PADERBORN UNIVERSITY
The University for the Information Society

Faculty for Computer Science, Electrical Engineering and Mathematics

Faculty for Computer Science, Electrical Engineering and Mathematics
Paderborn University
Communications Engineering
Prof. Dr.-Ing. Reinhold Häb-Umbach

Project Work

From SAD to ASR on the Fearless Steps Data

by

Bibash Thapaliya
Matr.-No.: 6898408

Dheeraj Rajashekar Poolavaram
Matr.-No.: 6898078

Saad Bin Abdul Mannan
Matr.-No.: 6897506

Padmanaban Krishnan
Matr.-No.: 6870391

Vivek Kandimalla
Matr.-No.: 6862482

Supervisor: M.Sc. Jens Heitkämper, M.Sc. Thilo von Neumann
Filing date: 29.10.2021
Number: EIM/NT – Summersemester 2021

Declaration

This Project Work, with the title "From SAD to ASR on the Fearless Steps Data", is the result of my own work and includes nothing that is the outcome of work done in collaboration, except where specifically indicated. It has not been published yet or submitted, in whole or in part, for a degree at any other university.

Paderborn, 29.10.2021 Bibash.T

(Bibash Thapaliya)

Paderborn, 29.10.2021 Dheeraj Poolavaram

(Dheeraj Rajashekar Poolavaram)

Paderborn, 29.10.2021 Saad Bin Abdul Mannan

(Saad Bin Abdul Mannan)

Abstract

Human-machine interactions through speech are one of the great inventions of modern times. To enhance this interaction, speech processing plays a crucial role in extracting meaningful information from audio signals in the modern-day. Depending on the end goal, long recordings with multiple speakers and adequate silences are commonly preferred to design a robust system for speech recognition. The Fearless Steps - 02 (FS-02) Corpus from NASA's Apollo space mission is investigated in our project to implement such an autonomous system capable of identifying the speaker and transcribing the speech content.

The proposed system is divided into three sub-tasks: Speaker Activity Detection (SAD), Speaker Identity Detection (SID), and Automatic Speech Recognition (ASR), each involving a supervised learning strategy. In this project paper, we describe innovations and challenges faced in the development of our work and present revised baseline results handled by different network architectures.

For the SAD, the task is to differentiate the tangible speech from a non-speech in audio. We achieved a DCF of 2.44% on the Development dataset of FS-02. To identify a speaker from the voice attributes, the SID system is developed. When evaluated on the FS-02 Development dataset, achieved a Top-5 Accuracy of 91.65% and an Accuracy of 76.89%. In ASR, the task is to incorporate the speaker identification in an end-to-end speech recognition system. The experiments on the FS-02 dataset show that the model achieves WER of 32.9% with the Baseline Transformer model.

For integrating the sub-systems, firstly a Speaker Diarization (SD) which is the process of identifying who is speaking when in a stream of audio, is carried out by using the temporal information from the SAD system combined with the speaker predictions from the SID system achieves a DER of 56.78%. Further, the available SAD information is incorporated into pre-trained ASR model and achieves a WER of 59.4%. The final integration of all the sub-systems leads to the *who spoke when and what was the content of the speech utterance.*

Contents

1 Introduction

The usage of a Neural Network (NN) has made significant breakthroughs in the field of scientific computations. As in other fields of scientific research, many tasks from the domain of speech processing using NNs with the correct data and network architectures can be performed. Additionally, modern hardware has led to improved speeds, and lower computational efforts in the processing of speech-related tasks [Jog+20].

The main objective behind this project is to implement a system that should accurately transcribe the detected speech content. Additionally, the speech must be distinguished from the non-speech content, and the speaker of the detected speech must be identified. To realize this system, it has been divided into three sub-tasks as shown in Figure 1.1:

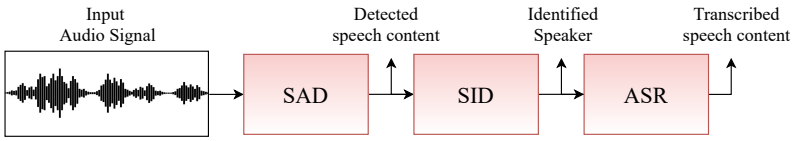


Figure 1.1: Speech Recognition System

Firstly, Speaker Activity Detection (SAD) is performed. The objective of SAD is to discern the speech from the non-speech parts of the audio that serves the primary input as speech-only information to perform further tasks. After performing SAD, we worked with Speaker Identity Detection (SID), which is the task of identifying a speaker from attributes of voices [LLL20]. Specifically, provided with an utterance of variable duration, the system assigns it to the best matching speaker in the speaker library. Finally, a Automatic Speech Recognition (ASR) is applied. This technology allows human beings to use their voices to communicate with a computer interface so that, in its most sophisticated variations, the interaction resembles normal human conversation.

To investigate the SAD problem, we used long duration audio files from the Fearless Steps - 02 (FS-02) Corpora, as they are continuous and natural. Since any sound made by a human from the vocal cord is considered speech and the non-speech includes all other sounds, it is challenging to distinguish between them. Moreover, the task gets complicated for a database like the FS-02, where there is a combination of short silences of milliseconds with more extended silences of hours in the audio stream [Zia+14]. However, these problems are tackled by using different architectures of NNs.

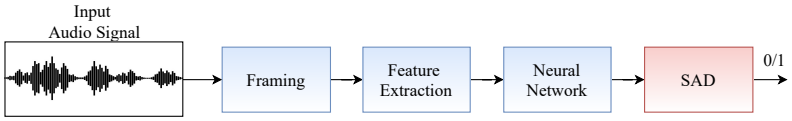


Figure 1.2: Speaker Activity Detection (SAD)

Figure 1.2 depicts the general processing of the SAD task. The input audio signal is pre-processed, segmented and the features are extracted before passing it to the neural network to learn the patterns for speech and non-speech activities. In the end, the network should generate activities, whether there is speech in a particular frame or not. Here, different architectures are chosen for comparison [LLL20].

Earlier, the SID baseline system developed for Fearless Steps - 01 (FS-01) used i-Vectors and x-Vectors for front-end processing [Jog+20]. These systems were suitable to the FS-01 SID data since it had at least 10 seconds of speaker activity. However, the current Fearless Steps - 02 (FS-02) SID data was altered to around 4 seconds per speaker on average, leading these embedded systems to be inadequate. Further, an alternate baseline system, more suited to the revised FS-02 SID data, the SincNet architecture making use of the speech data directly by performing time-domain convolutions between the input speech waveform and the SincNet filters were proposed [RB19]. However, to improve the earlier results and have the SincNet system as the baseline, a ResNet architecture based NN model operating on the speech features extracted using Mel-filterbank energy features has been proposed in [LLL20]. Figure 1.3 depicts a simple workflow for the SID task.

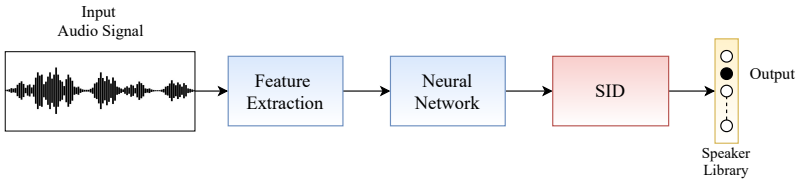


Figure 1.3: Speaker Identity Detection (SID)

Figure 1.4 shows a typical block diagram of ASR system. The main stages involved in the ASR process are feature extraction, training the acoustic model (Encoding), and matching (Decoding).

The first step of extracting the features of the acoustic signal is similar to the other two tasks. MFCC feature extraction algorithm has been used in this stage. The encoding part is where the acoustic model of the speech features is generated. An acoustic model represents the mathematical relationship between an audio signal and the phonemes or other linguistic units that make up speech. In the decoding part, the model is ready to decode (transcript) the audio clips into words. It is mainly backed by an acoustic model and a language model. A Language Model (LM) is the statistical representation of a sequence of words and sentences. For a given sequence of words, it assigns a probability to the whole sequence. The LM generates context that helps to distinguish between words and phrases that sound similar.

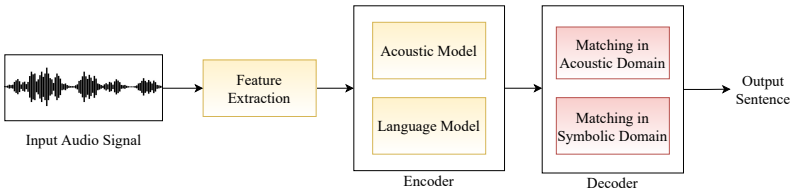


Figure 1.4: A block diagram of Speech Recognition system

The ASR task in this project uses the Transformer architecture model [Vas+17] to perform encoding and decoding operations on the given audio data. We also investigated the effects of including speaker information in the Transformer base model in the form of one-hot embedding and x-vectors. The implementations and comparisons of all these experiments are performed using the ESPnet toolkit [Wat+18a].

The process of identifying *who spoke when* in a speech recording involving multiple speakers is known as Speaker Diarization (SD). This task is challenging since the recording involves variable duration, rapid change of speakers, and overlapping speech content [ZWZ20]. Most commonly found SD systems make use of different clustering algorithms such as Agglomerative Hierarchical Clustering (AHC), Density Peak Clustering Algorithm (DPCA) using i-vectors and x-vectors [ZWZ20],[Gor+20]. Initially, in our task, we tried to implement the unsupervised approaches using x-vectors which did not lead to satisfactory results due to a higher correlation between the speaker classes. Further, we investigated the diarization by continuing with the supervised learning approaches implemented for the SAD and SID systems. Further, the available speech activity intervals from the pre-trained SAD system are incorporated into the pre-trained ASR model. This incorporation generates speech transcriptions from the ASR model. The basic workflow for the implemented SD system is as shown in Figure 1.5.

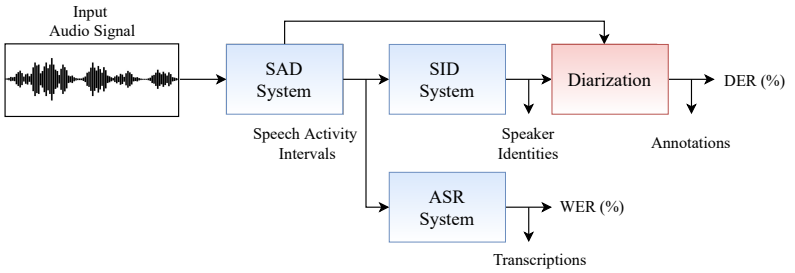


Figure 1.5: A block diagram of the implemented SAD to ASR system

The rest of the work is organized as follows. Chapter 2 provides a basic theory required for this project work, starting with the feature extraction methods for the audio data, fundamentals of neural nets, and the evaluation metric formulations. Chapter 3 deals with the structure of the Fearless database. Chapter 4 describes the architectures followed throughout the work. Finally, the evaluation and results are reported and discussed in Chapter 5, while conclusions are summarized in Chapter 6.

2 Basics

This chapter will explain several topics employed throughout our project work, giving a brief insight and a better understanding of the systems and architectures incorporated and implemented.

2.1 Feature Extraction

In general, speech is a varying time signal. The characteristics of this signal remain stationary only for a short period, where the analysis of the signal based on the waveform is complex. To obtain statistically significant information from the time domain signal, it is essential to have mechanisms for reducing the information that contains a relatively small number of parameters or features. These features are capable of describing the speech signal characteristically, allowing features corresponding to similar attributes to be grouped, which further help in the comprehension of the speech signal [ST13].

In this project work, we use Short Time Fourier Transform (STFT) which allows us to perform a time-frequency analysis, generating representations of both the time and frequency content of the signal [Raj20]. For the computation of STFT, the signal is divided into smaller blocks by multiplying with one of several window functions (ex. Rectangular, Hamming, Hann). We make use of the Hamming window of length 400 in this project work. Further, the windowed blocks of the signal are transformed into the frequency domain using the Discrete Fourier Transform (DFT) to obtain STFT spectrum as a function of the frequency bin k and the time shift n_o [US20]:

$$\mathbf{X}[k, n_o] = \sum_{n=0}^{N_w-1} \mathbf{x}(n_o + n)\mathbf{w}(n)e^{-j\frac{2\pi}{N_w}kn} \tag{2.1}$$

here in Equation 2.1, $\mathbf{X}[k, n_o]$ is the STFT signal, $\mathbf{x}(n)$ is the time domain signal, $\mathbf{w}(n)$ is the window function, N_w corresponds to the length of the window function and n is the summation index.

When the changing STFT spectra are plotted as a function of the time index n_0, we obtain a spectrogram which is the visual representation of the transformed speech signal.

The Mel-scale is a hearing-oriented frequency scale obtained by a logarithmic transformation on the frequency signal obtained from Equation 2.1. The main purpose of using the Mel-scale is to recreate the non-linear human ear perception of sound by having a higher resolution at lower frequencies and a lower resolution at higher frequencies. The obtained spectrogram is transformed to Mel scale using the transformations between Mel-filterbank energy features

m and Frequency f as below [Fay16].

With a reference value set at 1000 Hz = 1000 Mel, the approximate analytical equations mentioned below map the frequencies linearly up to 1000 Hz and logarithmic at higher frequencies :

$$\mathrm{m} = 2595 \log_{10} (1 + \frac{\mathrm{f}}{700}) \tag{2.2}$$

$$\mathrm{f} = 700(10^{\mathrm{m}/2595} - 1) \tag{2.3}$$

The Mel-bands that come from the Mel-filterbanks is highly correlated with subsequent and previous ones, causing difficulty distinguishing between features. To overcome the mentioned dependency, the features are de-correlated to different Mel-bands by a linear transformation with Discrete Cosine Transform (DCT). This dimensionality reduction algorithm provides distinct and uncorrelated features called Mel-Frequency Cepstral Coefficient (MFCC) which leads to a varying number of features depending on the audio data we use.

The SID task incorporates the Mel-filterbank energy features, and the SAD and ASR make use of the MFCC features in this project work.

2.2 Neural Networks

The Neural Network (NN) is an algorithm used in machine learning. It resembles the neuronal cells in a human brain [Rei20]. The human brain consists of 10^{11} computing *neurons* that communicate with one another through inter-neuron connection called *synapses*. Each neuron receives information from many thousands of other neurons. All the information is summed together, and if the information exceeds the threshold, then the neuron generates an activation potential passed to the other neuron through *axon*.

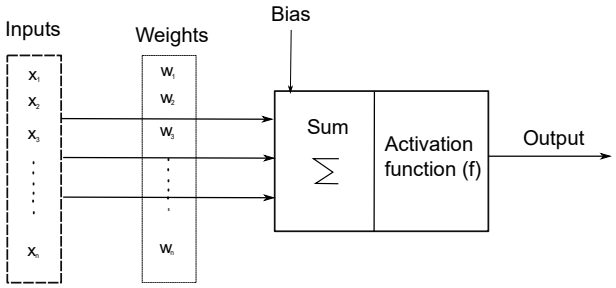


Figure 2.1: Simple Neural Network (NN)

In the same way, the NN consists of simple computing input units/nodes and exchange the information from one unit to other units through weight connectors and a bias unit. An input layer groups several input units together. The net activation is a weighted sum of input units. Then, it is passed through a nonlinear activation function. The NN is defined as

the transformation of input vectors $\mathbf{x} = (x_1, x_2, ...x_D)^T$ to the output $\mathbf{y}$ with the learnable parameter $\mathbf{w} = (w_1, w_2, ...w_k)^T$ as shown in the figure 2.1 [Rei20]:

$$\mathbf{y} = f(\mathbf{x}; \mathbf{w}), \tag{2.4}$$

where $\mathbf{w}$ are the weight parameters of the network.

A supervised learning approach is followed in our project. This is trained on data that has been labeled for a specific output/target. It splits into two categories: classification and regression.

In the classification problem, the class labels are assigned to the training data. The binary and multi-class classification problems are used in our work. The binary classification problem deals with two data classes so that the target/labels are either '0' or '1'. The multi-class problems deal with categorical data (more than two classes of data), which cannot be handled directly by machine learning algorithms. The model expects a categorical input and predicts categorical output data. Therefore, there is a need to represent the data in a form with which NN can work. For that purpose, firstly, there is a mapping of each data class to a different integer. The integer is further converted to a one-hot binary vector with a length equal to the total number of classes, a '1' for active speakers and a '0' for every inactive speaker. These encoded vectors serve as the targets/labels for the NN models in SID task.

Normally, the NN input vectors are the features extracted from the data, and the weight parameters are initialized with random values. The estimated output of the network is compared with the targets using the loss or error function. The loss function describes the variability of the estimated output to the target values. The loss is minimized by updating the weight parameters in the network using a back-propagation algorithm that is based on the gradient descent [DHS01]:

$$\boldsymbol{\nabla}\mathbf{w} = -\eta \frac{\partial J}{\partial \mathbf{w}} \tag{2.5}$$

here, J is a loss function, η is the learning rate that indicates the relative size of the change in weights.

If the training runs for the entire dataset, it is called an *epoch*. The training runs for multiple epochs to minimize the loss function. In our project, we use two loss functions. The *Binary Cross Entropy* loss which is given by [HW20]

$$J_{bce} = \frac{-1}{M} \sum_{m=1}^{M} [y_m \cdot \log_2(h_\theta(x_m)) + (1 - y_m \cdot \log_2(1 - h_\theta(x_m))], \tag{2.6}$$

here, M is number of training samples, y_m is target label either 0 or 1, x is input feature vector and h_θ is network weights. The *Categorical Cross Entropy* loss which is given by

$$J_{cce} = \frac{-1}{M} \sum_{k=1}^{K} \sum_{m=1}^{M} y_m^k \cdot \log_2(h_\theta(x_{m,k})) \tag{2.7}$$

here, K is the number of classes, and y_m^k is the $k - th$ class target label.

2.2.1 Multi-Layer Perceptron

The Multi-Layer Perceptron (MLP) is a basic NN model. The single hidden layer MLP is shown in Figure 2.2 consisting of the input layer, hidden layer, and output layer. The hidden layer is situated between the input and output layers. The term 'hidden' refers to the fact that their values are not visible at the network's input or output [Rei20].

