SummaryPro

Thesis Submitted in partial fulfillment
of the Requirements of the Degree of

Master of Science with Specialization in

Artificial Intelligence

by

Deep Kishor Malde

Roll Number – 04

G.R. Number – 3511618

Under the Supervision of

Dr. Vivek Sharma

April 2023

Nagindas Khandwala College (Autonomous)

Malad, Mumbai 400064

CERTIFICATE

This is to certify that the project titled, **"SummaryPro"**, is bonafied work of **"Malde Deep Kishor"** (**Roll No: 04** and **G.R. No: 3511618**) submitted to the Nagindas Khandwala College (Autonomous), Mumbai in partial fulfillment of the requirements for the award of degree of **"Masters of Science with Specialization in Artificial Intelligence".**

Dr. Vivek Sharma

Internal Examiner

External Examiner

Supervisor's Certificate

This is to certify that the dissertation entitled **"SummaryPro"** submitted by **"Malde Deep Kishor"**, **Roll No: 04** and **G.R. No: 3511618**, is a record of original work carried out by him under my supervision and guidance in partial fulfillment of the requirements of the degree of **Master Of Science with Specialization in Artificial Intelligence** at Nagindas Khandwala College (Autonomous), Mumbai 400064. Neither this dissertation nor any part of it has been submitted earlier for any degree or diploma to any institute or university in India or abroad.

Dr. Vivek Sharma

Internal Examiner

External Examiner

Declaration of Originality

I, ***Malde Deep Kishor*, *Roll No: 04*** *and* ***G.R. No: 3511618***, hereby declare that this dissertation entitled **"*SummaryPro*"** presents my original work carried out as a Master Student of Nagindas Khandwala College (Autonomous), Mumbai 400064. To the best of my knowledge, this dissertation contains no material previously published or written by another person, nor any material presented by me for the award of any degree or diploma of Nagindas Khandwala College (Autonomous), Mumbai or any other institution. Any contribution made to this research by others, with whom I have worked at Nagindas Khandwala College (Autonomous), Mumbai or elsewhere, is explicitly acknowledged in the dissertation. Works of other authors cited in this dissertation have been duly acknowledged under the sections "Reference" or "Bibliography". I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I am fully aware that in case of any non-compliance detected in future, the Academic Council of Nagindas Khandwala College(Autonomous),Mumbai may withdraw the degree awarded to me on the basis of the present dissertation.

Date: 11-April-2023

Place: Malad, Mumbai 400064

Malde Deep Kishor

ACKNOWLEDGEMENT

It's my pleasure to be indebted to varied people, who directly or indirectly contributed to the event of this work and who influenced our thinking, behavior and acts during the course of study.

I express my sincere gratitude to Coordinator **Dr. Pragati Hirwarkar** for her support, cooperation, and motivation provided to me during the training for constant inspiration, presence and blessings.

I'm thankful to the Project guide, **Dr. Vivek Sharma**, for his valuable suggestions which helps us lot in completion of this project. Lastly,

I might wish to thank the almighty and our parents for his or her moral support and friends with whom we shared our day-to-day experience and received plenty of suggestions that improved our quality of work.

Thanks for all your encouragement!

Malde Deep Kishor

Abstract

"No More Missed Points, No More Inefficient Meetings with SummaryPro"

World has greatly realised the powers of Artificial Intelligence in this century. Every domain seems to be reaping the fruits of this powerful set of tools available. AI has become a catchall term for applications that perform complex tasks that once required human input, such as communicating with customers online or playing chess. It has a number of subfields like Computer Vision, Natural Language Processing, etc. We further discuss the project which is a great solution for a modern day problem.

The global pandemic drastically changed the way with which we professionally worked. Hybrid and Remote Jobs are getting more and more appreciations and push. Along with these, online meetings and discussions too are on surge. With these increasing phenomena of online interactions, there raised a problem for the people who couldn't attend a few important meetings sighting to their very busy schedule, they missed important information which was supposed to be known! But there isn't anything to worry about, I have come up with a solution for this. A meeting summarizer which listens, understands, processes and summarizes the key highlights of an online meeting is described further. We leverage the insane powers of Natural Language Processing and Deep Learning, LSTM (Long Short Term Memory) to be precise for achieving our goals.

Phases:

1) Audio Transcription

2) Text pre-processing

3) Concept Extraction

4) Concept Representation

5) Concept Clustering

6) Summary Generation

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Problem Statement of Thesis

SummaryPro is a novel solution designed to address the growing concern regarding the inefficiency and time-consuming nature of manually taking meeting minutes. The problem is particularly challenging because meetings are essential for decision-making, problem-solving, planning, and brainstorming in the corporate world. Therefore, recording accurate meeting minutes is essential to ensure accountability and refer back to decisions made.

The COVID-19 pandemic has further complicated the problem, resulting in a shift towards virtual meetings that have increased the chances of ineffective interactions. As a result, there is a growing demand for a tool that can simplify the task of taking meeting minutes, offering a more efficient and streamlined approach to meeting management.

One of the main drawbacks of manual note-taking is human error, which may result in misinterpreted information or missed essential points. Additionally, manual note-taking is a time-consuming process that diverts valuable resources from other essential tasks. Therefore, SummaryPro aims to automate the process of recording and transcribing meetings using deep learning techniques, offering an accurate and comprehensive record of all discussions and decisions made during the meeting.

SummaryPro uses Mel Frequency Cepstral Coefficient to identify speakers, which helps to distinguish between different speakers, ensuring accurate and reliable transcriptions. The use of deep neural networks for converting audio files into plain text and transformers for summarizing the transcript into condensed minutes, offer an innovative solution for addressing the challenges of taking meeting minutes.

Moreover, the proposed solution addresses the need for a more efficient and streamlined approach to meeting management, particularly for the fast-paced and rapidly evolving business world. With the increasing demand for virtual meetings and the shift towards remote work, there is a growing need for a solution that can keep pace with these changes and simplify the task of taking meeting minutes.

SummaryPro offers an innovative, automated solution that meets these needs, enabling teams to be more productive, organised, and effective in their daily operations. The solution eliminates the need for manual note-taking, freeing up resources for other essential tasks. Additionally, the use of deep learning techniques offers a higher degree of accuracy and reliability, ensuring that all critical points are captured during the meeting.

In conclusion, SummaryPro is a game-changing solution that offers an automated, efficient, and reliable approach to meeting management. It eliminates the risk of human error and saves valuable time and resources while ensuring accurate and comprehensive record-keeping. The solution is particularly relevant in today's fast-paced and rapidly evolving business world, where virtual meetings and remote work have become the norm. SummaryPro offers a streamlined solution that meets the needs of modern businesses, enabling them to stay productive, organised, and effective.

1.2 Objectives

The objectives of SummaryPro are to:

- Streamline meeting management processes: The primary objective of SummaryPro is to simplify the process of meeting management. By automating the task of recording and transcribing meetings, the solution aims to reduce the workload on human resources and provide an efficient and streamlined approach to meeting management.

- Provide a comprehensive and accurate record: Another crucial objective of SummaryPro is to provide a complete and accurate record of all discussions and decisions made during the meeting. By eliminating the risk of missing essential points, the tool ensures that teams have a reliable and comprehensive record to refer back to when needed.

- Improve productivity and efficiency: SummaryPro aims to reduce the time and resources spent on manual minute-taking, resulting in an increase in productivity and efficiency. By automating the task of taking meeting minutes, teams can focus on other essential tasks and be more productive.

