23] Ali Muayad Jalil, Fadhil Sahib Hasan, and Hesham Adnan Alabbasi. “Speaker identification using convolutional neural network for clean and noisy speech samples”. In: (2019). DOI: 10.1109/CAS47993.2019.9075461.

[24] Gregory R. Koch. “Siamese Neural Networks for One-Shot Image Recognition”. In: (2015).

[25] Qin Li et al. “MSP-MFCC: Energy-Efficient MFCC Feature Extraction Method With Mixed-Signal Processing Architecture for Wearable Speech Recognition Applications”. In: *IEEE Access* 8 (2020), pp. 48720–48730. DOI: 10.1109/ACCESS.2020.2979799.

[26] Yi Liu et al. “Comparison of Multiple Features and Modeling Methods for Text-dependent Speaker Verification”. In: (2017). DOI: https://doi.org/10.48550/arXiv.1707.04373.

[27] *Long short-term memory*. URL: https://en.wikipedia.org/wiki/Long_short-term_memory (visited on 07/18/2022).

[28] A.A. Markov. *Theory of algorithms*. Moscow: Academy of Sciences of the USSR, 1954.

[29] *Mel scale*. URL: https://en.wikipedia.org/wiki/Mel_scale (visited on 07/17/2022).

[30] Karrar Neamah et al. "Discriminative Features Mining for Offline Handwritten Signature Verification". In: *3D Research* 5 (Feb. 2014). DOI: 10.1007/s13319-013-0002-3.

[31] Mariusz Owsianny and Piotr Francuzik. "Evaluation of speech recognition system for Polish". In: *Speech and Language Technology* (2012).

[32] Joseph W Picone. "Signal modeling techniques in speech recognition". In: *Proceedings of the IEEE* 81.9 (1993), pp. 1215–1247.

[33] K. Sreenivasa Rao, V. Ramu Reddy, and Sudhamay Maity. *Language Identification Using Spectral and Prosodic Features*. Springer International Publishing, 2015.

[34] Mirco Ravanelli and Y. Bengio. "Speaker recognition from raw waveform with SincNet". In: (2018).

[35] *Recurrent neural network*. URL: https://en.wikipedia.org/wiki/Recurrent_neural_network (visited on 07/18/2022).

[36] Joseph Redmon et al. "You Only Look Once: Unified, Real-Time Object Detection". In: (2016). DOI: 10.1109/CVPR.2016.91.

[37] Douglas A. Reynolds. "Speaker identification and verification using Gaussian mixture speaker models". In: (1995). DOI: 10.1016/0167-6393(95)00009-D.

[38] H. Robbins. "A Stochastic Approximation Method". In: *Annals of Mathematical Statistics* (1951). DOI: https://doi.org/10.1214/AOMS%2F1177729586.

[39] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning representations by back-propagating errors". In: *Nature* (1986). DOI: https://doi.org/10.1038/323533a0.

[40] Bowen Shi, Shane Settle, and Karen Livescu. "Whole-Word Segmental Speech Recognition with Acoustic Word Embeddings". In: (2020). DOI: 10.48550/ARXIV.2007.00183.

[41] Dan Su, Xihong Wu, and Lei Xu. "GMM-HMM acoustic model training by a two level procedure with Gaussian components determined by automatic model selection". In: (2010). DOI: 10.1109/ICASSP.2010.5495122.

[42] Andrew Viterbi. "Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm". In: *Information Theory, IEEE Transactions on* 13 (May 1967), pp. 260–269. DOI: 10.1109/TIT.1967.1054010.

[43] Matthew D. Zeiler. "ADADELTA: An Adaptive Learning Rate Method". In: (2012). DOI: https://doi.org/10.48550/arXiv.1212.5701.

[44] P. Zolfaghari and Tony Robinson. "Formant analysis using mixtures of Gaussians". In: Nov. 1996, 1229–1232 vol.2. DOI: 10.1109/ICSLP.1996.607830.