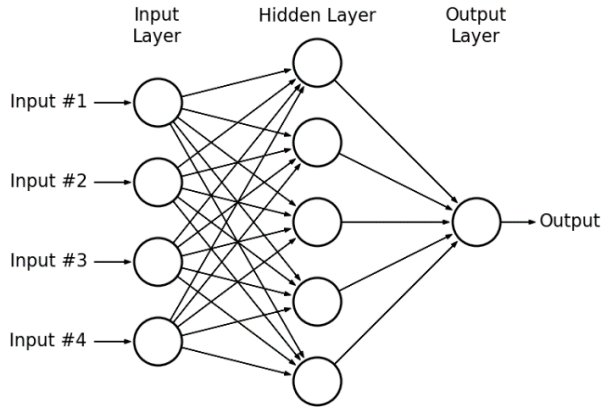


Figure 2.2: Multi-Layer Perceptron (MLP) [Has+15]

The most used activation function in the hidden layer is Rectified Linear Unit (ReLU) function. The output of the ReLU activation function is zero when the input vector x is less than zero and is equal to x when x is above or equal to zero

$$g(x) = \max(0, x) \tag{2.8}$$

where g represents the activation function, the output of the hidden layer is the input to the last layer. In our project work, we used the Sigmoid and Softmax activation function in the output layer. The output for both activation functions is between 0 and 1 and can therefore estimate a probability.

Sigmoid function

This activation function output is in the range of 0 and 1. The Sigmoid function is used for binary classification problems

$$g(x) = \frac{1}{1 + e^{-x}} \tag{2.9}$$

Softmax function

This activation function output is in the range of 0 and 1. The Softmax function is used for multi-class classification problems. An extension of the above Sigmoid function for K classes of data. For an $i - th$ class of data, the Softmax can be computed as below:

$$L_{S_i} = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}} \tag{2.10}$$

Additive Margin Softmax function

An extended version of the Softmax function as described in 2.2.1, is the Additive Margin Softmax (ADMS) Softmax which introduces a margin between the classes of data to maximize the distance between different classes and increase the concentration of the same classes of data [Wan+18]. The modified computation is shown below:

$$\psi(\theta) = \cos\theta - m \tag{2.11}$$

where $\psi(\theta)$ is the decision margin between classes, m is a positive integer that modifies the classification margin accordingly, and θ is the angle between different data classes. m is fixed at the beginning of the training and remains the same for all values of θ. It is introduced into the loss function 2.12 by Equation 2.11. For an $i-th$ class of data, the ADMS Softmax (L_{AMS_i}) can be computed as below:

$$L_{AMS_i} = -\frac{1}{n}\sum_{i=1}^{n}\log\frac{e^{s\cdot(\cos\theta_{y_i}-m)}}{e^{s\cdot(\cos\theta_{y_i}-m)} + \sum_{j=1,j\neq y_i}^{c} e^{s\cdot\cos\theta_j}} \tag{2.12}$$

2.2.2 Convolutional Neural Network

The Convolutional Neural Network (CNN) can be used for high-dimensional data [Rei20]. In MLP, each hidden node contains scalar weights. However, in the CNN, each node contains high dimensional planes for weights called the kernel, and input and output, known as the feature map/channel. A kernel is swept over the feature map from left to right and from top to bottom as shown in Figure 2.3. The size of the output feature map is increased by choosing the different sizes of kernels, strides, and padding. For further details, refer [DV18]. The output of the convolutional layer is calculated by [Rei20]

$$S(i,j) = \sum_{m}\sum_{n} I(i+m, j+n)K(m,n), \tag{2.13}$$

where $I(i,j)$ is the input feature vectors and $K(m,n)$ is the kernel. The following steps describes the CNN operation of the figure 2.3 [Kir+19].

1. The high-dimensional features extracted from the training data is fed into the input layer.

2. Each kernel of the convolution layer performs a linear convolution between high dimensional features and the corresponding kernel to generate the input feature map.

3. The input feature map is passed through the activation function to produce the output feature map.

4. A pooling layer reduces the dimension of the output feature map, which will be explained in the following subsection 2.2.2.

5. The output of the pooling layer is the input to a fully connected layer. The process of the last two layers is the same as the MLP network.

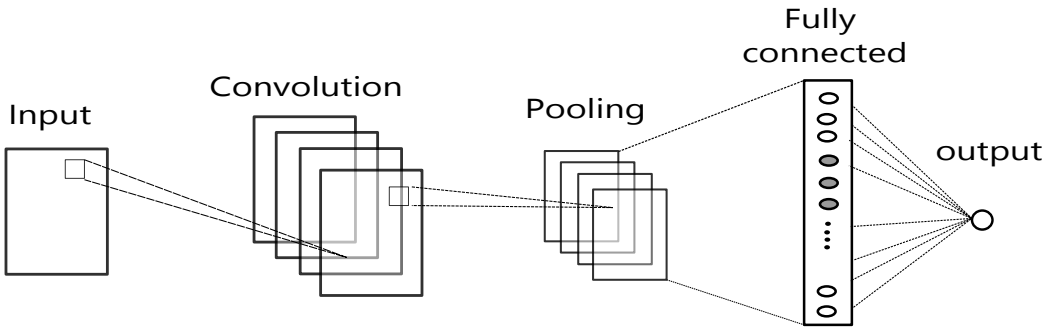


Figure 2.3: Convolutional Neural Network (CNN)

Pooling

As in figure 2.3, a pooling layer is used to reduce the dimension of the feature map. The idea behind the pooling operation is to extract the relevant information from the feature map and discard irrelevant information. Commonly, max pooling and average pooling are used.

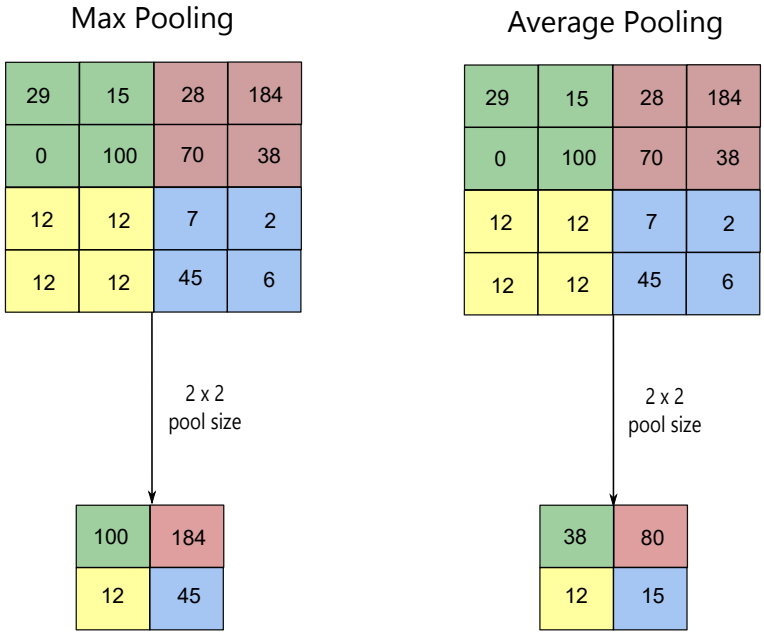


Figure 2.4: Pooling operation. This image is adapted from [MY19]

Figure 2.4 shows the operation of max and average pooling. Max pooling computes the maximum value, and the average pooling computes the mean value over the kernel size of 2×2.

2.2.3 Residual Neural Network

The increasing layers in CNN lead to a problem of vanishing/exploding gradients which causes the saturation of accuracy during the training of an NN [HS15], [SGS15]. However, a deep residual learning framework is addressed to overcome gradient degradation where

residual mapping is used to connect two layers explicitly. The advantage of ResNet is that a large number of layers can be trained easily without the increment of the percentage of error during the training period. In the Figure 2.5, the basic block of ResNet is depicted. The formulation of $F(\mathbf{x}) + \mathbf{x}$ is realized by the feed-forward network with skip or shortcut connections, where the data $\mathbf{x}$ from previous layer is again added after some operation $F(\cdot)$ (i.e.: ReLU) on it.

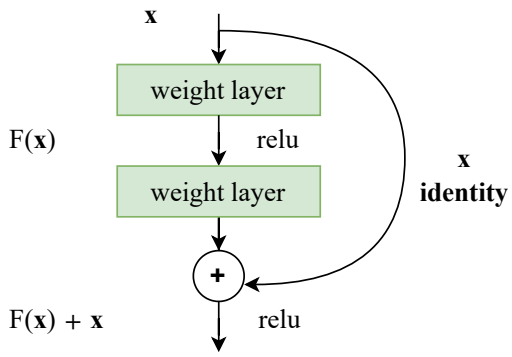


Figure 2.5: The building block for residual learning [He+15]

In our project work, SAD and SID tasks use blocks of convolutional layers to create a large network. The blocks used in SAD and SID are ResNet18 and ResNet34 which are covered in Chapter 4.

2.2.4 Recurrent Neural Network

Recurrent Neural Network (RNN) is an artificial NN where feedback connections are used in recurrent layers to exhibit the dynamic temporal behavior of data. These feedback loops help them maintain information over time. RNNs are employed to tackle the gradient vanishing problem. A particular type of RNN is called LSTM, which is described in the following section.

Long Short-Term Memory

The Long Short-Term Memory (LSTM) is a type of artificial RNN architecture that is commonly used in the field of NN. Along with standard units of RNN, it uses a particular memory cell that can store information for a more extended period. Since speech is continuous, and segmentation is also considered in our task, LSTM can be used for speech processing tasks.

2.3 Clustering Algorithms

Feature clustering algorithms are commonly used for the classification of extracted features from audio data. DBSCAN and Mean Shift are the two most commonly used algorithms and are investigated in our task.

2.3.1 DBSCAN

This algorithm requires two parameters to cluster features. The first parameter is the radius of a circle: ϵ, and the feature data points that is bounded by the circle is called ϵ-neighborhood of that center point. The other one is the *minPts* that captures a minimum number of data points bounded by that circle. Other points, which are not a part of any circles, are treated as noise. This density-based algorithm is suitable for an arbitrarily shaped dataset for feature classification [Est+96].

2.3.2 Mean Shift

Mean Shift is another density-based algorithm for feature clustering [CM02]. It requires only one parameter: ϵ, which is again, the radius of a circle. Based on the accumulated feature data points bounded by that circle, it estimates the gradient of the density function and tries to place centroids of the clusters at the maxima of that density function. Thereby all the data points in ϵ-neighborhood will climb up to a certain altitude defined by the local maxima of that cluster. This algorithm can identify the noise as well.

2.4 Similarity measurements

A similarity measure is a measure of how similar two data points are. Higher similarity implies smaller distance between two features, and the distance is larger for a lower degree of similarity. Similarity measurement techniques are used for DNN to check the data created with data augmentation methods. In general, similarities are measured in the range 0 to 1 [0,1]. This score is called the similarity score in the machine learning world.

2.4.1 Euclidean distance

The most common measure for distance is Euclidean distance. It is measured in Euclidean space and is also the best proximity measure for the length of two connecting points. The Pythagorean theorem can measure the length of the path connecting two points. It can be defined as:

$$d_i = \sqrt{\sum_{i=1}^{k} (x_i - y_i)^2} \tag{2.14}$$

where d_i is the distance between two data points x_i and y_i.

2.4.2 Manhattan distance

Manhattan distance measures the absolute difference between two data points in Cartesian coordinates and is defined as:

$$d_i = \sum_{i=1}^{k} |x_i - y_i| \tag{2.15}$$

2.4.3 Cosine Similarity

This metric finds the normalized dot products between two data points. Since it measures the angle between two vectors, the angle 0 degree implies similarity of 1, and the angle 90 degree implies similarity of 0. It is computed as:

$$\text{similarity}_{cosine}(A, B) = \cos(\theta) = \frac{A \cdot B}{||A||||B||} \tag{2.16}$$

where $A \cdot B$ is the dot product between A and B and the cosine angle measures the cosine similarity between these two points.

2.4.4 Jaccard Similarity

It is the ration of the set intersection cardinality over set union cardinality and is defined as:

$$\text{similarity}_{Jaccard}(A, B) = \frac{|A \cap B|}{|A \cup B|} \tag{2.17}$$

where the ratio implies Jaccard similarity coefficient that measures similarity between two finite sample sets.

2.5 Speaker Diarization Components

This section provides a brief overview of the components and the parameters involved in the task of the speaker diarization system.

2.5.1 Annotations

The temporal localization of speech corresponding to a speaker identity is referred to as annotation in the field of speaker diarization [Bre+20].

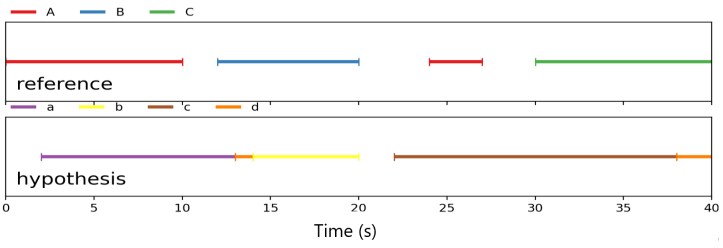


Figure 2.6: Reference and Hypothesis Annotations [Bre17]

In Figure 2.6, we can visualize the reference and hypothesis annotations. The reference annotations contain the ground-truth temporal speech activities and the corresponding speaker labels. The hypothesis annotations are computed as a result of the outputs of the SAD system, the temporal speech intervals, and SID system, the corresponding speaker predictions. Both the annotations together are used in the evaluation of the diarization system.

2.5.2 Hungarian Algorithm

The Hungarian algorithm is employed to establish a one-to-one mapping between the hypothesis outputs and the reference annotations. For the Speaker Diarization (SD) task, this algorithm is implicitly used to establish the mapping between the annotations. The goal of the algorithm is to solve the fundamental assignment problem to determine the optimum assignment. Further details about the algorithm can be found in the literature [Kuh55].

2.6 End-to-End Speech Recognition

Automatic Speech Recognition (ASR) task is in general a sequence-to-sequence problem where the system has to generate a sequence of output transcription S from an input sequence of acoustic features X extracted from speech, $X = (x_1, x_2, ...x_N)$ where $x_i \in \mathbb{R}^d$ is the feature vector at time t. The recognising process maximizes the likelihood function $P(s_i \mid X)$, s_i is the i^{th} most likely word sequence of sentence S, given the speech feature sequence X.

$$S = \underset{S}{\operatorname{argmax}}\, P(s_i \mid X) \tag{2.18}$$

$$P(s_i \mid X) = \frac{P(X \mid s_i)P(s_i)}{P(X)} \tag{2.19}$$

Here $P(s_i)$ is the prior probability for the word, which is generated by training the system under a large amount of labeled data by the models for matching. The job of ASR system is to solve the equation 2.18 [WWL19].

The traditional speech recognition systems use complex components such as Hidden Markov Models, Gaussian Mixture Models, Deep Neural Networks, n-gram, neural network-based language models, and complicated encoding-decoding algorithms. However, recent end-to-end speech recognition replaces all these different components in the traditional pipeline with a single end-to-end deep recurrent neural network [WWL19]. This is because the end-to-end approach tries to guess each letter on a given audio and combine those to form words and then the sentences with the help of effective utilization of the attention mechanism. On the other hand, the traditional approach, where vocabulary and n-gram are used, utilizes the information of words.

The recently developed end-to-end ASR system utilizes hybrid CTC-Attention architecture [Wat+17] which effectively inherits the advantages of both CTC [Zia+14] and attention-based architectures [Cho+15] in training and decoding. While the model is being trained, it utilizes the multi objective learning framework by combining both CTC, $\mathcal{L}^{\text{ctc}}$ and attention-based, $\mathcal{L}^{\text{att}}$ cross entropy losses which helps a model to improve its robustness and achieve fast convergence. The learning objective function is given as [KHW17]:

$$\mathcal{L}^{\text{total}} = \alpha\mathcal{L}^{\text{ctc}} + (1 - \alpha)\mathcal{L}^{\text{att}} \tag{2.20}$$

where α serves as a Multi-objective training parameter to linearly interpolate both objective functions, it varies between 0 and 1.

2.6.1 Transformer Model

This project work makes use of the Transformer architecture [Vas+17] that uses attention to boost the speed that these models can be trained. Here the conversion of input sequences into output sequences entirely relies on self-attention.

In LSTM, the sentence is processed sequentially, and information is retained through past hidden states with the assumption that each state is assumed to be dependent only on the previously seen state (Markov property). Training the LSTMs need to propagate the error back in time through words, one word at a time, which limits the model for parallel training, and also chances of losing earlier information is higher after a few time steps.

Model Architecture

Transformers overcome the problem associated with the architectures as mentioned above because the attention mechanism avoids the recursion by processing sentences as a whole and learning the relationships between words with the help of multi-head attention and positional embedding layers. Figure 2.8 shows the graphical representation of the Transformer and its components. The below-mentioned mathematical expressions based on Transformer are taken from [Vas+17].

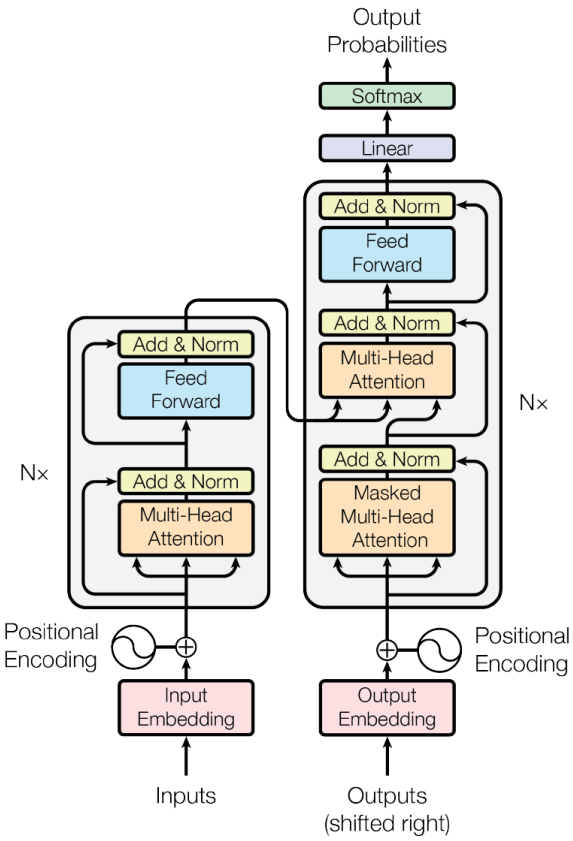


Figure 2.7: The Transformer-model architecture [Vas+17]

Input Embedding and Positional Encoding

Given an audio signal as an input to the model, the first step involved in the ASR process is input tokenization[WK92]. The process of collecting distinct elements, where the arrangement of the elements in the set is irrelevant, is called tokenization. For example:

Text	Tokenization
"Hello, I am Tom"	"Hello", "I", "am", "Tom" "Tom", "Hello","I","am" "I","am","Hello","Tom"

Since the order is irrelevant, the input set of sequence is denoted as $X = (x_1, x_2, ..., x_N)$ where $X \in \mathbb{R}^{N \times d_{in}}$. The elements of the sequence x_i are referred to as tokens and d_{in} is the input dimension.