- Enhance the quality of meetings: By eliminating the chances of missing important points and decisions, SummaryPro aims to enhance the quality of meetings. This tool helps to ensure that all decisions made during meetings are accurately recorded and shared with the team.

- Provide an easy-to-use and accessible tool: SummaryPro is designed to be an easy-to-use and accessible tool for businesses of all sizes. The solution aims to simplify the process of meeting management for small and large businesses alike, providing a user-friendly interface and an efficient approach to meeting management.

- Leverage deep learning and NLP techniques: SummaryPro leverages the latest advancements in deep learning and NLP techniques to deliver high-quality meeting summaries in real-time. By using Mel Frequency Cepstral Coefficient and deep neural networks, the solution can accurately transcribe meetings and summarise them into condensed minutes.

- Support continuous improvement and learning: SummaryPro provides an accurate and reliable record of meetings that can be referred back to in the future. By doing so, the tool supports continuous improvement and learning, enabling teams to identify areas for improvement and make necessary adjustments.

- Enhance communication and collaboration: Finally, SummaryPro aims to enhance communication and collaboration among team members. By providing a clear and concise record of meetings that can be shared and discussed, the solution helps to ensure that all team members are on the same page and working towards a common goal.

1.3 Existing System

There are several existing systems that are similar to SummaryPro in terms of their use of NLP and deep learning for audio transcription and summarization.

One such system is Otter.ai, which uses ASR technology and NLP algorithms to transcribe audio files into text and generate summaries. It also allows users to highlight important parts of the audio and add notes to the summary.

Another system is Sonix.ai, which uses machine learning and natural language processing to transcribe audio files and generate summaries. It also has a feature called "Speaker Identification," which automatically labels different speakers in the audio file, similar to SummaryPro's use of Mel Frequency Cepstral Coefficient, MFCC, to identify speakers.

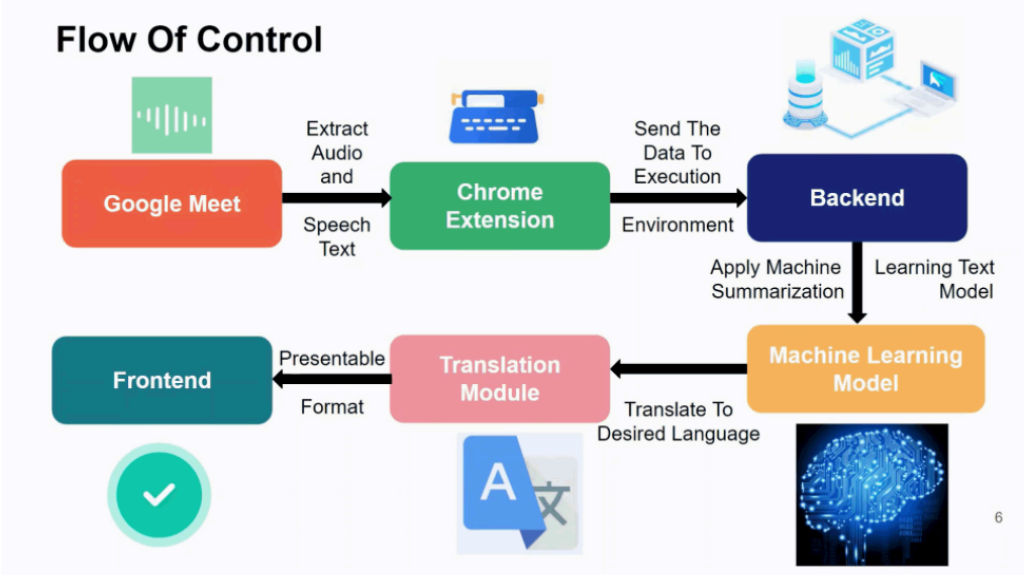
Another system similar to SummaryPro is Speechmatics, which uses deep learning and neural networks to transcribe audio files and generate summaries. It also has a feature called "Speaker Diarization," which can identify different speakers in the audio and label them accordingly.

Overall, these systems are similar to SummaryPro in terms of their use of NLP and deep learning for audio transcription and summarization. However, SummaryPro's specific focus on business meetings and its combination of extractive and abstractive summarization techniques may differentiate it from other existing systems.

1.4 Proposed System

The increasing volume of data available online has made text summarization an important area of research. With the development of deep learning as a machine learning area in the past decade, it has been applied to various fields, including speech recognition. Numerous research papers have been published on using deep learning techniques to improve the accuracy of traditional speech recognition systems based on Hidden Markov Models (HMMs) and Gaussian Mixture Models (GMMs). With the advancements in deep learning, researchers are exploring ways to incorporate these techniques into speech summarization to make the summarization process more efficient and accurate. In this article, we present SummaryPro, an Automated Meetings Summarizer that uses NLP and deep learning to transcribe and summarise audio files into clear minutes.

A tool designed to streamline the process of meeting management and provide a comprehensive and accurate record of all discussions and decisions



System Flow:

Extractive Summarization:

Extractive summarization is a simple yet effective way to summarise text by selecting the most important phrases and terms from the source text, creating a condensed summary that retains the key information. This approach selects relevant sentences or phrases from the text and combines them to create a summary. It ensures that the summary is faithful to the original text and can be easily verified.

Abstractive Summarization:

Abstractive summarization goes one step further than extractive summarization by creating a new, more concise message that provides an overview of the most important information in the text. Unlike extractive summarization, it doesn't just extract information, but generates new content that is a distilled version of the source text. It uses advanced NLP techniques to understand the meaning of the text and create a summary that captures the essence of the original text.

SummaryPro combines these techniques to revolutionise the process of meeting management. It provides businesses with a reliable and efficient way to retain records of their meetings, reducing the chances of missing important information. Additionally, its ability to transcribe and summarise audio files in real-time can help increase productivity and reduce the cost of ineffective meetings, contributing to the growth of businesses.

The AMBOC Model:

The development of our model was inspired by the AMBOC model, a well-established and researched deep learning model for text summarization. The AMBOC model was developed to create automated meeting summaries by identifying and extracting the most important concepts and information from meeting audio files. The process involved several stages, including audio transcription, text pre-processing, concept extraction, concept representation, concept clustering, and summary generation.

Audio Transcription:

In the first stage, the audio file of the meeting was transcribed into text using automatic speech recognition (ASR) technology. This stage ensured that the content of the audio file was converted into a format that could be processed and analysed by the model.

Text Pre-processing:

After the audio file was transcribed, the text was pre-processed to remove any errors, noise, and irrelevant information. This stage helped in improving the quality of the data used for further processing and analysis.

Concept Extraction:

The pre-processed text was then passed through a concept extraction module that used NLP techniques to identify and extract the key concepts and information from the text. This stage helped in focusing on the important aspects of the meeting, which were relevant for summarization.

Concept Representation:

The extracted concepts were then represented in a high-dimensional vector space using a bag-of-concepts (BOC) approach. In this approach, each dimension represented a unique concept.

2. LITERATURE SURVEY

Using latent semantic evaluation in textual content summarization and summary evaluation
- Josef Steinberger., KarelJezek.

- The authors propose the use of latent semantic evaluation (LSE) to evaluate the quality of text summarization algorithms and summaries.
- LSE is based on latent semantic analysis (LSA) and overcomes the limitations of traditional evaluation methods.
- The authors provide a comprehensive overview of the current state of the field of text summarization, including both extractive and abstractive approaches and existing evaluation methods and their limitations.
- The LSE approach measures the semantic similarity between a summary and the original text.
- The results of the experiments show that LSE is effective and provides more reliable results compared to traditional evaluation methods, and has the potential to serve as a universal evaluation method for different summarization algorithms and improve the accuracy and reliability of summary evaluation.