The semantics of the inputs are captured by an embedding that places the semantically similar inputs grouping close to each other in a vector space. Before feeding the input $\mathbf{x}$ to the encoder block, each symbol (word, character, sentence) is mapped onto a vector with the continuous value represented by the input embedding unit, which are the learnable parameters for the NN. These vectors help the learning algorithms to achieve better performance in speech recognition tasks [Mik+13].

Since the model does not contain any recurrence module such as Recurrent Neural Network (RNN) or Long Short-Term Memory (LSTM), the positional encoding unit injects the information about the relative or absolute position of the tokens in the sequence by adding positional encoding into the input embedding.

The dimension of positional encoding vectors is d_{model}. It is calculated by using the sine and cosine functions of different frequencies as follows [Vas+17]:

$$\mathrm{PE}_{(pos,2i)} = \sin\left(pos/10000^{2i/d_{model}}\right) \tag{2.21}$$

$$\mathrm{PE}_{(pos,2i+1)} = \cos\left(pos/10000^{2i/d_{model}}\right) \tag{2.22}$$

here pos and i represent the position in sequence and dimension, respectively.

Encoder and Decoder

The encoder is on the left, and the decoder is on the right side of Figure 2.8, which also consists of multi-head attention and position-wise fully connected feed-forward layers. The encoder transforms a speech feature sequence $(x_1, ..., x_T)$ to a hidden representation $(h_1, ..., h_L)$. Given h_i, the decoder then generates an output sequence $(y_1, ..., y_S)$ one character at a time. The decoder takes the previously emitted characters as additional inputs when generating the next character at each time step.

The encoder and decoder units are multiple identical encoders and decoders stacked on top of each other. The number of encoder and decoder units is a hyper-parameter. The encoder has two sub-units: The first is multi-head self-attention, and the second is a position-wise fully connected feed-forward network as shown in the figure 2.8. On the other hand, the

decoder has an additional sub-unit of multi-head attention over the output of the encoder stack, as shown in the Figure above.

The encoder and decoder units are multiple identical encoders and decoders stacked on top of each other. Both the encoder and the decoder stacks consist of the same number of attention units.

The encoder has two sub-units: The first is multi-head self-attention, and the second is position-wise fully connected feed-forward network as shown in the Figure. On the other hand, the decoder has an additional sub-unit of multi-head attention over the output of the encoder stack.

Attention

Attention works by focusing on the parts of the speech which are relevant instead of the entire sentence. In ASR, the two types of attention used are self-attention and cross-attention [CDL16]. Self-attention allows inputs to interact with each other and determines the priority of the attention, while cross-attention is termed the attention between encoder and decoder. Despite both attentions being similar, cross-attention uses attention information from the encoder and the calculated attention scores from the decoder.

Self-attention relates different positions of input sequences to compute representations for the inputs. The attention layer computes three vectors based on the input, termed as key(K), query(Q), and value(V). The dot product between K and Q is a scalar that serves as the relative weighting for a given position. This process is applied in parallel at every element in the input sequence, so that each element has an attention score as well. A softmax function is applied to those attention scores to ensure that the total weighting sums to 1.0, and then multiplied with the corresponding value vector.

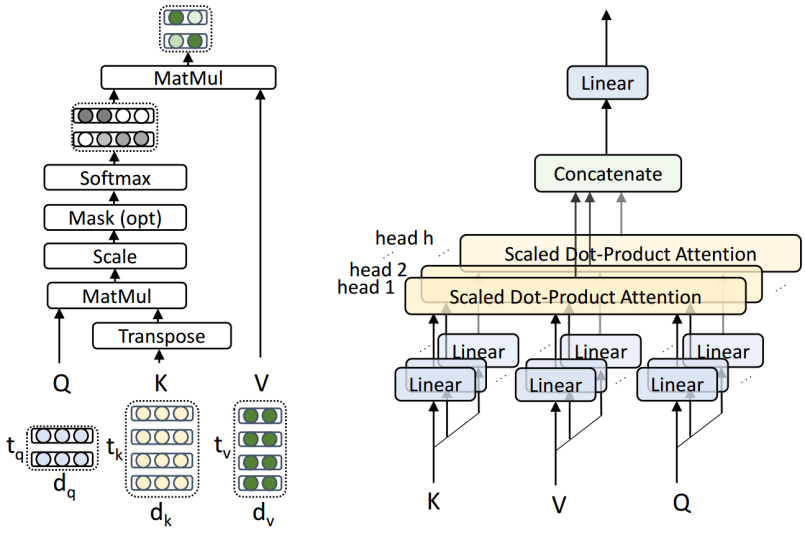


Figure 2.8: Multi Head Attention [DXX18]

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}}) \cdot V \tag{2.23}$$

Here d_k represents the corresponding element dimensions.The resulting new sequence of vector forms an internal representation of the input sequence, which will then be passed to a feed-forward fully connected layer. Since Transformer uses multiple number of such self-attention block in parallel and is called the multi-headed attention.Referring the paper [DXX18],

$$\text{Multi-Head}(Q, K, V) = (head_1, ..., head_h) \cdot W^O \tag{2.24}$$

$$head_i = \text{Attention}(Q\,W_i^Q, K\,W_i^K, V\,W_i^V) \tag{2.25}$$

Where the projections are parameter matrices $W_i^Q \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{model} \times d_v}$ and $W^O \in \mathbb{R}^{d_v \times d_{model}}$, $d_q = d_k = d_v = \frac{d_{model}}{h}$, Where d_q, d_k and d_v are dimension of query(Q), key(K) and value(V) respectively. The other sub-unit, Position-wise fully connected feed-forward networks, consists of two linear transformations with a ReLU activation in between.

$$\text{FFN}(x) = \max(0, x\,W_1 + b_1)\,W_2 + b_2 \tag{2.26}$$

Where the weights $W_1 \in \mathbb{R}^{d_{model} \times d_{ff}}$, $W_2 \in \mathbb{R}^{d_{ff} \times d_{model}}$ and biases $b_1 \in \mathbb{R}^{d_{ff}}$, $b_2 \in \mathbb{R}^{d_{model}}$. The linear transformations are carried out in similar way across different positions. To ensure the effective training process, each encoder and decoder unit are connected with layer normalization along with residual connection. The final output of the decoder is then sent to a linear layer, with Softmax, where the output character scores are determined.

2.7 Evaluation Metrics

This section deals with several evaluation metrics required to assess the behavior of the various implemented architectures for the SAD and SID tasks.

Precision, Recall, and F1-Score

The model output, when assigned to one of the classes, can also result in misclassifications. Here, in our case, we consider the confusion matrix shown in the figure 2.9. The rows correspond to the targets/ the actual classification for the different classes, with the corresponding columns being the predicted classifications. The main diagonal elements of the matrix represent the correctly classified elements.

From the confusion matrices in Figure 2.9, we can determine True Positives (TP), True Negatives (TN), False Positives (FP) and False Negatives (FN) where the TP are the elements labeled as positive by the model which are positive while FP are the elements labeled as positive by the model but are negative. Using these determined elements, we can compute metrics such as Precision, Recall, and F1-score [GBV20].

ACTUAL	Classes	PREDICTED Positive (1)	PREDICTED Negative (0)	Total
	Positive (1)	TP = 20	FN = 5	25
	Negative (0)	FP = 10	TN = 15	25
	Total	30	20	**50**

ACTUAL classification	Classes	PREDICTED classification a	b	c	d	Total
	a	6	0	1	2	9
	b	3	9	1	1	14
	c	1	0	10	2	13
	d	1	2	1	12	16
	Total	11	11	13	17	**52**

Figure 2.9: Examples for Two-Class and Multi-Class Confusion Matrices [GBV20]

The Precision is the measure of True Positives divided by all the elements classified as positives. This results in a measure for the model's reliability when it predicts a positive.:

$$\text{Precision} = \frac{\text{TP}}{\text{TP+FP}} \tag{2.27}$$

The Recall is the measure of True Positives divided by all the elements classified positively. This results in a measure for the model's ability to find all the positive units:

$$\text{Recall} = \frac{\text{TP}}{\text{TP+FN}} \tag{2.28}$$

The F1-Score is a weighted average between Precision and Recall. This measure gives an insight into how well the two metrics are balanced by linking both measures:

$$\text{F1-Score} = 2 \cdot \left(\frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \right) \tag{2.29}$$

However, in the multi-class classification case in the SID task having k classes, we need to extend further from the binary classification as required for the SAD task and modify the measures for Precision and Recall accordingly: For every k-th class, the measures are re-calculated,

$$\text{Precision}_k = \frac{\text{TP}_k}{\text{TP}_k + \text{FP}_k} \tag{2.30}$$

$$\text{Recall}_k = \frac{\text{TP}_k}{\text{TP}_k + \text{FN}_k} \tag{2.31}$$

Using the obtained measures for the k classes, for the multi-class case, we can compute the arithmetic mean of the metrics to finally arrive at the Macro Average Precision and Recalls:

$$\text{MacroAveragePrecision} = \frac{1}{K} \cdot \sum_{k=1}^{K} \text{Precision}_k \tag{2.32}$$

$$\text{MacroAverageRecall} = \frac{1}{K} \cdot \sum_{k=1}^{K} \text{Recall}_k \tag{2.33}$$

Finally, the Macro F1-Score is just the harmonic mean of the above calculated metrics:

$$\text{Macro F1-Score} = 2 \cdot \left(\frac{\text{MacroAveragePrecision} \cdot \text{MacroAverageRecall}}{\text{MacroAveragePrecision}^{-1} + \text{MacroAverageRecall}^{-1}} \right) \tag{2.34}$$

Receiver Operating Characteristic (ROC) Curve

The Receiver Operating Characteristic (ROC) curve illustrates the diagnostic ability of a binary classifier system. It plots the values for True Positive Rate (TPR) against False Positive Rate (FPR) for different decision threshold settings. The area under the ROC curve is known as AUC, which is an operating-point-independent performance measure. The higher the AUC, the better the ability of the classifier to distinguish the positive and negative classes.

Detection Cost Function

The Detection Cost Function (DCF) for the SAD task is performed for evaluation [Bye19]. It adds up the False Negative Rate (FNR) and False Positive Rate (FPR) by weighting them with 0.75 and 0.25 respectively with:

$$\mathrm{FNR} = \frac{\mathrm{FN}}{\mathrm{TP} + \mathrm{FN}} \tag{2.35}$$

$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{TN} + \mathrm{FP}} \tag{2.36}$$

$$\mathrm{DCF} = 0.75 \times \mathrm{FNR} + 0.25 \times \mathrm{FPR} \tag{2.37}$$

Top-5 Accuracy

The evaluation of the SID task and the FS-02 challenge also involves a metric known as Top-5 Accuracy. It indicates the confidence that an expected answer matches one of our model's top 5 highest probability predictions. This metric considers a classification to be correct if any of the five predictions match the target label. This metric is also different from the conventional accuracy where the model prediction must be exactly the expected answer [Dan21].

Diarization Error Rate

The standard metric for the evaluation and comparison of the speaker diarization systems is called the Diarization Error Rate (DER). This metric also takes the overlapping speech into consideration and it is calculated as:

$$\mathrm{DER} = \frac{\text{false alarm} + \text{missed detection} + \text{confusion}}{\text{total duration}} \tag{2.38}$$

Here in Equation 2.38, the several components involved in the computation of the DER are shown. The false alarm denotes the duration of non-speech misclassified as speech, the missed detection is the duration of speech treated as non-speech, confusion is the duration of speech assigned to a wrong speaker and the total duration indicates the total over all the speakers' speech duration [Bre17].

Equal Error Rate

Equal Error Rate (EER) is a single number performance metric commonly used to measure and compare the overall accuracy level of different classification tasks. EER can be obtained by finding the interception point of two graphs, one for False Acceptance Rate (FAR) and the other for False Rejection Rate (FAR), as shown in Figure 2.10. The value of EER indicates that false identification and false rejection are minimal. Thus, the lower the FAR and the FAR values, the lower the EER value, which in turn indicates a better accuracy performance of a classification system [Yaa+20]. Our experiments use EER to evaluate the quality of speaker information vectors extracted from the SID model.

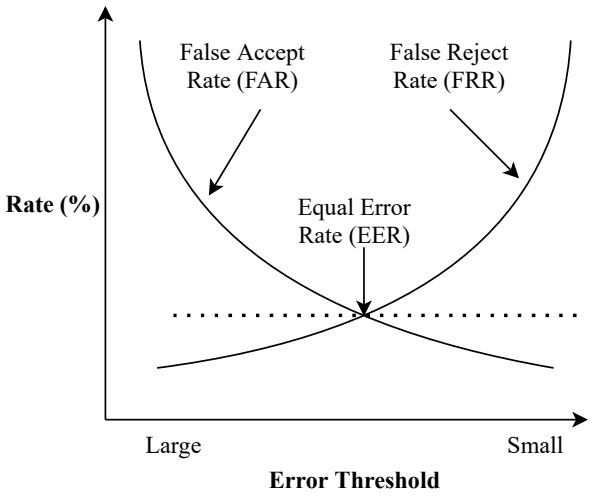


Figure 2.10: Equal Error Rate

Word Error Rate and Character Error Rate

While evaluating the performance of the ASR model, one analyzes how many symbols (words or characters) have been correctly recognized by the system for the given input acoustic signal. In our project, this is carried out by use of two score matrices: the Word Error Rate (WER) and the Character Error Rate (CER). They are computed using the Levenshtein distance [Zia+14]. It compares text produced by the model and the reference text for a given audio sample. It counts the number of insertions (I), substitutions (S), and deletions (D) of symbols (characters or words) needed to transform the given input audio signal into the text and normalizes the result.

Insertion is when a word is added in the hypothesis despite not being present in the acoustic feature. For example, the "cow eats grass" sequence is transcribed as "cow eats the grass."

Substitution is defined when an incorrect entry substitutes a word in the input sequence. For example, when "it is raining" is transcribed as "it is training."

Deletion occurs when a word is omitted from the transcribed hypothesis. For example, when "was a good day" is decoded as "was XXX good day"

The formula for computing WER is given below, where N is the total number of symbols in an actual transcription.

$$\mathrm{WER} = \frac{\mathrm{S} + \mathrm{I} + \mathrm{D}}{\mathrm{N}} \tag{2.39}$$

The aforementioned metrics are used for the evaluation of different NN models in the SAD, SID and ASR tasks. Further, they are also utilized in the Speaker Diarization (SD).

3 Database

The NASA Apollo Space Program was the first-ever effectively carried out space voyage. The program endured for over four a long time with six missions that completed, the Apollo 11 holds extraordinary importance from technical and operational perspectives [Han+18]. This particular mission was carried out for eight days. Two analog reel-to-reel recording machines were used to record all the communications between the workforce involved in the mission, guaranteeing progression without any loss of information. Each of the two machines comprises 30 channels, among which 29 channels on the tape were utilized to record the information, with one channel recording the Mission Elapsed Time (MET) in an encoded format.

The United States National Archives and Records Administration (NARA) carried out the digitization of both the tape-recorded data simultaneously and managed to preserve the synchronicity of the individual channels along with the respective MET channel. This data was stored at 44.1 kHz and was later down-sampled to 8 kHz allowing a more acceptable frequency resolution for the speech analysis. Each of these digitized data was stored as half-hour chunks per channel. It also consists of other information such as the mission name, which recorder it belonged to, and start and end times as per the MET [Han+18].

From Figure 3.1, we can see a perspective of the eight different stages involved in the entire mission that can be classified. The kind of information available varies from stage to stage of the mission.

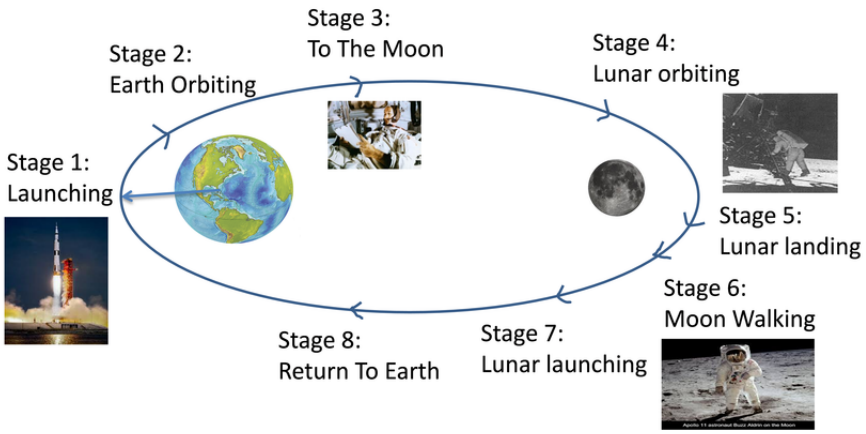


Figure 3.1: Overview of the timeline of Apollo 11 mission [Han+18]

From Figure 3.1, stages 1, 5, and 6 were classified to be high-impact mission-critical events leading to the development of the 100-hour Challenge Corpus. The speech data had a quality varying between 0 and 20dB Signal-to-Noise Ratio (SNR) in this corpus. To have an equitable distribution of data into the training, test, and development sets for the challenge tasks, the

corpus data were further categorized based on noise levels, amount of speech and silence content as summarised in Table 3.1:

Table 3.1: Total duration of speech and silence (in hours) of the dataset, and average number of speakers per hourly segments [Han+18]

Data-set	Speech Duration (h)	Silence Duration (h)	Average no. of Speakers/hour
Train	25.9	31.3	35
Dev	3.5	6.6	29
Test	5.5	10.2	31

Challenge Tasks

As a part of the initial research direction, UTDallas-CRSS hosted challenge tasks which are designed to advance research in the fields of Speech Processing, Machine Learning and Natural Language Understanding:

1. Speaker Activity Detection (SAD)
2. Speaker Diarization
3. Speaker Identity Detection (SID)
4. Automatic Speech Recognition (ASR)

The first edition of this challenge made use of Fearless Steps - 01 (FS-01) which encouraged the development of unsupervised/semi-supervised speech and language systems. Moving forward to the next phase of the challenge Fearless Steps - 02 (FS-02) focuses on the development of supervised learning strategies. Along with the 80 hours of ground-truth information through training (Train) and development (Dev) sets, an extra 20 hours of blind-set assessment (Eval) information has been made available for this challenge. Additional tracks for streamlined speech recognition and speaker diarization are added in FS-02 [Jog+20]. The FS-02 Challenge Corpus is divided into:

1. Audio Streams: Audio streams consist of unmodified, digitized 8kHz audio recordings from the Apollo 11 mission. Due to the channels, tape aging, and other non-stationary noise sources, the audio quality is degraded. These audios are further divided into Train, Dev, and Eval datasets with the progressive order of degradation. The Train dataset has the most negligible degraded audio streams, followed by the Dev dataset, consisting of 125 and 30 minutes long audio streams. Since only the Train and Dev datasets have labeled data, they are used for supervised learning, carried out throughout this project.