Automatic minute generation for parliamentary speech using conditional random fields. Acoustics, Speech, and Signal Processing
- Zhang, Justin Jian Fung, Pascale Chan, Ricky

- The study aimed to find meeting minutes in legislative speeches by dividing the text into smaller chunks.
- The prominent characteristics of each chunk were extracted using a machine learning classifier called the Conditional Random Field (CRF).
- The chunks were stored in a tree structure using a logical syntax tree to make it easier to extract important features from sentences.
- The accuracy of the experiment was evaluated using the ROGUE-L F-Measure, with a result of 73.2% accuracy.
- The study has limitations, including the prepared nature of parliament remarks and the computational complexity of the CRF training process, which makes it challenging to retrain the model with new data if necessary.

Automated Generation of Meeting Transcription Using Deep Learning Techniques
- Megha Manuel1, Amritha S Menon1 , Anna Kallivayalil1 , Suzana Isaac1 and Lakshmi K.S2

- The AMBOC system is divided into three components: speech-to-text, speaker verification, and text organisation.
- The AMBOC system delivers improved results through the utilization of Google Speech API, MFCC, and Transformers technology.
- The study is currently limited to the Indonesian language.
- The use of the Google API has resulted in reduced accuracy when applied to other languages

3. REQUIREMENTS

3.1 Software Requirements:

- Operating System: Windows 10, macOS, or Linux
- Python 3.7 or higher
- TensorFlow and Keras
- Numpy, Scipy, Pandas, and Scikit-learn
- NLTK and Gensim

3.2 Hardware Requirements:

- Processor: Intel Core i5 or higher
- RAM: 8 GB or higher
- Storage: 500 GB HDD or 256 GB SSD
- Sound card and microphone (for recording meetings)
- Internet connectivity (for downloading required libraries and updates)
- Note: The hardware requirements may vary depending on the size and duration of the meetings being summarized. Larger and longer meetings may require higher processing power and more storage.

4. METHODOLOGY

The growing volume of data available online has made text summarization an increasingly important area of research. The development of deep learning as a machine learning area in the past decade has led to its application in a wide range of fields. Deep learning has been applied to the field of speech, including speech recognition, with numerous research papers published on the topic. These studies have focused on using deep learning techniques to improve the accuracy of traditional speech recognition systems, which are based on Hidden Markov Models and Gaussian Mixture Models. These systems use HMMs to describe speech signals and provide a foundation for traditional speech recognition. With the advancements in deep learning, researchers are now exploring ways to incorporate these techniques into speech summarization and make the summarization process more efficient and accurate.

An Automated Meetings Summarizer, SummaryPro, is a crucial tool for efficient and accurate records of business meetings. It uses NLP and deep learning to transcribe and summarise audio files into clear minutes. The tool eliminates manual minute-taking and reduces the risk of missing important information. The tool uses Mel Frequency Cepstral Coefficient (MFCC) to identify speakers, allowing it to accurately transcribe the audio into plain text, ready for summarization.

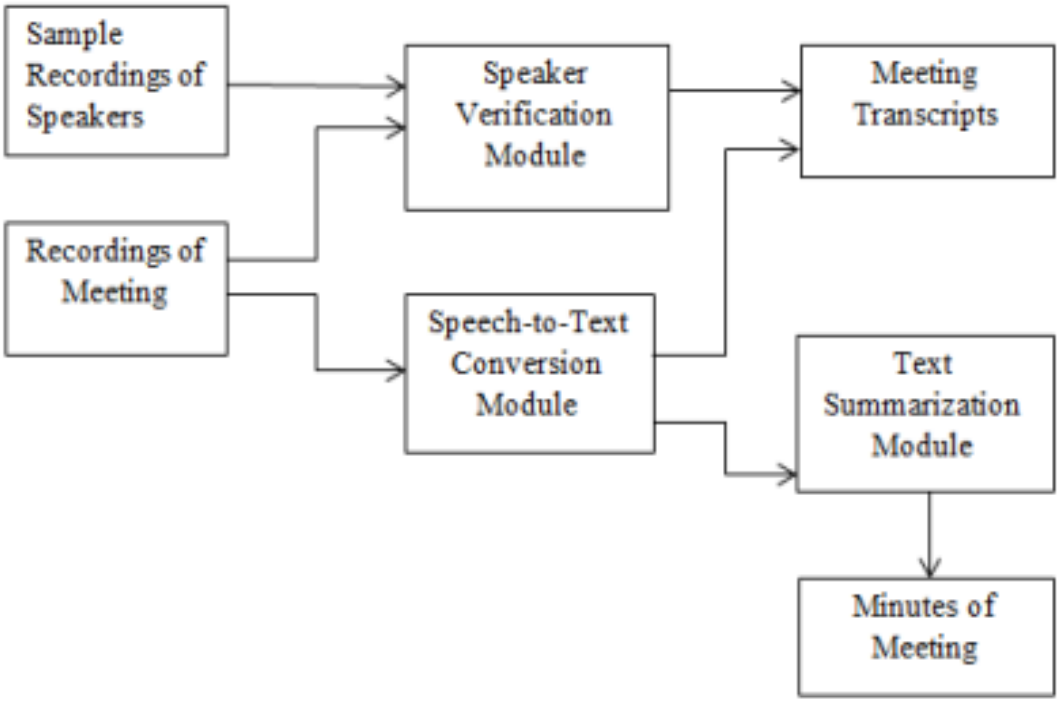
To achieve the best results, SummaryPro employs two NLP-based summarization techniques: extractive and abstractive summarization.

I. Extractive Summarization - This approach selects the most important phrases and terms from the source text, creating a condensed summary that retains the key information. It is a simple yet effective way to summarise text by picking out the most relevant sentences or phrases.

II. Abstractive Summarization - This technique goes one step further by creating a new, more concise message that provides an overview of the most important information in the text. Unlike extractive summarization, it doesn't just extract information, but generates new content that is a distilled version of the source text.

By combining these techniques, SummaryPro has the potential to revolutionise the process of meeting management. It provides businesses with a reliable and efficient way to retain records of their meetings, reducing the chances of missing important information. Additionally, its ability to transcribe and summarise audio files in real-time can help increase productivity and reduce the cost of ineffective meetings, contributing to the growth of businesses.

The development of our model was inspired by the AMBOC model, a well-established and researched deep learning model for text summarization. We took the fundamental concepts and architecture of the AMBOC model and made modifications to suit our specific requirements and to improve its performance on the task of summarising business meetings. Our model uses advanced techniques in Natural Language Processing (NLP) and deep learning, such as Mel Frequency Cepstral Coefficient (MFCC) and a combination of extractive and abstractive summarization, to transcribe and summarise audio files of meetings into comprehensive and accurate minutes. By taking inspiration from the AMBOC model and incorporating our own modifications, we have created a highly effective and efficient tool for meeting management.



The AMBOC model was developed to create automated meeting summaries by identifying and extracting the most important concepts and information from meeting audio files. The process involved the following stages:

Audio transcription: In the first stage, the audio file of the meeting was transcribed into text using automatic speech recognition (ASR) technology. This stage ensured that the content of the audio file was converted into a format that could be processed and analysed by the model.

Text pre-processing: After the audio file was transcribed, the text was pre-processed to remove any errors, noise, and irrelevant information. This stage helped in improving the quality of the data used for further processing and analysis.

Concept extraction: The pre-processed text was then passed through a concept extraction module that used NLP techniques to identify and extract the key concepts and information from the text. This stage helped in focusing on the important aspects of the meeting, which were relevant for summarization.

Concept representation: The extracted concepts were then represented in a high-dimensional vector space using a bag-of-concepts (BOC) approach. In this approach, each dimension represented a unique concept. This stage helped in representing the concepts in a numerical format that could be processed and analysed by the model.

Concept clustering: The BOC representation of the concepts was then clustered to identify the most important and relevant concepts that best described the meeting. This stage helped in identifying the key topics discussed in the meeting and in generating a summary that captured the essence of the meeting.

Summary generation: The final stage involved using the identified and clustered concepts to generate a concise and informative summary of the meeting. This stage combined the results from the

previous stages to produce a summary that captured the essence of the meeting and conveyed the most important information in a concise and easy-to-understand format.

By following these stages, the AMBOC model was able to create accurate and comprehensive meeting summaries that accurately captured the key information and concepts discussed during the meeting. Our model takes inspiration from the AMBOC model, but may have some modifications or improvements based on the specific requirements and needs of our task.

Implementation Details:

Datasets to be used:

Our research uncovered that previous studies often utilise similar corpora for their experiments. It makes sense that these corpora can serve dual purposes, both for speech recognition (converting speech into text) and audio/text summarization. To make it easy for others to replicate these studies, we have compiled a list of the corpora used in the evaluations found in various publications. This list is available in the form of a table and indicates whether the corpora are publicly available or needs to be obtained through a request. The majority of the speech-related corpora are in English, contain one or two speakers, and range in size from small to moderate, with only a few surpassing 500 hours.

Corpus	Summarised Content	Language	No. of speakers	Size
AMI	Meeting	English	More than 2	100 hours
ICSI	Meeting	English	More than 2	70 hours
MATRICS	Multimodal meeting	English	More than 2	10 hours
TEDe	Lecture	English	1	50 hours+75 hours
CSJ	Lecture; Task-oriented dialogue	Japanese	1 or 2	658 hours
TDT2	Broadcast news	English	1 or 2	518-1036 hours

1. **AMI Meeting Corpus**: This dataset contains multi-modal data from meetings, including audio, transcriptions, and annotations. The dataset can be downloaded from the following link: http://groups.inf.ed.ac.uk/ami/corpus/

2. **ICSI Meeting Corpus**: This corpus contains recordings of meetings from the International Computer Science Institute. The dataset can be downloaded from the following link: https://sail.usc.edu/public_datasets/icsi/

3. **NIST Meeting Corpus**: This corpus contains audio and transcriptions of meetings recorded in a conference room setting. The dataset can be downloaded from the following link: https://www.nist.gov/itl/iad/mig/nist-sre/2004-2006-meeting-corpus

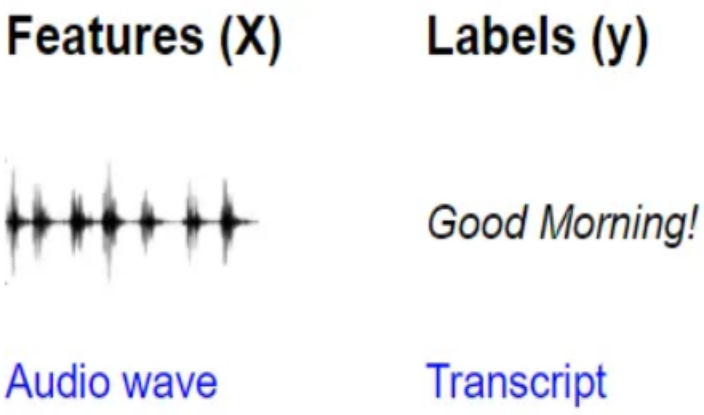
4. **Meetings Marburg**: This corpus contains recordings and annotations of meetings held in a business setting. The dataset can be downloaded from the following link: https://www.ukp.tu-darmstadt.de/data/meetings-corpus/

5. **QMSum - Yale University:** QMSum is a new human-annotated benchmark for query-based multi-domain meeting summarization task, which consists of 1,808 query-summary pairs over 232 meetings in multiple domains.
https://github.com/Yale-LILY/QMSum#readme

6. Timit Dataset - Linguistic Data Consortium & University of Pennsylvania: The TIMIT Acoustic-Phonetic Continuous Speech Corpus is a standard dataset used for evaluation of automatic speech recognition systems. It consists of recordings of 630 speakers of 8 dialects of American English each reading 10 phonetically-rich sentences. It also comes with the word and phone-level transcriptions of the speech.
https://catalog.ldc.upenn.edu/LDC93s1

Audio Transcription - *aka Automatic Speech Recognition (ASR):*

The aim of the model is to understand how to use the audio input to anticipate the text information of the spoken words and sentences.



Automatic Speech Recognition uses audio waves as input features and the text transcript as target labels

A. Data pre-processing:

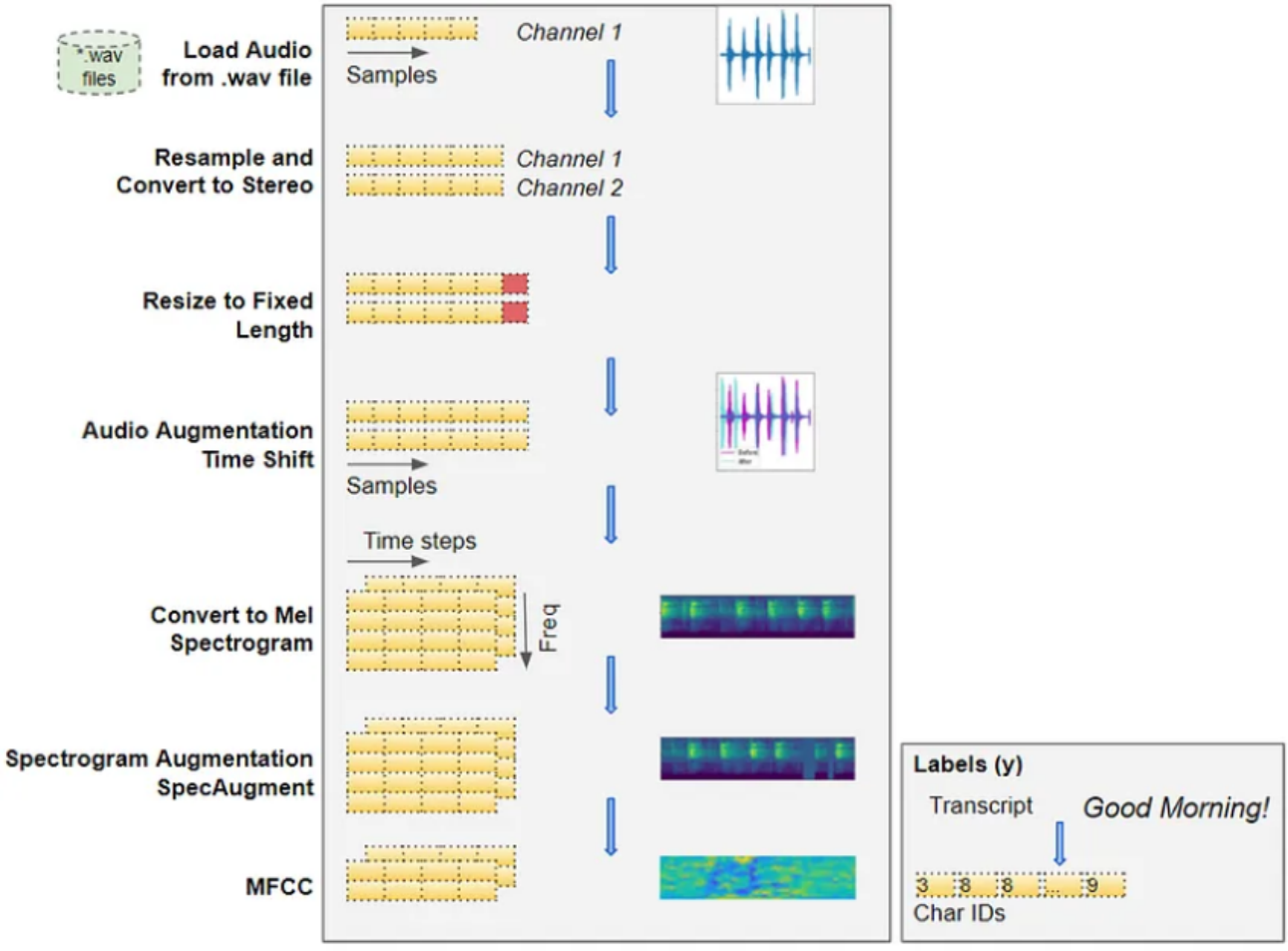
1. Load Audio Files:

a. The input data for speech recognition models typically consists of audio recordings in formats such as ".wav" or ".mp3".

b. To process this audio data, it needs to be extracted and converted into a 2D Numpy array where each number represents the intensity or amplitude of the sound at a specific moment in time.

c. The number of measurements in the Numpy array is determined by the audio's sampling rate, with a single row of numbers corresponding to one second of audio.

d. Audio recordings can be either mono or stereo, resulting in a 2D or 3D Numpy array, respectively, with one or two sequences of amplitude numbers.



2. Convert to uniform dimensions: sample rate, channels, and duration:

a. Audio data can vary in sample rates, channels, and durations, leading to differences in dimensions.

b. For effective deep learning, input items must have uniform dimensions.

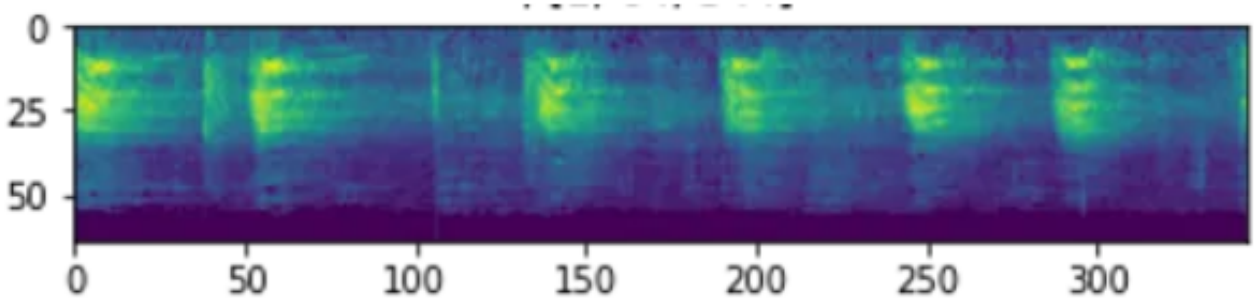
c. Data cleaning steps can standardize dimensions by resampling, converting channels, and standardizing duration.
d. Padding or truncating can be used to ensure consistent duration.
e. A noise-removal algorithm can improve spoken audio focus if the quality is poor.

3. Data Augmentation of raw audio:

a. Data augmentation techniques can be used to add more variety to the input data, which helps the model learn to generalize to a wider range of inputs.
b. Time Shifting can be applied to the audio by randomly shifting it left or right by a small percentage.
c. Changing the Pitch or Speed of the audio by a small amount can also be used as a data augmentation technique.

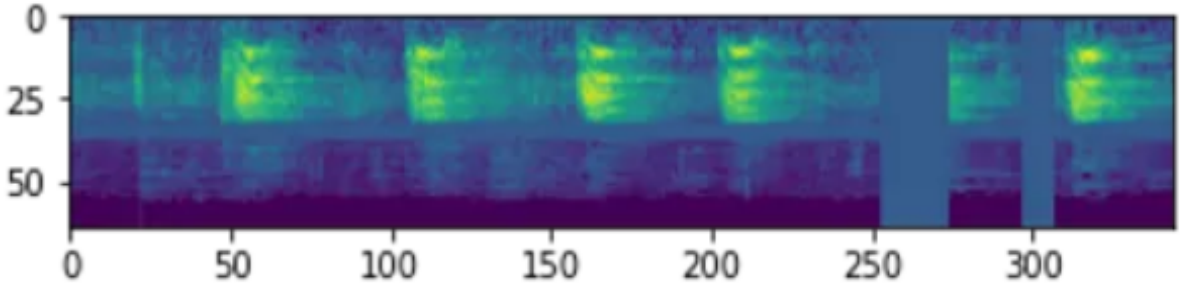
4. Mel Spectrograms:

a. The raw audio data is transformed into Mel Spectrograms, which represent the audio as an image by decomposing it into its constituent frequencies.



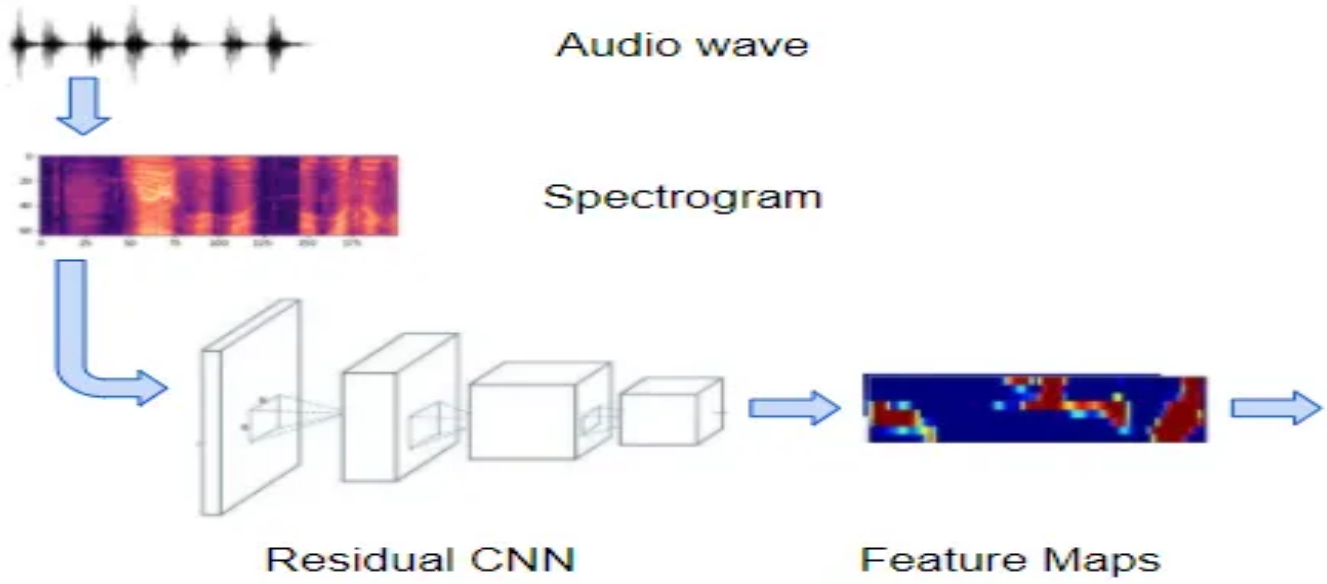
5. MFCC (Mel Frequency Cepstral Coefficients):

a. It is beneficial to further process Mel Spectrograms into MFCC (Mel Frequency Cepstral Coefficients), especially for human speech.
b. MFCCs compress the Mel Spectrogram by extracting the most important frequency coefficients, which correspond to the frequency ranges used in human speech.