Table 3.2: FS-02 audio stream data-sets

Data-set	Stream (min)	No. of examples
Train	30	125
Dev	30	30

2. Audio Segments: The audio segments are short-duration speech sections segmented from the audio streams. Each segment contains a continuous speech utterance from a single speaker. The SID task in the FS-01 challenge provided 183 speakers with a minimum of 10 seconds of training data. However, in FS-02, it is extended by adding over 30,000 additional utterances for 218 speakers. Table 3.3 illustrates the statistics of audio segments provided for the ASR task. While this task has the advantage of having fully diarized segments, the single word utterances shorter than 0.2s pose a challenge to the systems.

Table 3.3: Duration Statistics of audio segments for ASR [Jog+20]

Data-set	No. of Segments	Utterance Duration (s)		
		mean	min	max
Train	35,474	2.85	0.10	70.37
Dev	9,203	2.97	0.12	67.39
Test	13,714	2.78	0.10	53.04

Table 3.4: FS-02 audio segments for SID task

Data-set	No. of speakers	No. of Segments
Train	218	27336
Dev	218	6373

All the above-described datasets are used for the realization of SAD, SID, ASR tasks in this project work. These challenges aim at developing robust speech and language systems for naturalistic audio. FS-02 enabled the development of new supervised systems for core-speech tasks on Apollo data through its Challenge Corpus.

4 Architecture

This chapter deals with various network architectures along with their configurations employed in the implementation phase. From the SAD to ASR tasks, we discuss all the approaches investigated throughout the project work.

4.1 Speech Activity Detection

In this work, we implemented different CNN models to perform the SAD task. To enhance the detection capability of speech activity in sequential audio data, a combined architecture of the residual network and Long Short-Term Memory (LSTM) called ResNet-LSTM [LLL20] is used.

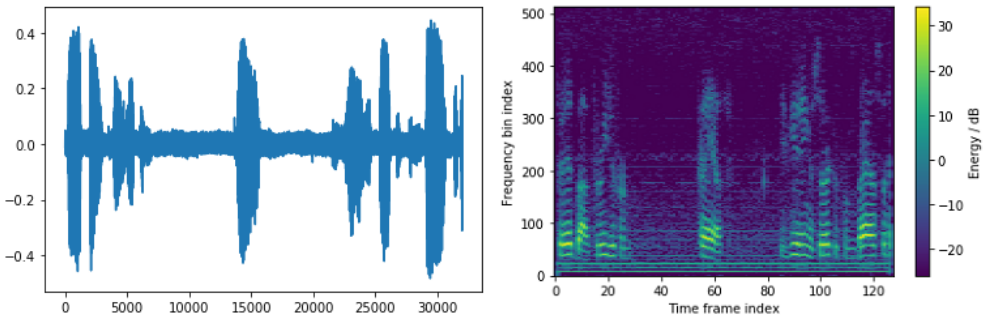
4.1.1 System

The MFCC feature extraction method is approached in our project as described in Section 2.1. A Mel-spectrogram is a visual representation of a frequency spectrum as it changes over time. The frequency dimension represents the 13 *Cepstral co-efficients* of the MFCC features and the time dimension represents the time frame index determined by STFT shift of 10 ms carried out for the segmented audio with the window length of 25 ms.

We followed two different data preparation methods for segmenting the audio to estimate speech activities, individual frames of an audio segment, or the entire segment. For all the frames of an audio segment, a frame-wise estimation is made with the corresponding targets, whereas for an entire audio segment, we obtained the segment-wise estimation, which is considered a target. A mini-batch of extracted MFCC features from the audio segments are fed as input to NN, and losses are calculated for the mini-batch.

4 s segmentation

In the first approach, we chunked the long audio data into 4 s segments. A 4 s chunked audio of 32000 samples and corresponding Mel-Spectrogram is provided in Figure 4.1 to make it more vivid.



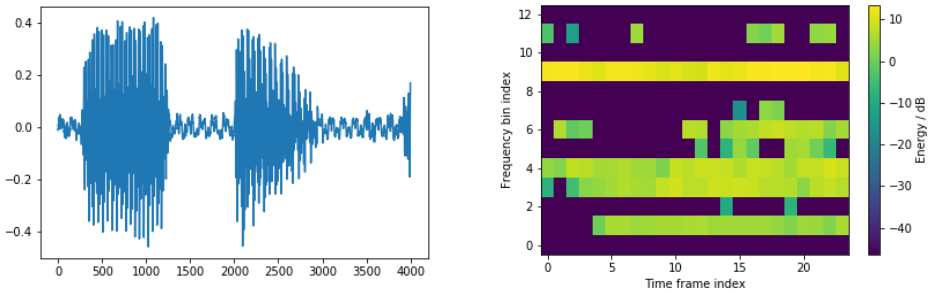
(a) Audio data (b) Mel-spectrogram

Figure 4.1: 4 s audio segment from the first audio of train dataset and corresponding Mel-spectrogram

The MFCC features are calculated over 4 s segments leading to 199 frames in time and 13 features in the frequency dimensions, respectively. The NN is trained later using these features to detect the speech activity on each frame of a 4 s segment.

0.5 s segmentation

In second approach, we segmented the audio data into 0.5 s chunks. For each audio segment, we calculated the MFCC features.



(a) Audio data of 0.5 s (b) Mel-Spectrogram of 0.5 s

Figure 4.2: 0.5 s audio segment and the corresponding Mel-spectrogram

In Figure 4.2b, the time dimension represents 23 frames, and the frequency dimension represents 13 features, respectively. The peaks in the audio data can either be the actual speech or noise, and the color in the spectrogram represents the energy levels of the signal. The NN is trained with the different energy levels of MFCC features to distinguish speech and non-speech activity for the whole segment.

4.1.2 Architectures

In our project work, we implemented simple CNN and LSTM-based ResNet18 model. The following Section describes the architecture behind these two systems.

Convolutional Neural Network-based SAD

The main task of the SAD was to implement the reference paper [LLL20] which used LSTM-based ResNet18 model. Before we implemented the reference paper [LLL20], we attempted a simple CNN model to understand the motive of the ResNet model. The CNN created in such a way that the output of the last layer contains 128 channels, which is also the same for the last layer of the ResNet18 models.

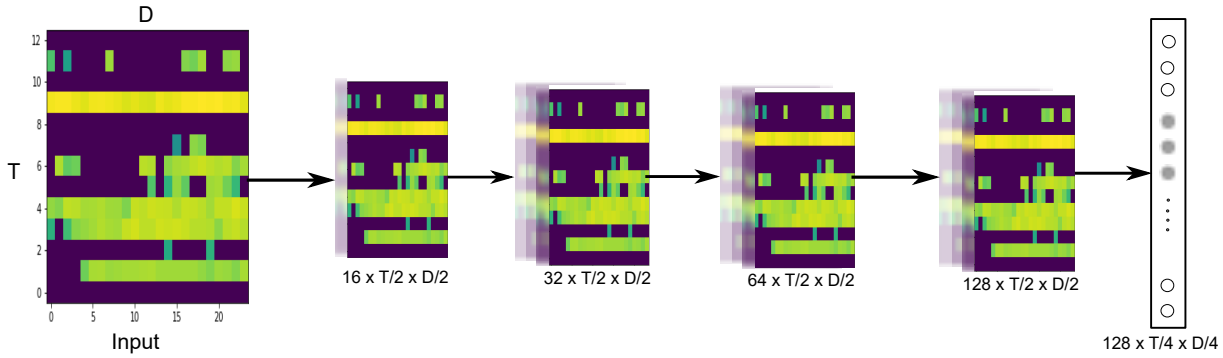


Figure 4.3: Simple CNN model

Figure 4.3 shows the simple CNN model where T denotes the time frames and D denotes the MFCC features. The batch of extracted MFCC features is the input to the model. Each block in Figure represents the increase in the number of channels. The linear layer with the Sigmoid activation function is used as the last layer.

Table 4.1: Simple CNN model

Layer	Parameters	Output Size
Input	-	$T \times D$
Simple Convolutional Network	conv 3×3, 4, /2	$4 \times \frac{T}{2} \times \frac{D}{2}$
	conv 3×3, 16	$16 \times \frac{T}{2} \times \frac{D}{2}$
	conv 3×3, 32	$32 \times \frac{T}{2} \times \frac{D}{2}$
	conv 3×3, 64	$64 \times \frac{T}{2} \times \frac{D}{2}$
	conv 3×3, 128 ,/2	$128 \times \frac{T}{4} \times \frac{D}{4}$
Linear	$128 \times \frac{T}{4} \times \frac{D}{4}$, with Sigmoid	1

Table 4.1 describes each layer's number of channels and output size. The inputs are fed into the network, and then it convolves with a kernel filter of size 3×3. The output channel is increased to 4 and is passed through the ReLU activation function.

The max-pooling operation of kernel size 2×2 is performed at the end of the first output layer. The dimension of the inputs is reduced by half at the output layer. The following

layer channel widths are increased as $16, 32, 64, 128$ with kernel size of 3×3. The dimension is reduced once again at the end of the last convolutional layer. The fully connected linear layer with the Sigmoid function predicts the speech or non-speech activity.

ResNet-LSTM based SAD

In our work, we implemented an LSTM based ResNet18 to get the advantage of the combination of ResNet and LSTM network and it is depicted in Figure 4.4. It consist of three main components: a ResNet front-end, a one-dimensional (1-D) mean statistics pooling layer over feature axis and an LSTM back-end which includes two bidirectional LSTM layers followed by a linear layer with the Sigmoid non-linearity. The configuration of ResNet-LSTM is given in Table 4.2.

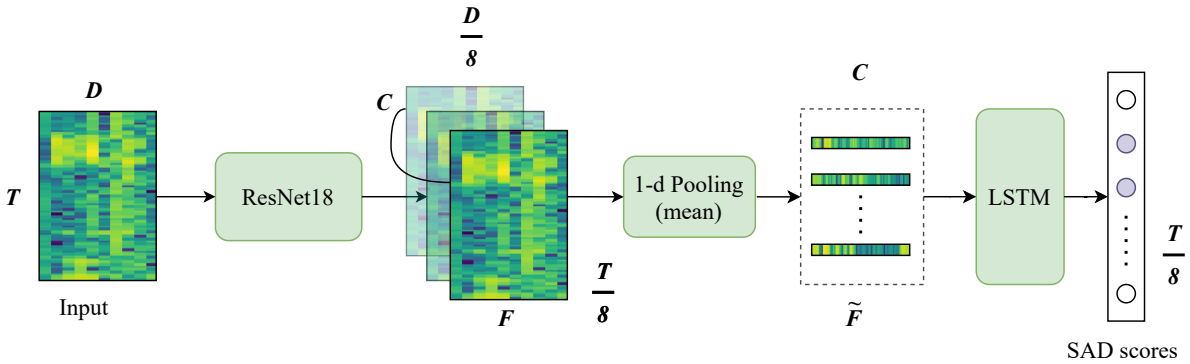


Figure 4.4: The complete structure of ResNet-LSTM based SAD system

Table 4.2: Model Parameters and output size of ResNet-LSTM [LLL20]

Layer	Parameters	Output Size
Input	-	$T \times D$
ResNet	conv 3×3, 16	$16 \times T \times D$
	$\begin{bmatrix}\text{conv } 3 \times 3, 16\\ \text{conv } 3 \times 3, 16\end{bmatrix} \times 2$	$16 \times T \times D$
	$\begin{bmatrix}\text{conv } 3 \times 3, 32\\ \text{conv } 3 \times 3, 32\end{bmatrix} \times 2\ ,\ /2$	$32 \times \frac{T}{2} \times \frac{D}{2}$
	$\begin{bmatrix}\text{conv } 3 \times 3, 64\\ \text{conv } 3 \times 3, 64\end{bmatrix} \times 2\ ,\ /2$	$64 \times \frac{T}{4} \times \frac{D}{4}$
	$\begin{bmatrix}\text{conv } 3 \times 3, 128\\ \text{conv } 3 \times 3, 128\end{bmatrix} \times 2\ ,\ /2$	$128 \times \frac{T}{8} \times \frac{D}{8}$
1-D Pooling	*mean*	$128 \times \frac{T}{8}$
transpose	-	$\frac{T}{8} \times 128$
Bi-LSTM	64 units per direction, 2 layers, drop=0.5	$\frac{T}{8} \times 128$
Linear	128×1, with Sigmoid	$\frac{T}{8} \times 1$

With the input features of $T \times D$, the ResNet performs a high-level feature mapping $\boldsymbol{F} \in \mathbb{R}^{C \times \frac{T}{8} \times \frac{D}{8}}$. Here T and D denote the number of frames along the time axis and the number of feature dimensions along the frequency axis, respectively, and the number of CNN channels is denoted by C. The mean statistics are captured by 1-d pooling over the frequency axis that leads to a feature mapping of $\tilde{\boldsymbol{F}} \in \mathbb{R}^{C \times \frac{T}{8}}$. The features relative to the task are extracted before feeding it to the LSTM back-end layer. The network performs binary classification of speech activity in a frame and provides SAD scores in the range of [0,1] corresponding to the activity [LLL20].
Compared to our basic CNN model, this ResNet-LSTM is built on the residual set with channel widths of $16, 32, 64, 128$ blocks and a bi-directional LSTM before the fully connected layer at the end. So the front-end helps the network to tackle the gradient vanishing problem, and the back-end takes care of learning the continuous nature of time-series like sequential audio data [He+15].

4.2 Speaker Identity Detection

In the following Section, the implemented system and the employed Architectures in the Speaker Identity Detection (SID) task are explained in detail.

4.2.1 System

First, the characteristics of the input audio signal with the help of Mel-energy filterbanks are obtained in the form of features. As seen from the Chapter 2.1, in the data-preparation phase, an STFT is initially computed with a FFT of size 512 leading to the computation of 64-dimensional Mel-energy filter-banks which are the inputs to our NN models. Further experiments were carried out by modifying the input features. A logarithmic function was applied to the input Mel-energy filterbanks.

From Figures 4.5 and 4.6, we can observe how the input audio signal in the time domain and the resulting Mel-spectrogram looks. The *energy/ DB* scale on the right of Figures indicates the possible range of energy levels. From Figures, we can observe the speech activity in the form of peaks in the audio signal and corresponding regions with higher energy levels in the resulting Mel-spectrogram. There could also exist noise within the audio signal, which results in subtle energies correspondingly.

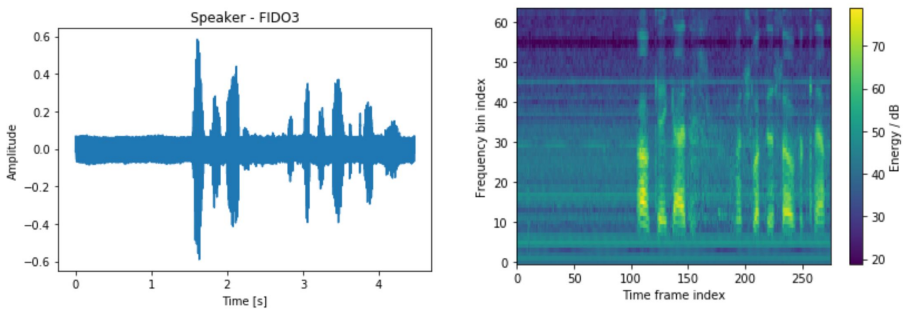


Figure 4.5: Example-1 for an audio signal and the resulting Mel-spectrogram for a speaker

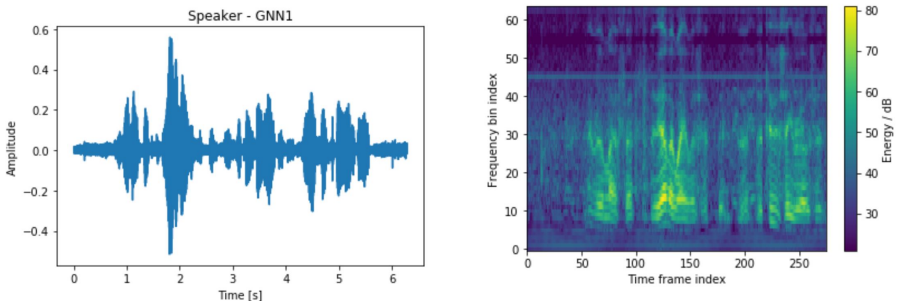


Figure 4.6: Example-2 for an audio signal and the resulting Mel-spectrogram for a speaker

The data is here in the form of segments, where every segment corresponds to a unique speaker. Every speaker from the FS-02 corpora is uniquely identified with a speaker identity. Every speaker identity is associated with a unique one-hot encoded binary vector serving as the targets for the training, validation, and evaluation of NNs in SID as explained in Chapter 2.2. The extracted features from the input audio segments, along with their corresponding binary vectors as targets, serve as the inputs to the NN models.

4.2.2 Architectures

The main goal of the SID task was to implement the deep ResNet vector from [LLL20]. However, before implementing the main architecture, several NN models were built from scratch, experimented, and the results are compared in this task. The comparison is discussed further in Section 5.2. This Subsection deals with various NN architectures worked within the SID task.

Simple Convolutional Networks

Once the data in the required format for the training of an NN is ready, the next step involved in the implementation of a NN to predict the correct speaker. For this purpose, a mini-batch of 10 segments is used for the training process. We discuss several simple convolutional networks in this Subsection.