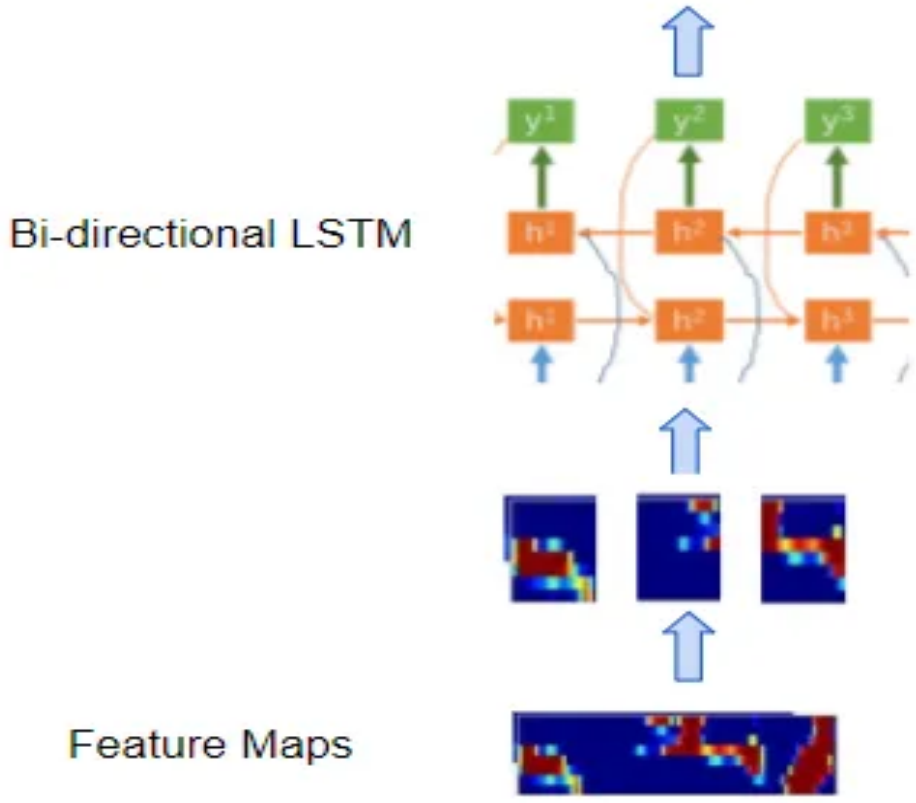


B. Model Architecture

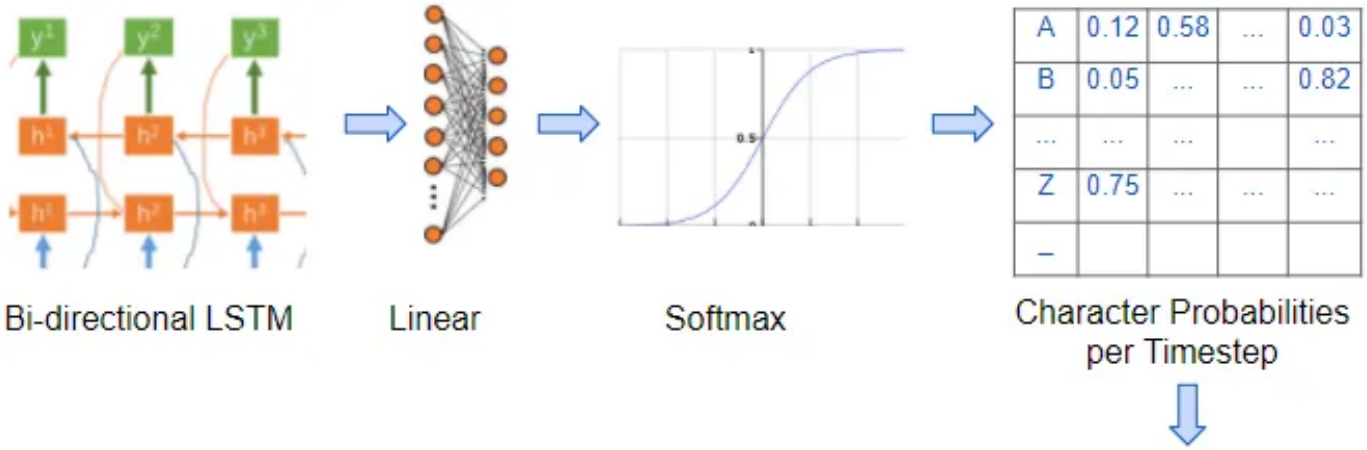
To recognize spoken words, a common approach is to use a speech recognition model such as the Deep Speech model from Baidu. This model combines a CNN and RNN architecture and employs the CTC Loss algorithm to demarcate each character in the speech.



1. A regular convolutional network consisting of a few Residual CNN layers that process the input spectrogram images and output feature maps of those images.