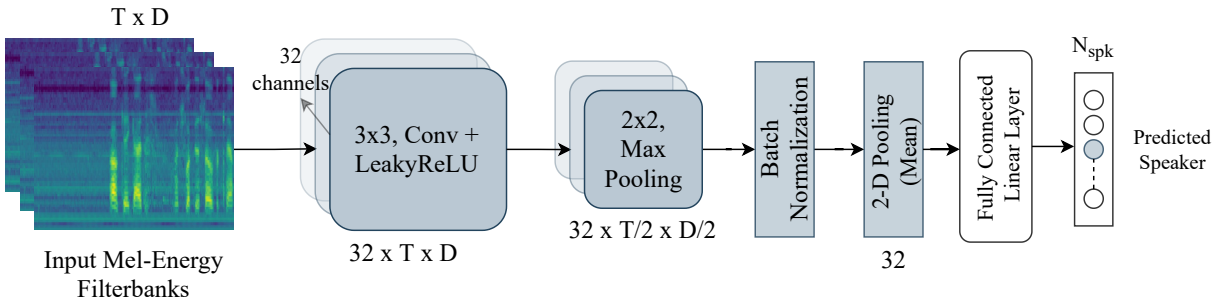


Figure 4.7: A Simple Convolutional Network - 1

First, we consider the network in Figure 4.7. The inputs having the dimensions $T \times D$ where T corresponds to the frames along the time axis, and D is the feature dimensions along the

frequency axis, are provided to a single layer of CNN consisting of 32 channels. A kernel of size 3×3 performs the convolution process with a stride of 1. At the end of the CNN layer, the dimensions are $32 \times T \times D$. At this stage, an activation function of Leaky ReLU, an extended variant of the ReLU as described in 2.2.1. The advantage of Leaky ReLU over the commonly used ReLU is that a slope is introduced to the left of $x = 0$, causing a *leak* and in return extending the range of the ReLU. This usage of modified ReLU ensures the gradient never falls below 0 and thus overcoming the problem of dying ReLU. The Leaky ReLU function is shown below [S19]:

$$g(x) = \max(0.01x, x), \tag{4.1}$$

where $g(\cdot)$ represents the activation function and x is the feature vector.

Further, with a kernel size of 2×2, a max pooling operation is performed which reduces the dimensions along the T and D by a factor of 2 leading to $32 \times \frac{T}{2} \times \frac{D}{2}$.

Additionally, a Batch-normalization is performed at this stage to accelerate the process of training an NN by stabilizing the input layer distributions. A normalization achieves this via mini-batch statistics, where each batch of features in the layer is normalized to have the mean of 0 and a variance of 1. For a mini-batch size of m, the mean, μ_j and the variance, Σ_j are re-computed as in Equations 4.2 and 4.3 and used in 4.4 to find the normalized result for a mini-batch of feature vectors [San+19]:

$$\mu_{x_j} = \frac{1}{m} \sum_{i=1}^{m} x_i \tag{4.2}$$

$$\Sigma_{x_j} = \frac{1}{m} \sum_{i=1}^{m} (x_i - \mu_{x_j})^2 \tag{4.3}$$

$$y_j = \frac{x - \mu_{x_j}}{\sqrt{\Sigma_{x_j} + \epsilon}}, \tag{4.4}$$

here, y_j is the output feature vector of the j-th batch after normalization.
Further, the Batch-normalized data is pooled over the joint dimension of time and frequency by calculating the mean statistics, generating the utterance level-representation.

Each unit in the N_{spk} corresponds to a registered speaker identity in the list of speakers in FS-02 SID dataset. The utterance level-representation is transformed to predictions for the N_{spk} with the help of mapping from a Fully Connected Linear layer followed by a Softmax function which transforms all the mappings to values between 0 and 1, so they can be interpreted as probabilities.

We also consider an extended version of Figure 4.7 with an additional CNN layer with 64 channels. The corresponding dimensions associated with this model are as shown in the Table 4.3 which summarises various layers and their respective dimensions for the above discussed Convolutional Networks.

Table 4.3: Summary of Simple Convolutional Networks

Model	Layer	Output size
Simple Convolutional Network - 1	$\left[\text{ conv } 3 \times 3,\ 32 \right], /2$	$32 \times \frac{T}{2} \times \frac{D}{2}$
	2-D Pooling	32
	Linear layer	$32 \times N_{spk}$
Simple Convolutional Network - 2	$\begin{bmatrix}\text{conv } 3 \times 3,\ 32 \\ \text{conv } 3 \times 3,\ 64\end{bmatrix}, /2$	$64 \times \frac{T}{2} \times \frac{D}{2}$
	2-D Pooling	64
	Linear layer	$64 \times N_{spk}$
Simple Convolutional Network - 3	$\begin{bmatrix}\text{conv } 3 \times 3,\ 32 \\ \text{conv } 3 \times 3,\ 64 \\ \text{conv } 3 \times 3,\ 128\end{bmatrix}, /2$	$128 \times \frac{T}{2} \times \frac{D}{2}$
	2-D Pooling	128
	Linear layer 1	128×256
	Linear layer 2	$256 \times N_{spk}$

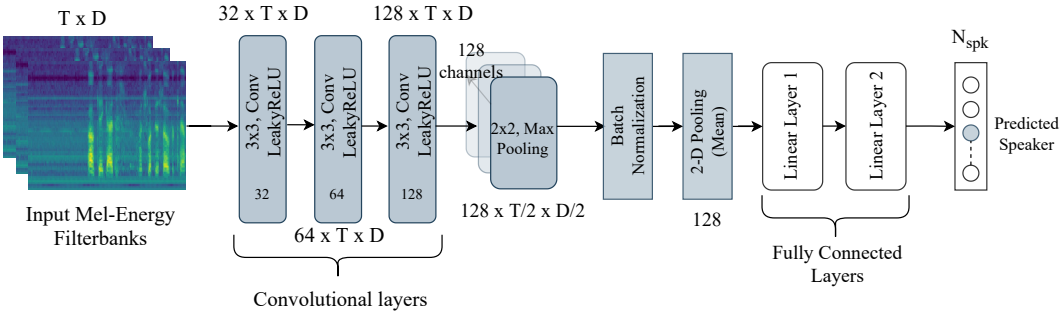


Figure 4.8: A Simple Convolutional Network - 3

Figure 4.8 is a relatively complex model compared to the other two models discussed in this Section. As seen in Chapter 5, the performance of all these models is evaluated in comparison to a more complex, robust model using ResNet34 architecture, as discussed next. The corresponding dimensions associated with this model are as shown in Figure 4.8.

Deep ResNet Vector-based SID

As seen from the description of the ResNet in Section 2.2.3, we employ those principles for making use of a ResNet34 architecture having the network as shown in Table 4.4.

The architecture discussed in this section has three main components: a ResNet front-end, a 2-D statistical pooling layer, and a feed-forward fully connected network [LLL20]. As seen for the earlier models, the input features of the shape $T \times D$ are sequentially provided at the ResNet34 front-end. This ResNet34 front-end has the channel widths set to $32, 64, 128, 256$. This ResNet front-end system transforms the sequential inputs to a high-level feature mapping $\boldsymbol{F} \in \mathbb{R}^{C \times \frac{T}{8} \times \frac{D}{8}}$ as in the case of the SAD task. Here C refers to the number of CNN channels.

Similar to the earlier seen models, the Leaky ReLU is applied at the end of every layer combined with Max pooling, ensuring the dimensionality reduction along both the T and D axis. Also, a 2-D pooling over the joint time and frequency axis ensure the generation of an utterance-level representation of C, which is the number of channels at the output of the CNN layers. Last, a feed-forward network consisting of two linear layers transforms the utterance to predictions for the registered speakers in the library, N_{spk}.

Figure 4.9 shows a representation of a Deep ResNet vector implementation which is very similar to the earlier discussed models except for more complex, robust, and larger channel widths for the CNN layers. Table 4.4 summarizes various parameters and the respective output sizes of the model.

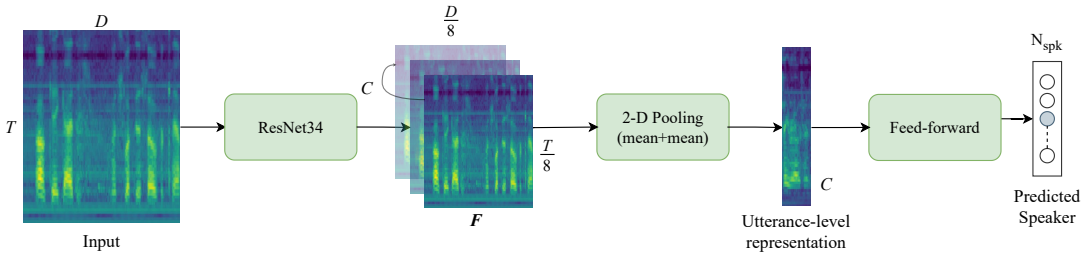


Figure 4.9: The structure of the Deep ResNet vector system [LLL20]

Table 4.4: Model Parameters and output size of Deep ResNet vector [LLL20]

Layer	Parameters	Output Size
Input	-	$T \times D$
ResNet34	conv 3×3, 32	$32 \times T \times D$
	$\begin{bmatrix}\text{conv } 3 \times 3, 32\\ \text{conv } 3 \times 3, 32\end{bmatrix} \times 3$	$32 \times T \times D$
	$\begin{bmatrix}\text{conv } 3 \times 3, 64\\ \text{conv } 3 \times 3, 64\end{bmatrix} \times 4\ ,\ /2$	$64 \times \frac{T}{2} \times \frac{D}{2}$
	$\begin{bmatrix}\text{conv } 3 \times 3, 128\\ \text{conv } 3 \times 3, 128\end{bmatrix} \times 6\ ,\ /2$	$128 \times \frac{T}{4} \times \frac{D}{4}$
	$\begin{bmatrix}\text{conv } 3 \times 3, 256\\ \text{conv } 3 \times 3, 256\end{bmatrix} \times 3\ ,\ /2$	$256 \times \frac{T}{8} \times \frac{D}{8}$
2-D Pooling	$mean, mean$	256
Linear 1	256×128, $dropout = 0.5$	128
Linear 2	$128 \times N_{spk}$	N_{spk}

4.3 Speaker Diarization

To improve the current SID results, we investigated another approach by obtaining the x-vectors from the pre-trained Deep ResNet vector of the SID system and employing the unsupervised form of clustering using DBSCAN, Means-shift algorithms. However, as discussed later in the evaluation chapter, the results were inconclusive, and the diarization could not be executed. So, we continued to improve the SID system, and the later Subsections here will discuss the executed diarization process in detail.

4.3.1 x-vector approach

In this experiment, by using clustering algorithms that use the extracted x-vectors from the SID system, we tried to execute the process of SD. Although any layer of a DNN can capture the characteristics of different targets, we extract the x-vectors from the affine layer of the DNN for embedding.

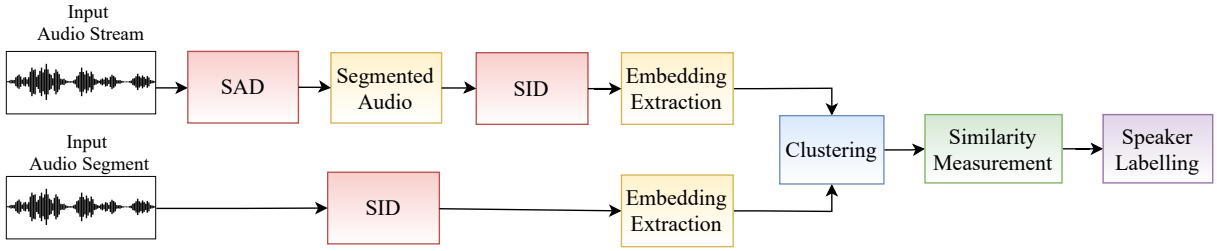


Figure 4.10: Workflow for proposed diarization system using x-vector approach.

The goal was to cluster the x-vectors for different speakers and thereby improve the existing results from the SID system in an unsupervised manner. After obtaining x-vectors, DBSCAN and Mean Shift clustering algorithms were tried to cluster the features captured by x-vectors. Since these algorithms apply to any shape of dataset, specifically non-spherical datasets, and the result is not sensitive to parameter selection, they are well suited for unsupervised clustering applications. Further, we measure the similarity between the clustered x-vectors using distance and similarity measurements. Euclidean distance, cosine similarity, Manhattan distance, and Jaccard similarity are the various measurements investigated in our task.

4.3.2 Temporal localization and Speaker identities

For the process of diarization, hypothesis annotations need to be computed. This Subsection provides a brief overview of how different components required for the task are computed. This process is carried out with *pyannote*, an open-source toolkit based on PyTorch machine learning framework [Bre+20].

Reference Annotations

As described in Subsection 2.5.1, the reference annotations are first obtained using the FS-02 development stream dataset. The mentioned dataset has information corresponding to the speech intervals and the corresponding speaker identities from the FS-02 audio streams. By

combining the two information, reference annotations are obtained, which serve as the ground truth in the process of SD.

Hypothesis Annotations

In order to obtain the hypothesis annotations, firstly a baseline system is established by using the ground-truth speech intervals and obtaining the corresponding speaker identities with the trained Deep ResNet vector from the SID system as shown in 4.9. The process is summarized in Figure 4.11:

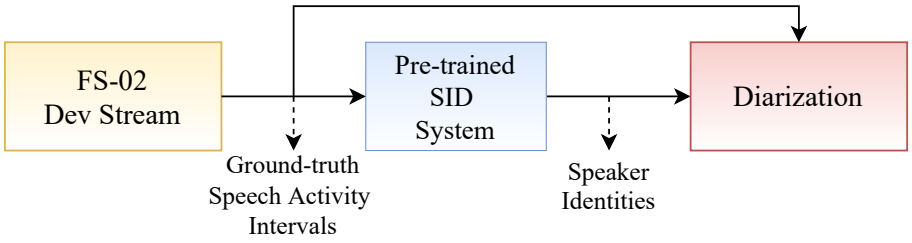


Figure 4.11: Baseline Diarization System Flow

Further, a diarization is carried out by making use of the same FS-02 stream audio through the SAD system and obtaining the speech activity intervals as outputs. These intervals are later passed through the trained Deep ResNet vector as explained in the previous approach. The corresponding identities are accumulated and diarized along with the available intervals from the SAD system. The process is shown in Figure 4.12:

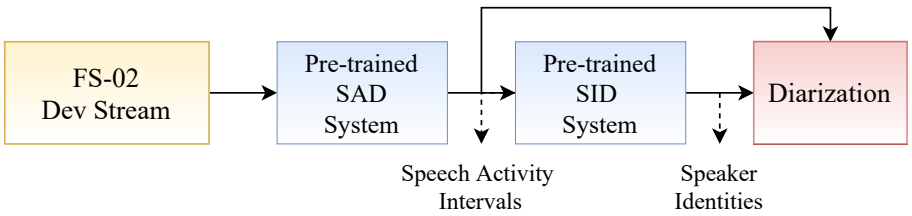


Figure 4.12: Proposed Diarization System Flow

The above mentioned architectures are evaluated based on Diarization Error Rate (DER) metric as described in Subsection 2.7 and the results are discussed in Section 5.3.

4.4 Automatic Speech Recognition

In this work, we used the Transformer Architecture and modified it to realize the implementations used in [SJU20; RBR19; Tan+16].

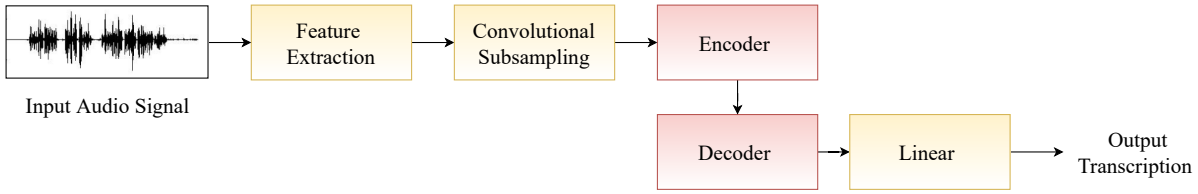


Figure 4.13: ASR System Flow

4.4.1 System

Figure 4.13 presents the workflow of the ASR system. The input speech signal is converted into an acoustic feature using MFCC bands. This speech signal is transformed into an 80-dimensional feature vector. The acoustic feature is passed to a convolution layer, ensuring standard dimensionality (256 in our case) before passing it to the encoder layers.

The transformer architecture implemented on ESPnet [Wat+18b] consists of a twelve-layered encoder and a six-layered decoder. The encoder is a combination of self-attention and feed-forward layers. These layers multiply the features with weight matrices and generate a query, key, and value sets. The self-attention then calculates the scalar dot product attention and propagates it to further layers. All the propagated resulting scores are concatenated, and the final attention score is calculated by multiplying it with a weight matrix. The decoder uses the key-value pairs from the encoder and computes its query vector to calculate the attention scores. The self-attention in the decoder slightly differs from the encoder by only using the known positions of the output sequence.

The final output of the decoder is then sent to a linear layer, with Softmax, where the output character scores are determined.

4.4.2 Speaker Adaption for ASR training

In our project work, the motive is to incorporate speaker adaptation in the ASR model. Speaker adaptation is a technique in which an ASR system is adapted to the acoustic characteristics of a particular speaker by using a small sample of utterances from that speaker. It bridges the gap between Speaker-independent (SI) and Speaker-dependent (S-D) models [HL93] by using speaker identities in a SI system. The focus is on minimizing the discrepancy between the evaluation data and the trained models, thus reducing the WER. The segment data used in our SID and ASR experiments have a unique speaker corresponding to each segment. An exact classification generated by the SID model provides one-hot encoded binary vector corresponding to a unique speaker. The generated one-hot vectors can be used as a speaker embeddings. However, one-hot encoded embedding vectors can be generated directly without the help of the SID model, as the FS-02 records for Train and Dev are well labelled. On the other hand, a pre-trained SID model facilitates the creation of speaker embedding x-vectors for Train and Dev segment data for ASR training, and validation purposes.

Speaker Embedding in the form of one-hot vector

A one-hot vector in our ASR experiments is a 1 × N vector used to distinguish between the individual speakers, where N is the number of speakers. In our case, the value of N is 167 since there are 167 common speakers active in both the development set and the training set. The vector consists of 0s in all cells except for a single 1 in one cell, used to identify the speaker uniquely.

Speaker Embedding in the form of x-vector

The x-vector of speaker embedding is extracted for each audio segment from the affine layer of the Deep ResNet vector model used in the SID task, as shown in Figure 4.14. The 100-dimensional x-vectors, in comparison to the one-hot vectors, contain linguistic and speaker identity features from the audio utterance. Hence, it is expected to capture all speaker-distinguishing information at the phonetic level.

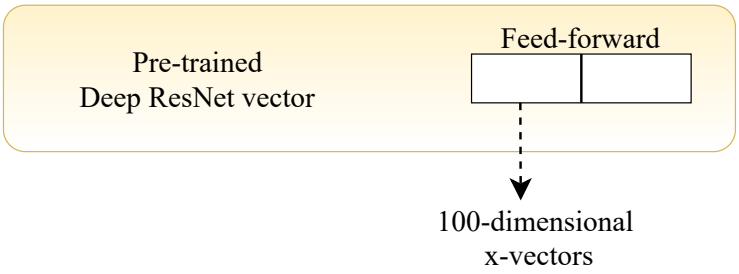


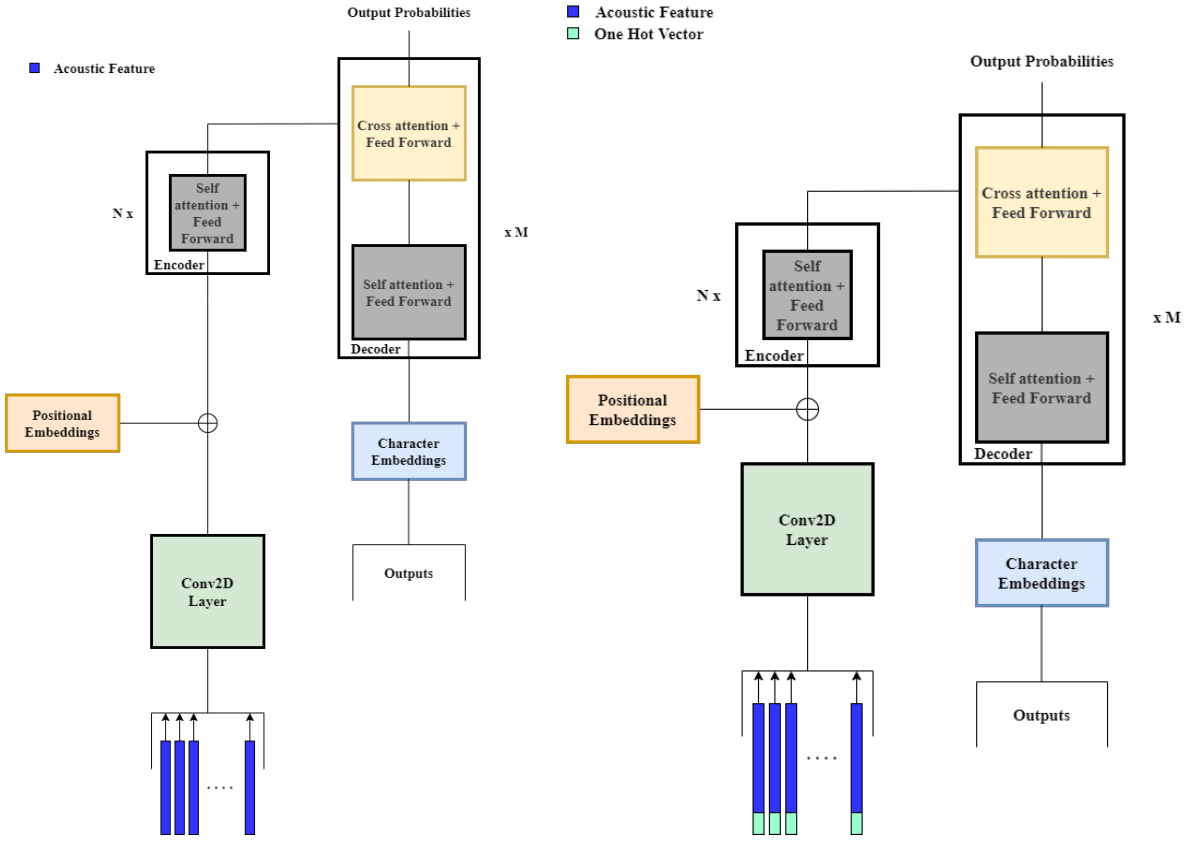
Figure 4.14: Extraction of x-vectors from pre-trained Deep ResNet vector

In the following Subsections, we describe how speaker adaptation is implemented using different techniques when training a Transformer model in the hope of improving its performance.

4.4.3 Speaker Information to Transformer Model

The *Base Model* in our experiment is the Transformer model trained without additional speaker information added to the model. The effect of providing speaker information is investigated by *Adding* and *Concatenating* the embedding vectors at different stages of model training as well as during testing, which are described below:

One-hot Speaker Vector Concatenation

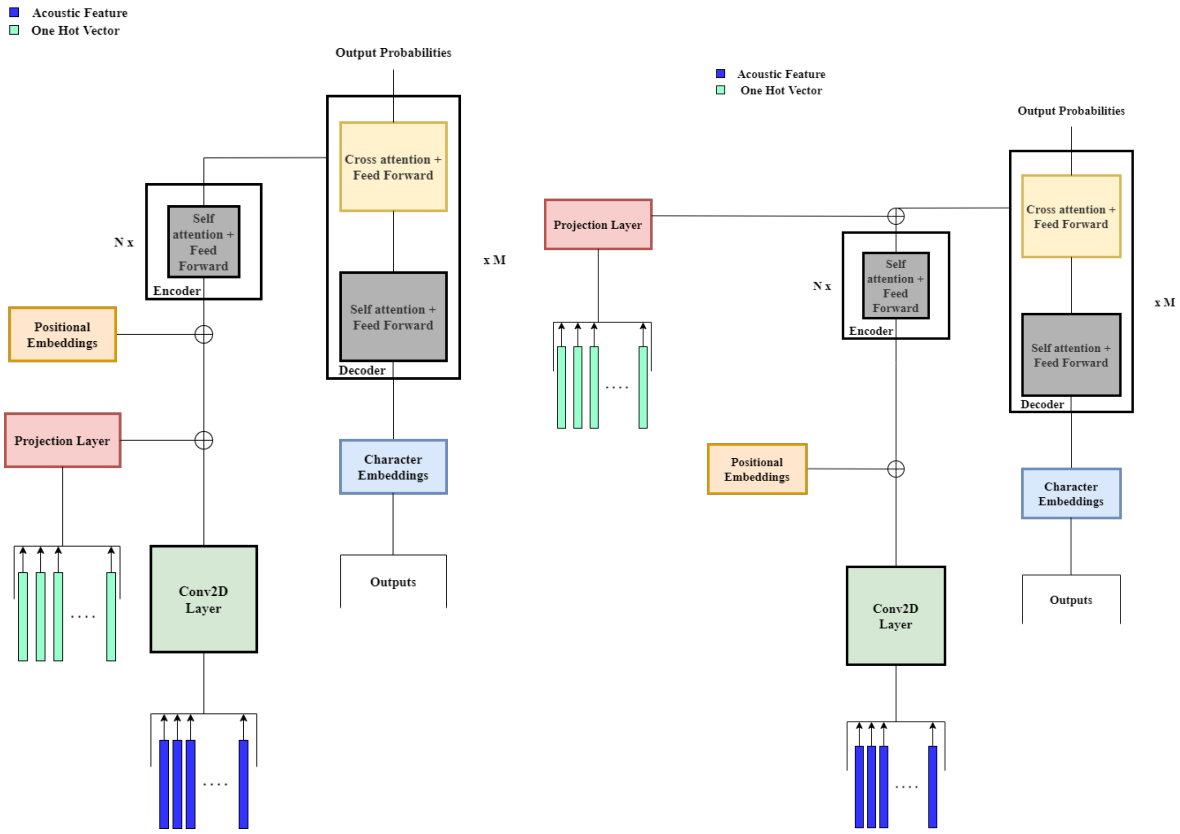


(a) Transformer Model Baseline (b) Concatenation of one-hot vectors to acoustic features

Figure 4.15: One-hot vector Concatenation to Transformer Base Model [SJU20]

Figure 4.15a is the Transformer baseline model. It is the original model provided by ESPnet. On the right-hand side, Figure, 4.15b depicts the technique that the one-hot vectors are concatenated with the acoustic features and sent to the acoustic model for training. The acoustic feature dimension is 80, and the dimension of the speaker vectors is 167, giving a combined 247-dimensional feature vector which is then input to *Conv2DLayer* of the Model.

One-hot Speaker Vector Addition



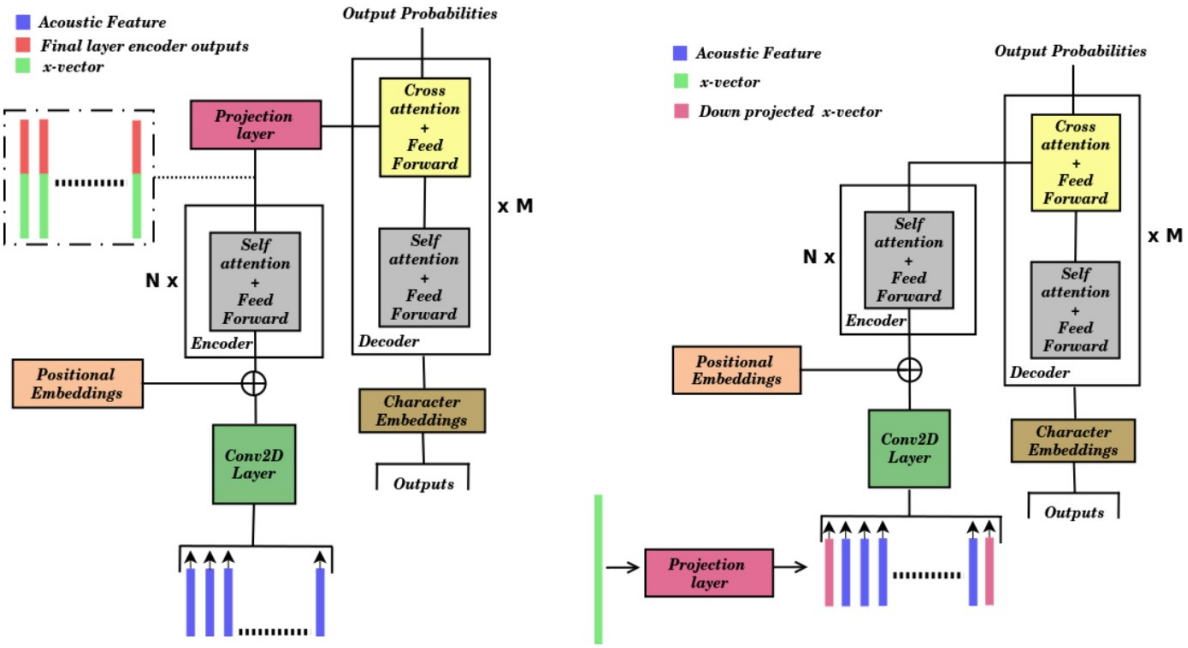
(a) One-hot vector Addition at Embedding (b) One-hot vector Addition at the Final Encoder

Figure 4.16: One-hot vector Addition to Transformer Base Model [SJU20]

In this methodology, the one-hot vectors are linearly projected to match the dimension of the encoder, 256. These vectors are added to different layers of the network. In the first implementation, the projected vectors are added after the convolution layer shown in Figure 4.16a. While in Figure 4.16b, the projected vectors are added to the final encoder output.

x-vector Concatenation

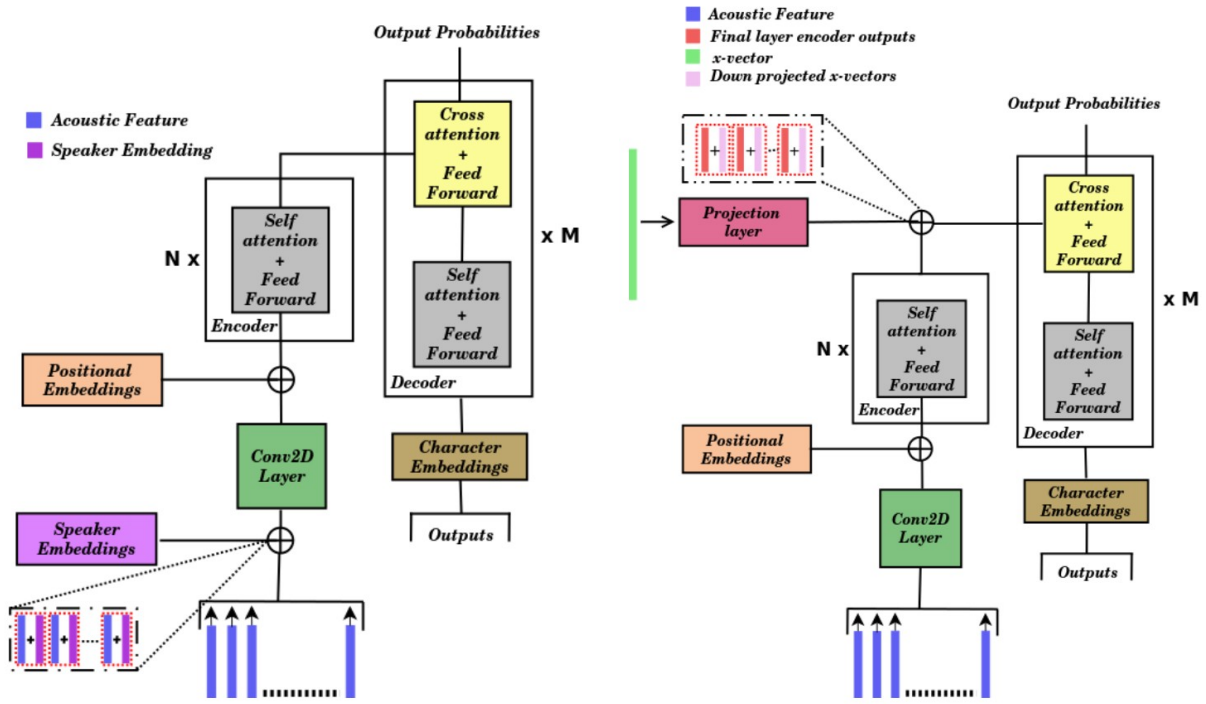
A 100 dimensional x-vector embedding is obtained for each audio utterance using the SID model as discussed above. Figure 4.17a shows the idea of concatenating the x-vectors to the output of the encoder block. In this model, we first projected the 100 dimension x-vectors to 256 dimensions using the linear layer then this projected output is concatenated with the encoder output. At this point, the dimension of the concatenated vectors becomes 512 (speaker vector 256 + encoder output 256) as the encoder generates 256-dimensional vectors for each input utterance. However, the input size of the decoder is 256; thus, to match the decoder's input size, We projected the 512 dimensional concatenated vectors to 256 using another linear layer. These down projected 256-dimensional vectors are then given to the decoder block, as shown in Figure.



(a) x-vector Concatenation with Encoder Output (b) Stacking down projected x-vector as first and last frames along with the input acoustic feature frames

Figure 4.17: x-vector Incorporation to the Transformer Base Model [SJU20]

Figure 4.17b shows another technique to incorporate the x-vectors with the acoustic features frame. The 100-dimensional x-vectors are first down projected to 80 dimensions, so that their dimension matches the 80-dimensional acoustic features vector. After that, this 80-dimensional x-vectors are stacked on the top and bottom of the acoustic features frame corresponding to each utterance. This results, the dimension of acoustic feature frame is increased by 2. These are then passed through the *conv*2*d* layer, as shown in Figure 4.17b.



(a) x-vector Addition to Acoustic feature frames.

(b) x-vector Addition with Encoder Output.

Figure 4.18: x-vector Incorporation to the Transformer Base Model [SJU20]

x-vector Addition

The addition of the x-vectors is carried out in two different ways. Firstly, we directly added it with the acoustic features vector as shown in Figure 4.18a. In order to match its dimension with features vector, we first down projected 100-dimensional x-vectors to 80 dimensional, then it is added to the entire frame of the acoustic features.

In another approach, we add the x-vectors with the output of the encoder. Figure 4.18b depicts the idea of this technique. Here, with the help of a linear layer, 100-dimensional x-vectors are upscaled to 256 dimensions to match the dimension with the encoder output. Then this upscaled 256 dimension x-vectors are added with the encoder output. Since after this adding operation, the dimension remains unchanged, which is the decoder input size, and we pass it to the decoder input as shown in Figure 4.18b.

All these different techniques are trained with the training and development set of the FS-02 dataset. The most common reasons for poor training results in ASR training are the mismatch of speakers in the datasets and the presence of short utterances [Bel+21]. To overcome the shortcomings mentioned above, it is ensured that the same number of speakers is used for both the training and development datasets. The training results are evaluated and compared in Chapter 5.

5 Evaluation

This chapter deals with evaluating all the network architectures investigated so far and comparing results for different metrics.

5.1 Speech Activity Detection

In this section, we evaluated the architectures and compared two different approaches taken during the pre-processing data stage.

We used Binary Cross Entropy loss function, which computes the losses between network output scores and ground-truth binary labels as targets. The Stochastic Gradient Descent optimizer is used with a learning rate initialized as 0.01 and decreased by a factor of 1/10 for every 3 epochs. The training terminates if the loss on the FS-02 Dev dataset is not improved for 10 epochs. To perform the evaluation, the checkpoint with the lowest loss is used.

4 s segmentation

As mentioned earlier in Chapter 4, we segmented the whole audio into 4 s chunks in the first approach, then 199 frames in the time axis and 13 features in the frequency axis are obtained using MFCC transformation of each segment. The audio streams which are not divisible by 4 s chunks are handled with padding during evaluation. These features are then evaluated on the Dev dataset of FS-02 to generate related activity for each of 199 frames.

To evaluate the network performance, the obtained frame-wise estimation is compared with the ground truth labels. Later, a smoothing function is employed to avoid sudden misclassifications for certain intervals in a segment. The true and the estimated activities of each frame are considered for computing the DCF and F1-score. A comparison is made with the following models in this work, and the results are depicted in Table 5.1

Table 5.1: DCF of the SAD systems

Model	Segmentation	Smoothing	F1-score	DCF
CNN	4 s	✓	**95.15**	**2.44**
ResNet-LSTM	4 s	✓	94.15	3.32

Although in our work, the CNN performs better over the ResNet-LSTM in the case of 4 s audio segmentation approach, with the selection of proper hyperparameters, a better result can be achieved using the later architecture.

0.5 s segmentation

The evaluation was carried out on the Dev stream of the FS-02 dataset. The Dev dataset consists of 30 audio streams, each of 30 minutes long. The MFCC features are calculated for half a second chunks for each of the audio streams. The evaluation is carried out for 1,19,940 audio segments. The trained model with the lowest validation loss is considered for the evaluation. The F1-score and DCF metrics are calculated as described in Chapter 2.7.