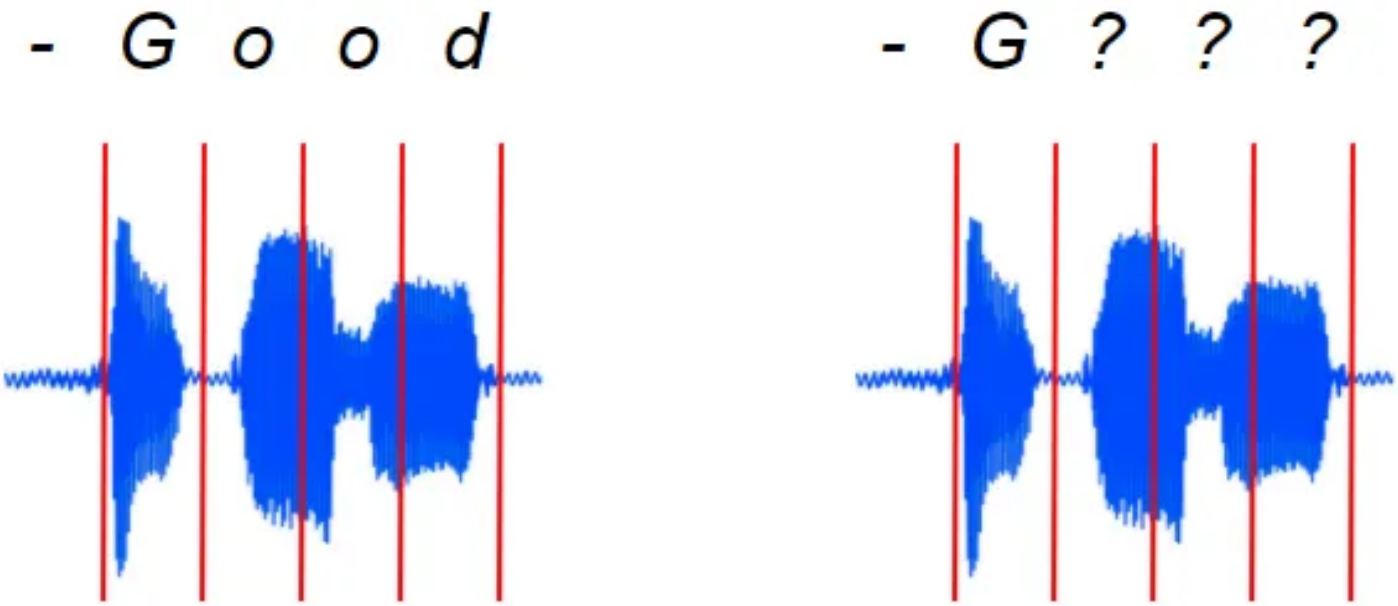


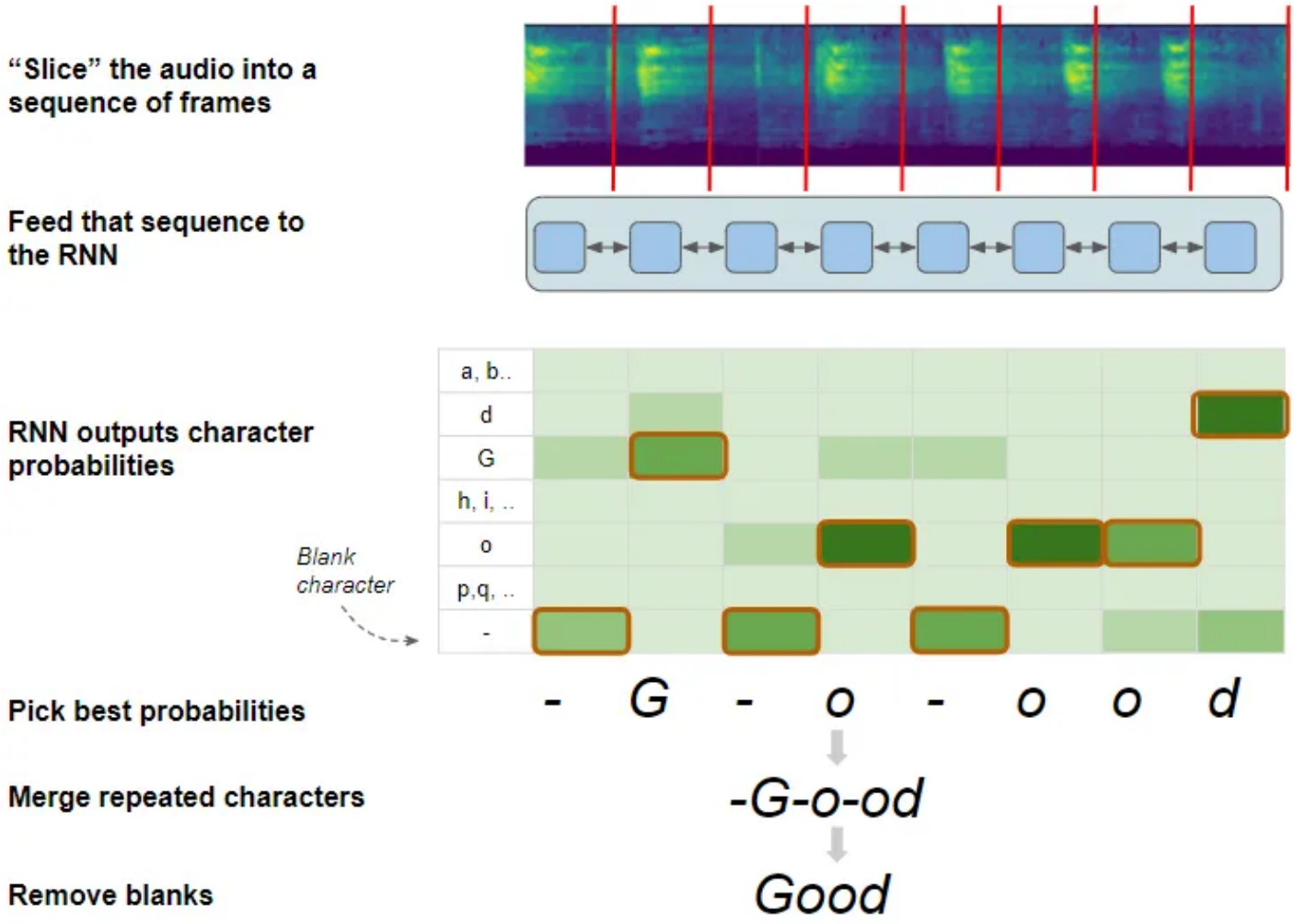
2. A linear layer with softmax is utilized to generate character probabilities for each timestep of the output, based on the LSTM outputs.



3. In addition, there are linear layers that act as an intermediate step between the convolutional and recurrent networks. These layers aid in transforming the outputs of one network into the appropriate inputs for the other.

4. Our model receives the Mel Spectrogram images as input and generates a set of probabilities for each character at every timestep, corresponding to a frame in the Spectrogram.





While describing the CTC Decoder during Inference, we implicitly assumed that it always picks a single character with the highest probability at each timestep.

5. FLOW CONTROL

The flow of control in a Chrome extension using automatic meeting summarization can be described as follows:

Audio Recording: The Chrome extension will start by recording the audio of the meeting, either through the user's microphone or by importing an existing audio file.

Transcription: The audio file will then be transcribed into text using automatic speech recognition (ASR) technology.

Pre-processing: The transcribed text will undergo pre-processing to remove errors, noise, and irrelevant information.

Concept Extraction: The pre-processed text will then be passed through a concept extraction module that uses NLP techniques to identify and extract the key concepts and information from the text.

Concept Representation: The extracted concepts will then be represented in a high-dimensional vector space using a bag-of-concepts (BOC) approach, where each dimension represents a unique concept.

Concept Clustering: The BOC representation of the concepts will then be clustered to identify the most important and relevant concepts that best describe the meeting.

Summary Generation: The final stage involves using the identified and clustered concepts to generate a concise and informative summary of the meeting.

Displaying the Summary: The generated summary will be displayed in the Chrome extension, allowing the user to easily access and review the most important information from the meeting.

The Chrome extension will leverage existing technologies and techniques in ASR and NLP to provide a reliable and efficient solution for meeting summarization. With its ability to automatically transcribe and summarise audio files in real-time, it has the potential to significantly improve the productivity and efficiency of businesses.