The targets are either 0 or 1, obtained from the activity dataset of the FS-02 Corpora. These targets are compared with different threshold values of the estimated output of the Dev stream. This estimated output is in the range of between 0 to 1. If the estimated values are greater than the threshold, we assign our prediction to 1 otherwise to 0. Then we compare it with targets and calculate the F1 score and DCF. The one with low DCF is considered the optimal threshold for the model because it sums up the false-negative rate and the false positive rate by weights of 0.75 and 0.25 as mentioned in Chapter 2. Table 5.2 shows the accuracy for different threshold values for the ResNet18 model. For a threshold of 0.3 with the ResNet18 models, the best DCF was obtained.

Table 5.2: Results of different threshold values of ResNet18 model

Threshold	F1-score	DCF
0.3	87.2	**7.82**
0.4	85.4	7.96
0.5	87.8	8.0
0.6	88.1	8.4

As described in Chapter 4 we developed simple CNN before we implemented ResNet18 model. The results of the CNN and ResNet model with corresponding optimal thresholds are shown in Table 5.3. The DCF value of the CNN model is comparable to the ResNet model. These similar findings might result from estimating the target for every half a second in a data preparation stage. Because in some half a second frame, there might be a non-speech activity frame, but we considered it a speech activity using OR logic over whole activity region and then trained the model. In the evaluation, the estimated values are extended to half a second activities to catch up the actual activity length and compared with the target activities. To conclude, the second way of our data preparation might be the reason for the comparable performance of different networks.

Table 5.3: Comparison of CNN and ResNet18 model

Model	Threshold	F1-score	DCF
CNN	0.1	86.3	8.23
ResNet-LSTM	0.3	87.2	7.82

Comparison

In this section, we will compare the performance of two different data preparations. In 0.5 s segmentation, we estimated the target for every 0.5 s with the corresponding activity. This is caused by the second approach of our data preparation because even in some 0.5 s segments, there is a chance of some speech activity frames. However, when we estimated all the frames in the audio segment, we overlooked the remaining non-speech regions and considered as speech activities. In contrast, the other approach follows a mapping of each frame with properly labeled targets, which makes the network learn more efficiently.

When evaluating the network, threshold is applied to the estimated value and then extended to half a second activity intervals and compared to the target values, whereas a smoothing function is used in the latter case after estimating the activity, which reduces FPRs and provides lower DCF score. The DCF scores for both approaches are provided in Tables 5.3 and 5.1.

As mentioned in chapter 4, the main aim to compare the result of the DCF value with the reference paper [LLL20]. The paper achieved the DCF of 1.123% in the dev dataset, whereas our network performs at its best with the DCF value of 3.334%. Although we implemented the model exactly as [LLL20], the result was not close enough due to the choice of hyperparameters, like different combinations of learning rates, batch sizes and epochs. However, from the experiments, we see some improvements in using the ResNet-LSTM network over CNNs.

5.2 Speaker Identity Detection

This section deals with the comparison of results for the SID task. All the models discussed in Subsection 4.2.2 are trained using the same set of parameters. A SGD optimizer and a Categorical CE loss function as in Equation (2.7) are employed for the training process. The overall training was executed for 50 epochs on the Train dataset of FS-02.

The models mentioned in Table 5.4 were trained at a constant learning rate of $(1 \cdot 10^{-3})$. Then, the checkpoint with the lowest validation loss is selected.

The metrics discussed in Section 2.7 are used for the evaluation process on FS-02 Dev dataset contain the exact, same $N_{\text{spk}} = 218$ as the Train dataset.

Initially, the macro-averaging modes for Precision, Recall, and F1-score were chosen for our task.and there is no consideration to the class size since classes with different sizes are equally weighted at the numerator, implying that the classes with more segments have the same importance as the smaller ones. The obtained metrics evaluate the algorithm from a class standpoint: high Macro-F1 values indicate that the algorithm performs well on all the classes, whereas low Macro-F1 values refer to poorly predicted classes [GBV20].
An extended version of macro-averaging is weighted-averaging, which is supported by weights in the form of the number of true instances for each class. The advantage of using weighted averaging over macro mode is to account for class imbalance. The modified metrics were also investigated within this task.

Table 5.4: Comparison of the Accuracy results of various NN models implemented for SID

Model	Accuracy	Top-5 Accuracy
Simple Convolutional Network 1	14.273	38.897
Simple Convolutional Network 2	20.762	46.938
Simple Convolutional Network 3	31.295	58.160

Table 5.5: Comparison of other metrics of various NN models implemented for SID

Model	Precision		Recall		F1-Score	
	Macro	Weighted	Macro	Weighted	Macro	Weighted
Simple Convolutional Network 1	5.345	8.200	10.474	14.273	6.334	9.3352
Simple Convolutional Network 2	10.061	14.630	15.258	20.762	10.873	15.387
Simple Convolutional Network 3	18.225	27.529	21.903	31.295	18.895	27.715

By extending the training of the Deep ResNet vector with a learning rate scheduler of 0.1, 0.01, and 0.001 changing at 25 and 40 epochs, respectively, the results improved from earlier with a constant learning rate and arrived at a Top-5 Accuracy of 88.70% as compared to the 90.78% achieved in [LLL20] model. Since the slow training problem of any NN model can be overcome with the help of a learning rate scheduler, accelerating the training to ensure a faster converging of the optimization problem, we have employed it for our system to achieve better results.

In order to improve the results, an alternative approach by further replacing the Categorical Cross Entropy (CE) with Additive Margin Softmax (ADMS) loss, the Deep ResNet vector was retrained with other hyperparameters remaining the same. However, the results remained incoherent and offered no improvement than the current results. Hence, further experiments were continued with the Categorical CE loss.

Later, the Deep ResNet vector was trained using the modified input logarithmic Mel-energy filterbanks. The Top-5 Accuracy improved to 91.65%, indicating a marginal 0.87% improvement over the baseline from [LLL20]. The results are tabulated in the Tables 5.6 and 5.7.

Table 5.6: Accuracies of various Deep ResNet vector implementations

Model	Accuracy	Top-5 Accuracy
Deep ResNet Vector	61.943	82.304
Deep ResNet Vector (+ LR)	67.711	88.699
Deep ResNet Vector (+ ADMS Loss)	62.283	82.722
Deep ResNet Vector (+ LR + Log-Mel filterbanks)	**76.886**	**91.652**

Table 5.7: Other metrics of various Deep ResNet vector implementations

Model	Precision		Recall		F1-Score	
	Macro	Weighted	Macro	Weighted	Macro	Weighted
Deep ResNet Vector	46.014	62.552	46.846	61.943	45.811	61.225
Deep ResNet Vector (+ LR)	52.864	68.110	53.695	67.711	52.714	67.004
Deep ResNet Vector (+ LR + Log-Mel filterbanks)	46.029	**75.246**	41.149	**76.886**	41.849	**75.532**

5.3 Speaker Diarization

This section discusses the Speaker Diarization (SD) experiments and the results obtained. The first part summarizes the results attained for clustering approaches, followed by the latter part dealing with the systems discussed in Section 4.3.2.

Since the raw FS-02 dataset is heavily corrupted with noise; the x-vectors could not capture different targets from noisy data. Hence, the unsupervised approach could not group or cluster the x-vectors well enough. For further inspection, we reduced the dimension of 100-dim x-vectors. This approach shows that most of the data variability lies in one dimension and is difficult to separate for speaker clustering.

We have used distance measurement techniques such as Euclidean and Manhattan to draw useful information from x-vectors. We tried to compare all the x-vectors with each other from FS-02 segment dataset. The correlation between these vectors are found higher even when it comes to the comparison of different speakers, and thereby the distance between them was found very low.

The similarity score of the x-vectors corresponding to the same speaker with different segments should be higher and it should be lower for the different speakers. Our goal was to find the similarity between audio segments drawn from the stream dataset using SAD with the ground truth audio segments and assign speaker labels with a higher similarity using clustered x-vectors. However, this approach did not work well since all the x-vectors from the segment dataset are very similar, and they only differ in the fourth digit. Hence no helpful information could be drawn from this approach.

5.3.1 Reference Annotations

The reference annotations as described in Subsection 4.3.2 are obtained using the information available in the FS-02 Development stream dataset. An example for the computed annotations can be visualized in Figure 5.1. Here, the annotations are for 200 speech intervals between 12 unique speakers and are based on *who* is the active speaker and *when* did the speaker speak.

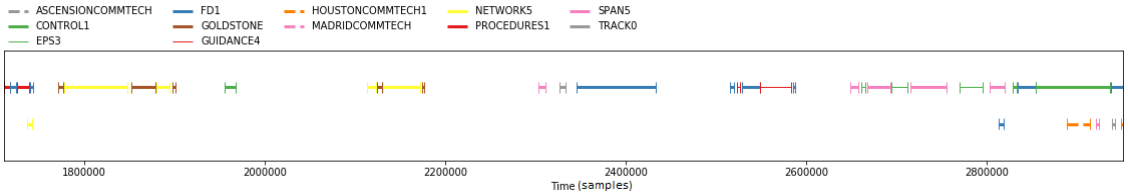


Figure 5.1: Reference Annotations using Ground-truth FS-02 Dev stream audio

5.3.2 Hypothesis Annotations

Groundtruth FS-02 Dev stream audio

The first experiment carried out in our project work with regards to the Speaker Diarization (SD) task is the evaluation of DER with the ground truth speech activity intervals from the FS-02 Development stream dataset and its corresponding speaker identities obtained from the pre-trained Deep ResNet vector of the SID system. The pre-trained model was trained using the FS-02 SID train segments dataset. For this experiment, the achieved DER is 19.39%. All the metrics were calculated concerning the total speech activity duration for the ground truth and were determined to be 5.579 hours. The other metrics concerned with the diarization are tabulated in Table 5.8.

Table 5.8: Diarization results with the groundtruth information

Metrics	Results (%)
False Alarm	0
Missed Detection	0
Confusion	19.39
Correct	80.60
DER	19.39

The parameters were calculated concerning the total speech activity duration for the ground truth was determined to be 5.579 hours. In Table 5.8, the zero value of false alarm and missed detection are since the same ground-truth intervals are used for both the reference and hypothesis annotations. Figure 5.2 shows an example of the hypothesis annotations for the same 200 speech intervals from FS-02 Dev stream dataset considered in Subsubsection 5.3.2 for the system described in Figure 4.11.

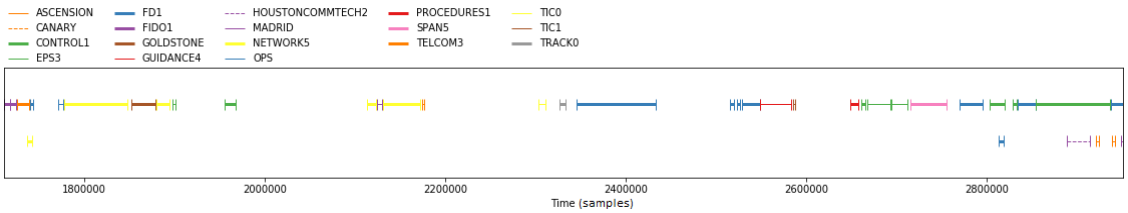


Figure 5.2: Hypothesis Annotations using Ground-truth FS-02 Dev stream audio and corresponding predictions from pre-trained SID system

SAD+SID systems

The speech activity intervals obtained from passing the FS-02 Development stream audios through the pre-trained SAD system and their corresponding speaker predictions from the pre-trained SID system combine to form the basis of this experiment. The attained DER for this experiment is 56.79%, and the remaining parameters are tabulated in the Table 5.9. It can also be further discussed that the increased DER is a result of the errors inherent to both the pre-trained systems being integrated into this task. Errors pertaining to the SAD system are manifested in the form of the lower number of speech activity intervals (around 50% concerning the ground truth FS-02 data) and hence, leading to a lower number of speaker predictions. Figure 5.3 shows an example of the hypothesis annotations for the system described in Figure 4.12.

Table 5.9: Diarization results with the SAD+SID system

Metrics	Results (%)
False Alarm	23.76
Missed Detection	0.22
Confusion	32.80
Correct	66.97
DER	56.79

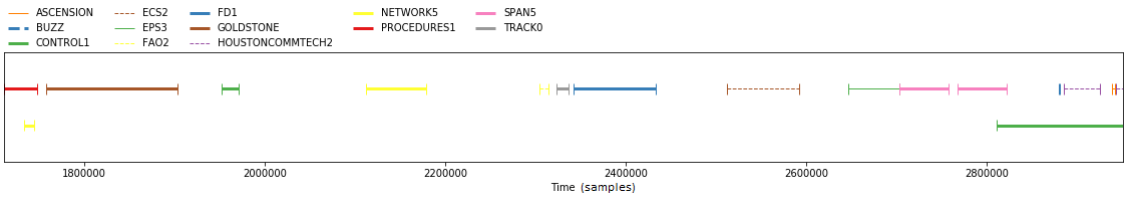


Figure 5.3: Hypothesis Annotations using speech activity intervals from the pre-trained SAD system and corresponding predictions from the pre-trained SID system

An extended approach was to further segment the procured speech activity intervals from the SAD system. All the intervals were split into two halves, and the annotation process was carried out again. This experiment increases the DER to 73.26%. Other metric results are tabulated in Table 5.10.

Table 5.10: Diarization results with the segmented SAD+SID system

Metrics	Results (%)
False Alarm	23.70
Missed Detection	0.22
Confusion	49.32
Correct	50.44
DER	73.26

From Table 5.10, we can observe the increased Confusion and decreased Correct percentages concerning the previous approach. This is because, even with a higher number of intervals, the speaker predictions obtained from the SID system are affected. The SID system needs the segments to be of a minimum length to be correctly predicted and can be visualized in the histogram shown in Figure 5.4. When the activities from SAD are segmented, some of the segmented intervals are not big enough for the SID system to predict the speakers correctly and hence, the fall in the Correct and increased DER percentages.

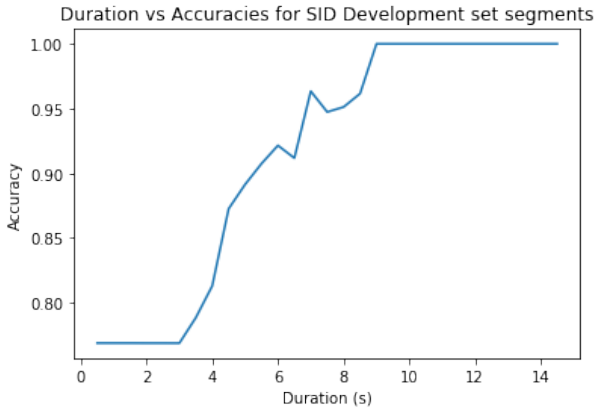


Figure 5.4: Histogram of the segment duration vs SID performance for the FS-02 Dev segments

5.4 Automatic Speech Recognition

In ASR, the transformer is trained using the acoustic features from the FS-02 datasets. We trained the ASR model using the training segments called Train set, and for validation and testing, we used development segments called Dev set. ESPnet provides the facility to create language models. For our experiment, we used language models trained on FS-02 and Wall Street Journal (WSJ). The WSJ language model is based on the WSJ0 corpus. The WSJ0 training corpus consists of 283 speakers evenly split by gender: 142 male and 141 female speakers. The dataset has about 2964 words out of 13,501 unique words in common with the FS-02 dataset.
We initially evaluated the baseline implementations used in the ASR task. For that, We configured the parameters corresponding to the Transformer architecture as follows:

- Number of layers in encoder/decoder: 12/6
- Word embedding size: 256
- Number of heads for transformer self-attention: 4
- Hop length: 70
- Learning Rate (ASR Model Training): 0.0025
- Batch Size: 10

The Baseline model's performance is evaluated and tabulated in Table 5.11.

Table 5.11: Results for the Baseline model

Model Implementation	Language Model	WER (%)
Baseline	None	41.3
	Fearless	39.9
	WSJ	57.8
Baseline with discarding the training utterances having word count less than 3	Fearless	36.8
	WSJ	48.9

It shows that the Fearless language model performs better than WSJ as Fearless consists of more vocabulary relevant to its dataset compared to WSJ. At the same time, discarding the training samples with less than three words reduces the training data by a few hundred more than the original Fearless data set; however, this improved the WER value significantly. In further experiments, based on the results obtained with the base model, we used a fearless language model to evaluate the impact on WER results by integrating speaker information into the Transformer architecture.

To investigate the impact of speaker embedding in the form of a one-hot vector, we ensured that the same speakers are present in both the Train Set and the Dev Set. The number of unique speakers in the Train set and Dev set is 256 and 201, respectively. There are also 167 common speakers in the Train set and the Dev set. The use of only common speakers in both data sets for ASR training further reduces the total number of training and development data, as shown in Table 5.12.

Table 5.12: Data used for training Transformer Model with incorporation of Speaker Information

Dataset	Number of Original Data	Number of Speakers	Number of Data used for one-hot Speaker Embedding	Number of Data used for x-vector Embedding
Train set	35,474	256	33,345	30,978
Dev set	9,203	201	9,029	8,462

We also trained the Base model with the same dataset used for one-hot speaker Embedding, consisting of common speakers in both Train and Dev sets. We found that the reduction in WER value was obtained for both the Base model and the model embedding the speakers in the form of a one-hot vector. This indicates that the inclusion of speaker information in a one-hot vector did not help the model learn the speaker information to improve the training and validation. This is due to the highly imbalanced nature of FS-02 data set as a certain number of speakers in the Dev set have a lower number of utterances compared to others as shown in Figure 5.5 and Table 5.14, and that impacts the model performance. The WER results of speaker embedding in the form of a one-hot vector are given in Table 5.13.

Table 5.13: Results of Speaker Embedding in the form of a one-hot vector

Model Implementation	WER (%)
Base Model	**32.9**
Concatenation of one-hot vectors to acoustic features	44.0
One-hot vector Addition at Embedding	**32.9**
One-hot vector Addition at the Encoder output	33.0

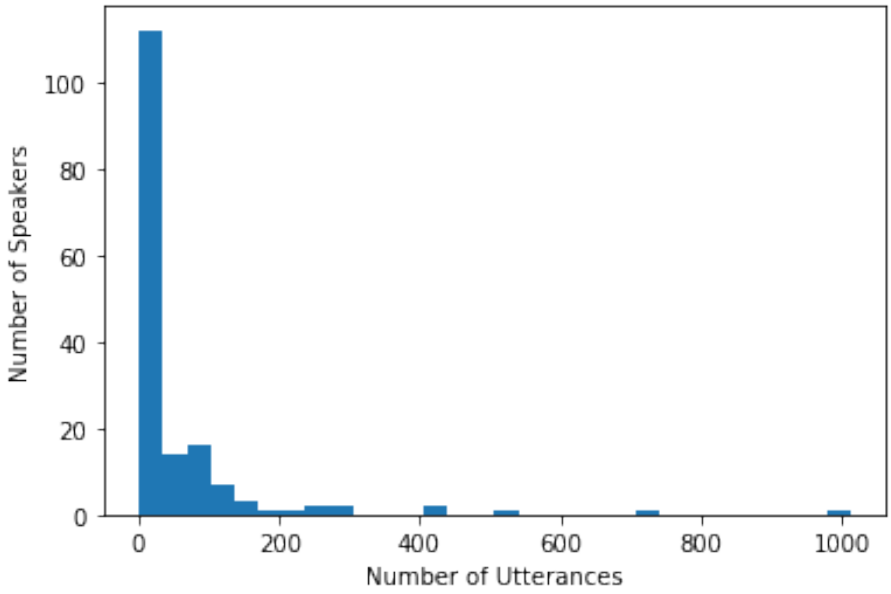


Figure 5.5: Utterance-Speaker disproportion

Table 5.14: Utterance-Speaker disproportion values

Utterances	Number of speakers
More than 100	21
Less than 100 and greater than 50	23
Less than 50 and greater than 25	19
Less than 25	100
Less than 10	61
Less than 5	37
Less than 2	12

The inclusion of speaker information in the form of x-vectors was implemented similarly to the inclusion of one-hot vectors in the Transformer model. We performed the addition and concatenation of x-vectors directly with acoustic features and with the output of the encoder. As explained in the architecture section, we applied a linear transformation to the x-vectors to align the dimensions before performing these operations. However, since the x-vectors are extracted from the internal layers of the deep ResNet network of the SID model, their dimension is 100. At the same time, the SID model only includes the utterance segments in the model that have more than 4000 samples and therefore generates the x-vectors only for these segments. This leads to a further subtraction of the training samples for the ASR model training. The actual amount of data available for ASR training contains the speaker information in the form of x-vectors as specified in the Table 5.12.

Table 5.15: Results after Speaker Information Incorporation with Transformer Model

Model Implementation	WER (%)
Base Model with data set available for x-vector speaker Embedding	38.2
x-vector Concatenation with Encoder Output	39.4
Stacking the x-vector on the top and bottom of the each acoustic features frame.	37.7
x-vector Addition with Encoder Output	39.6
x-vector addition to acoustic features	38.3

At the same time, we also trained the Base model with the same number of data available for x-vectors incorporation for a fair performance comparison between the Base model without containing any speaker information and the Model incorporating speaker information in the form of x-vector. The effects of the x-vectors in the Transformer model with different techniques are given in Table 5.15.

The results show that the WER value is significantly higher than the experimental results given in the Table 5.13. This is due to two reasons: First, to reduce the 2,367 training audio segments with less than 4000 samples. Second, we measured the separability of x-vectors corresponding to different speakers by determining the cosine similarity and the Equal Error Rate (EER). The cosine similarity and EER between two randomly selected x-vectors corresponding to two different speakers averaged 0.98 and 0.97, respectively, indicating that the quality of the x-vectors is not good enough when it comes to introducing information about the different speakers into the transformer model and that the model, therefore, does not benefit adequately from the inclusion of the x-vectors in order to improve WER significantly than the Base Model.

Comparison

The results of the different approaches to train the ASR model using the Fearless Step datasets are given in the Tables 5.11 and 5.15. This confirms that after ensuring that the common speakers are present in both the Train Set and the Dev Set, the Base model gives the best value for WER, i.e., 32.9%. Therefore, in our complete system, we use the Base model for transcribing the Diarized speaker segments. The WER value of the Speaker-Diarized segments transcribed with the ASR Base model is 59.4%.

6 Summary

With the ongoing advancements in the field of Artificial Intelligence and Machine Learning, Neural Networks can be used to process speech, identify the speakers and interpret the text better than ever. From talking to robots to aiding the visually and hearing-impairments, speech recognition plays a vital role in people's lives. This project work primarily focuses on the development of an NN-based speech recognition system using the audio data from Fearless Steps - 02 (FS-02) Corpus from NASA's Apollo-11 space mission having long recordings. In this work, we presented different network architectures for the development of our work and compared revised baseline results. For the system, speech processing was employed, and analyzed using three sub-tasks: Speaker Activity Detection (SAD), Speaker Identity Detection (SID), and Automatic Speech Recognition (ASR), with supervised learning strategies.

For the SAD, different DCF scores are obtained after the evaluation of different models on the Dev dataset of FS-02. It indicates the effects of the data preparation and the choosing of different hyper-parameters. The 0.5 s segmentation of the data preparation produced comparable result with CNN and ResNet-LSTM model. However, in the case of 4 s segmentation, the network can perform frame-wise estimation and provides better accuracy. The CNN and LSTM based ResNet systems are employed to realize the baseline result.

The developed SID system predicts the speaker identity by incorporating the features extracted from the speaker attributes with the help of Log Mel-filterbanks. Several architectures were experimented with in this task. When evaluated on the FS-02 Dev dataset, the deep ResNet vector system achieved the best results with Top-5 Accuracy of 91.65% as compared to the 90.78% baseline result and an Accuracy of 76.89%. The speakers in the dataset are not equally distributed which leads to severe class imbalance, the models investigated fail to generalize equally well for all the speakers.

In ASR, the experiments were conducted to investigate the effects of including speaker information in the Transformer Base model in the form of one-hot embedding as well as x-vectors. The results indicate that the Base model achieves the best WER value, 32.9% compared to other techniques that incorporate speaker information. Since the FS-02 dataset is very unbalanced and the speakers are not evenly distributed in the dataset, this leads to an inferior quality of x-vectors from the pre-trained SID-ResNet model, which could not introduce additional information about the speakers into the transformer model to improve the training.

Also, Speaker Diarization (SD) systems were investigated using the supervised learning approaches implemented for the SAD and SID systems. The first experiment was carried out with FS-02 development stream data which resulted in a DER of 19.39%. Later, another experiment involved the combination of SAD+SID systems which achieved a DER of 56.78%.

Due to the imperfections in the systems, the errors manifest and lead to a higher DER. Further, the available speech activity intervals from the pre-trained SAD system were incorporated into the pre-trained ASR model. The ASR model generates speech transcriptions of speech activity intervals and gives a WER value of 59.4%.

For Future work, the system currently utilizes only the FS-02 information and to further improve the standalone results or to implement it as a real-time system, other speech-related datasets can be employed.

List of Figures

List of Tables

Acronyms

ADMS Additive Margin Softmax.

ASR Automatic Speech Recognition.

BCE Binary Cross Entropy.

CE Cross Entropy.

CER Character Error Rate.

CNN Convolutional Neural Network.

CTC Connectionist Temporal Classification.

DCF Detection Cost Function.

DCT Discrete Cosine Transform.

DER Diarization Error Rate.

Dev Development data-set.

DFT Discrete Fourier Transform.

DNN Deep Neural Network.

EER Equal Error Rate.

FAR False Acceptance Rate.

FAR False Rejection Rate.

FFT Fast Fourier Transformation.

FN False Negatives.

FNR False Negative Rate.

FP False Positives.

FPR False Positive Rate.

FS-01 Fearless Steps - 01.

FS-02 Fearless Steps - 02.

LM Language Model.

LR Learning Rate.

LSTM Long Short-Term Memory.

MET Mission Elapsed Time.

MFCC Mel-Frequency Cepstral Coefficient.

MLP Multi-Layer Perceptron.

NN Neural Network.

NNs Neural Networks.

ReLU Rectified Linear Unit.

ResNet Residual Neural Network.

RNN Recurrent Neural Network.

ROC Receiver Operating Characteristic.

S-D Speaker-dependent.

SAD Speaker Activity Detection.

SD Speaker Diarization.

SGD Stochastic Gradient Descent.

SI Speaker-independent.

SID Speaker Identity Detection.

STFT Short Time Fourier Transform.

TN True Negatives.

TP True Positives.

TPR True Positive Rate.

WER Word Error Rate.

Bibliography

[Bre17] H. Bredin. "pyannote.metrics: a toolkit for reproducible evaluation, diagnostic, and error analysis of speaker diarization systems". In: *Interspeech 2017, 18th Annual Conference of the International Speech Communication Association*. Stockholm, Sweden, 2017. URL: http://pyannote.github.io/pyannote-metrics.

[Bre+20] H. Bredin et al. "pyannote.audio: neural building blocks for speaker diarization". In: *ICASSP 2020, IEEE International Conference on Acoustics, Speech, and Signal Processing*. Barcelona, Spain, 2020.

[Bye19] F. Byers. *2017 Pilot Open Speech Analytic Technologies Evaluation (2017 NIST Pilot OpenSAT): Post Evaluation Summary*. 2019.

[CDL16] J. Cheng, L. Dong, and M. Lapata. *Long Short-Term Memory-Networks for Machine Reading*. 2016. arXiv: 1601.06733 [cs.CL].

[Cho+15] J. K. Chorowski et al. "Attention-Based Models for Speech Recognition". In: *Advances in Neural Information Processing Systems*. Ed. by C. Cortes et al. Vol. 28. Curran Associates, Inc., 2015. URL: https://proceedings.neurips.cc/paper/2015/file/1068c6e4c8051cfd4e9ea8072e3189e2-Paper.pdf.

[CM02] D. Comaniciu and P. Meer. "Mean shift: A robust approach toward feature space analysis". In: *IEEE Transactions on pattern analysis and machine intelligence* 24.5 (2002), pp. 603–619.

[Dan21] T. A. Dang. "Accuracy and Loss: Things to Know about The Top 1 and Top 5 Accuracy". In: (2021). URL: https://towardsdatascience.com/accuracy-and-loss-things-to-know-about-the-top-1-and-top-5-accuracy-1d6beb8f6df3.

[DHS01] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. 2nd ed. New York: Wiley, 2001. ISBN: 978-0-471-05669-0.

[DV18] V. Dumoulin and F. Visin. *A guide to convolution arithmetic for deep learning*. 2018. arXiv: 1603.07285 [stat.ML].

[DXX18] L. Dong, S. Xu, and B. Xu. "Speech-Transformer: A No-Recurrence Sequence-to-Sequence Model for Speech Recognition". In: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2018, pp. 5884–5888. DOI: 10.1109/ICASSP.2018.8462506.

[Est+96] M. Ester et al. "A density-based algorithm for discovering clusters in large spatial databases with noise." In: *kdd*. Vol. 96. 34. 1996, pp. 226–231.

[Fay16] H. M. Fayek. "Speech Processing for Machine Learning: Filter banks, Mel-Frequency Cepstral Coefficients (MFCCs) and What's In-Between". In: (2016). URL: https://haythamfayek.com/2016/04/21/speech-processing-for-machine-learning.html.

[GBV20] M. Grandini, E. Bagli, and G. Visani. "Metrics for Multi-Class Classification: an Overview". In: (2020). arXiv: 2008.05756 [stat.ML].

[Gor+20] A. Gorin et al. *"This is Houston. Say again, please". The Behavox system for the Apollo-11 Fearless Steps Challenge (phase II)*. 2020. arXiv: 2008.01504 [eess.AS].

[Han+18] J. Hansen et al. "Fearless Steps: Apollo-11 Corpus Advancements for Speech Technologies from Earth to the Moon". In: Sept. 2018, pp. 2758–2762. DOI: 10.21437/Interspeech.2018-1942.

[Has+15] H. Hassan et al. "Assessment Of Artificial Neural Network For Bathymetry Estimation Using High Resolution Satellite Imagery In Shallow Lakes: Case Study El Burullus Lake." In: *International Water Technology Journal* 5 (Dec. 2015).

[He+15] K. He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV].

[HL93] X. Huang and K. Lee. "On speaker-independent, speaker-dependent, and speaker-adaptive speech recognition". In: *IEEE Transactions on Speech and Audio Processing* 1.2 (1993), pp. 150–157. DOI: 10.1109/89.222875.

[HS15] K. He and J. Sun. "Convolutional Neural Networks at Constrained Time Cost". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015.

[HW20] Y. Ho and S. Wookey. "The Real-World-Weight Cross-Entropy Loss Function: Modeling the Costs of Mislabeling". In: *IEEE Access* 8 (2020), 4806–4813. ISSN: 2169-3536. DOI: 10.1109/access.2019.2962617. URL: http://dx.doi.org/10.1109/ACCESS.2019.2962617.

[Jog+20] A. Joglekar et al. "FEARLESS STEPS Challenge (FS-2): Supervised Learning with Massive Naturalistic Apollo Data". In: (2020). arXiv: 2008.06764 [eess.AS].

[KHW17] S. Kim, T. Hori, and S. Watanabe. "Joint CTC-attention based end-to-end speech recognition using multi-task learning". In: *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2017, pp. 4835–4839. DOI: 10.1109/ICASSP.2017.7953075.

[Kir+19] S. Kiranyaz et al. *1D Convolutional Neural Networks and Applications: A Survey*. 2019. arXiv: 1905.03554 [eess.SP].

[Kuh55] H. W. Kuhn. "The Hungarian method for the assignment problem". In: *Naval Research Logistics Quarterly* 2.1-2 (1955), pp. 83–97. DOI: https://doi.org/10.1002/nav.3800020109. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/nav.3800020109. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/nav.3800020109.

[LLL20] Q. Lin, T. Li, and M. Li. "The DKU Speech Activity Detection and Speaker Identification Systems for Fearless Steps Challenge Phase-02". In: (Aug. 2020).

[Mik+13] T. Mikolov et al. “Distributed Representations of Words and Phrases and Their Compositionality”. In: *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*. NIPS’13. Lake Tahoe, Nevada: Curran Associates Inc., 2013, 3111–3119.

[MY19] M. C. S. Muhamad Yani Budhi Irawan. “Application of Transfer Learning Using Convolutional Neural Network Method for Early Detection of Terry’s Nail”. In: (2019). URL: https://www.researchgate.net/publication/333593451_Application_of_Transfer_Learning_Using_Convolutional_Neural_Network_Method_for_Early_Detection_of_Terry’s_Nail.

[Raj20] B. Rajoub. “Chapter 2 - Characterization of biomedical signals: Feature engineering and extraction”. In: *Biomedical Signal Processing and Artificial Intelligence in Healthcare*. Ed. by W. Zgallai. Academic Press, 2020. DOI: https://doi.org/10.1016/B978-0-12-818946-7.00002-0. URL: https://www.sciencedirect.com/science/article/pii/B9780128189467000020.

[RB19] M. Ravanelli and Y. Bengio. “Speaker Recognition from Raw Waveform with SincNet”. In: (2019). arXiv: 1808.00158 [eess.AS].

[RBR19] J. Rownicka, P. Bell, and S. Renals. *Embeddings for DNN speaker adaptive training*. 2019. arXiv: 1909.13537 [cs.CL].

[Rei20] Reinhold Haeb Umbach. “Statistical and Machine Learning”. In: (2020).

[S19] H. S. “Activation Functions : Sigmoid, ReLU, Leaky ReLU and Softmax basics for Neural Networks and Deep Learning”. In: (Jan. 2019). URL: https://medium.com/@himanshuxd/activation-functions-sigmoid-relu-leaky-relu-and-softmax-basics-for-neural-networks-and-deep-8d9c70eed91e.

[San+19] S. Santurkar et al. *How Does Batch Normalization Help Optimization?* 2019. arXiv: 1805.11604 [stat.ML].

[SGS15] R. K. Srivastava, K. Greff, and J. Schmidhuber. *Highway Networks*. 2015. arXiv: 1505.00387 [cs.LG].

[SJU20] V. M. Shetty, M. S. M. N. J, and S. Umesh. *Investigation of Speaker-adaptation methods in Transformer based ASR*. 2020. arXiv: 2008.03247 [eess.AS].

[ST13] U. Shrawankar and V. M. Thakare. “Techniques for Feature Extraction In Speech Recognition System : A Comparative Study”. In: (May 2013).

[Tan+16] T. Tan et al. “Speaker-aware training of LSTM-RNNS for acoustic modelling”. In: *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2016. DOI: 10.1109/ICASSP.2016.7472685.

[US20] R. H. Umbach and J. Schmalenströer. “Digital Speech Signal Processing”. In: (2020).

[Vas+17] A. Vaswani et al. *Attention Is All You Need*. 2017. arXiv: 1706.03762 [cs.CL].

[Wan+18] F. Wang et al. “Additive Margin Softmax for Face Verification”. In: *IEEE Signal Processing Letters* 25.7 (2018), pp. 926–930. DOI: 10.1109/LSP.2018.2822810.

[Wat+17] S. Watanabe et al. “Hybrid CTC/Attention Architecture for End-to-End Speech Recognition”. In: *IEEE Journal of Selected Topics in Signal Processing* 11.8 (2017), pp. 1240–1253. DOI: 10.1109/JSTSP.2017.2763455.

[Wat+18a] S. Watanabe et al. *ESPnet: End-to-End Speech Processing Toolkit*. 2018. arXiv: 1804.00015 [cs.CL].

[Wat+18b] S. Watanabe et al. "ESPnet: End-to-End Speech Processing Toolkit". In: *Proc. Interspeech 2018*. 2018, pp. 2207–2211. DOI: 10.21437/Interspeech.2018-1456. URL: http://dx.doi.org/10.21437/Interspeech.2018-1456.

[WK92] J. J. Webster and C. Kit. "Tokenization as the Initial Phase in NLP". In: *Proceedings of the 14th Conference on Computational Linguistics - Volume 4*. COLING '92. Nantes, France: Association for Computational Linguistics, 1992, 1106–1110. DOI: 10.3115/992424.992434. URL: https://doi.org/10.3115/992424.992434.

[WWL19] D. Wang, X. Wang, and S. Lv. "An Overview of End-to-End Automatic Speech Recognition". In: *Symmetry* 11.8 (2019). ISSN: 2073-8994. DOI: 10.3390/sym11081018. URL: https://www.mdpi.com/2073-8994/11/8/1018.

[Yaa+20] M. N. Yaacob et al. "Decision Making Process in Keystroke Dynamics". In: *Journal of Physics: Conference Series* 1529 (Apr. 2020), p. 022087. DOI: 10.1088/1742-6596/1529/2/022087.

[Zia+14] A. Ziaei et al. "Speech Activity Detection for NASA Apollo Space Missions: Challenges and Solutions". In: Sept. 2014.

[ZWZ20] X. Zhang, W. Wang, and P. Zhang. "Speaker Diarization System Based on DPCA Algorithm for Fearless Steps Challenge Phase-2". In: *Proc. Interspeech 2020*. 2020, pp. 2602–2606. DOI: 10.21437/Interspeech.2020-1666.