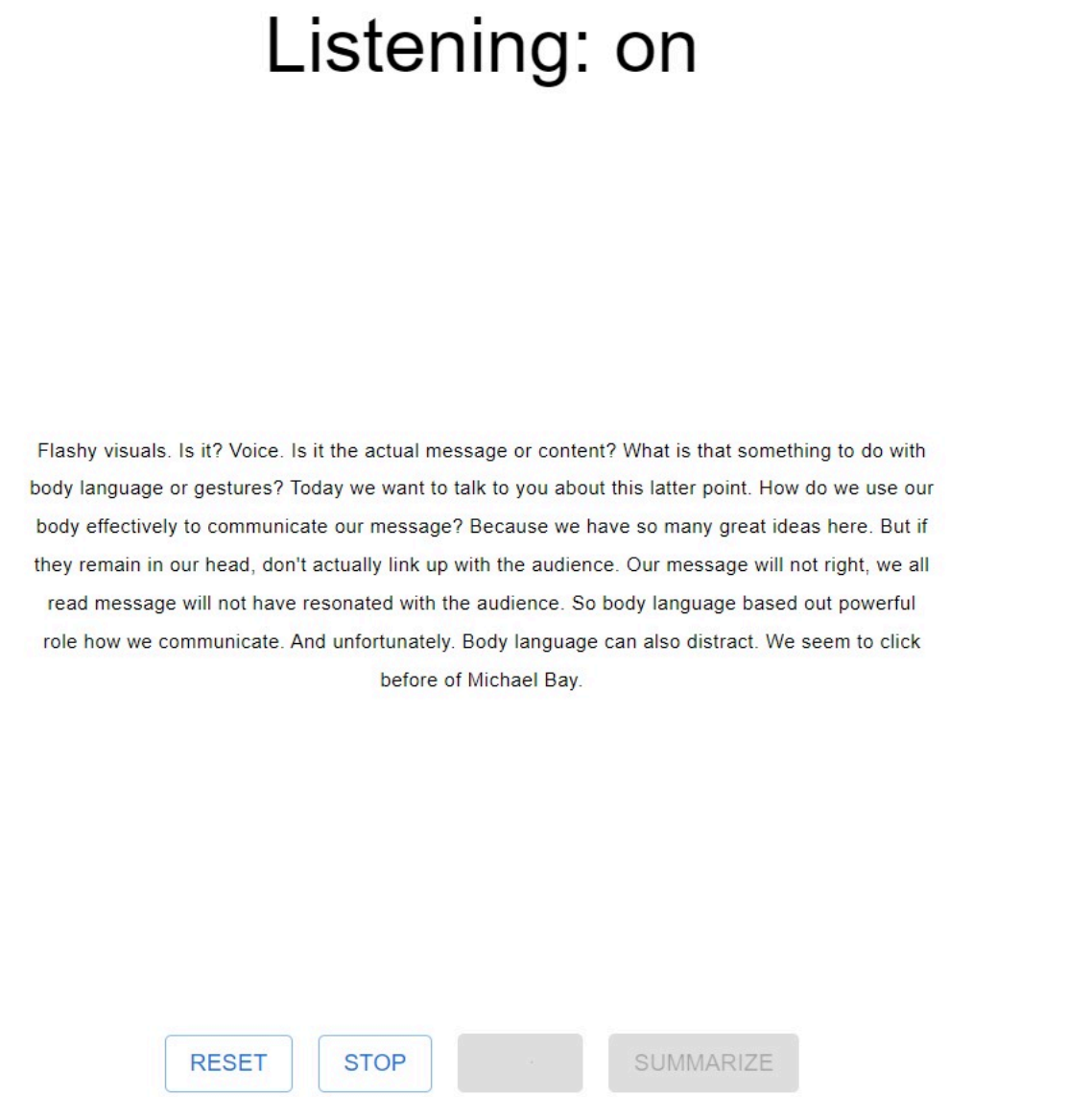
"No More Missed Points, No More Inefficient Meetings with SummaryPro"

6. Screenshots

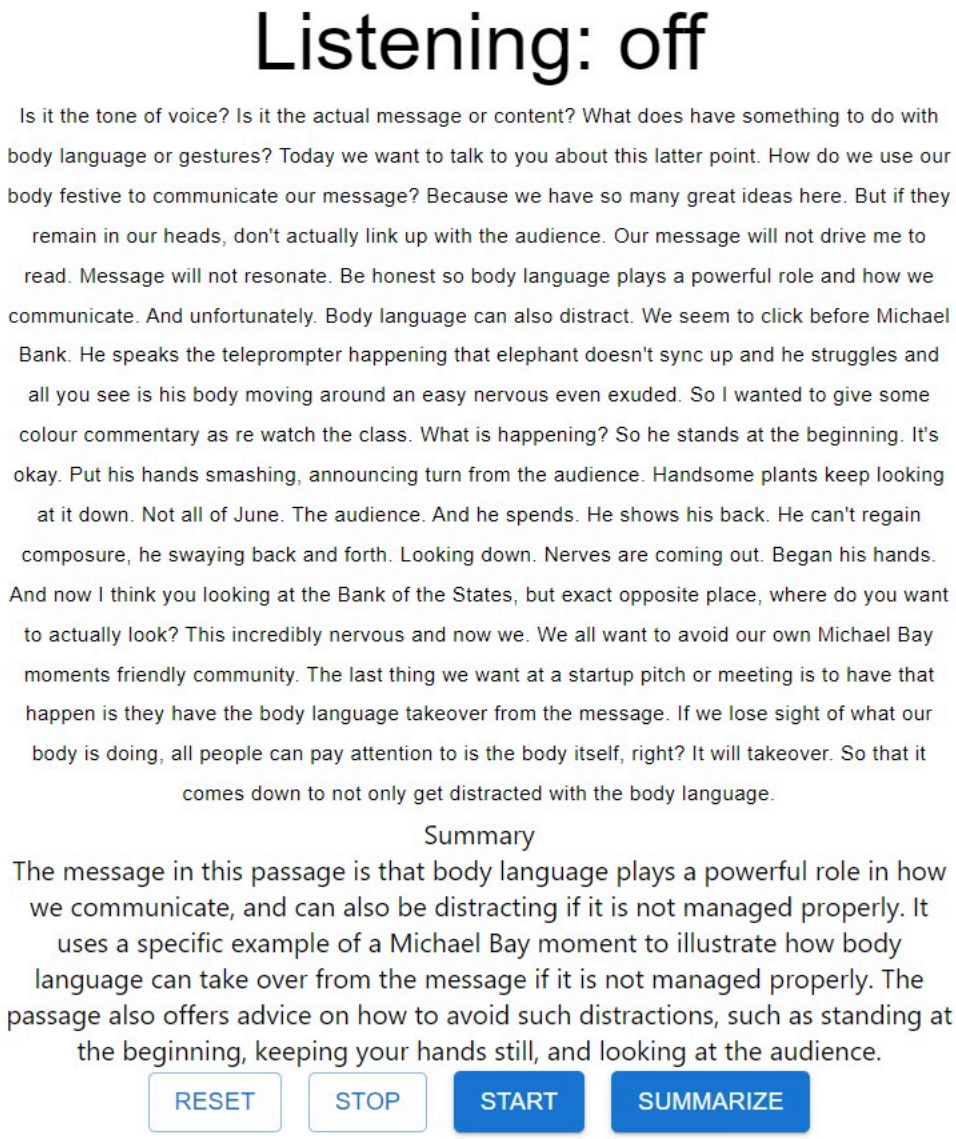
1. Initial Launch Screen

2. Listening On

3. Summarizing after transcript is generated



4. Summarized content.



7. CODE

Import the Libraries

```
import numpy as np
import pandas as pd
import re
from bs4 import BeautifulSoup
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from nltk.corpus import stopwords
from tensorflow.keras.layers import Input, LSTM, Embedding, Dense, Concatenate, TimeDistributed
from tensorflow.keras.models import Model
from tensorflow.keras.callbacks import EarlyStopping
import warnings
pd.set_option("display.max_colwidth", 200)
warnings.filterwarnings("ignore")
```

```
Using TensorFlow backend.
```

```
[ ] data=pd.read_csv("../input.csv",nrows=100000)
```

Drop Duplicates and NA values

```
[ ] data.drop_duplicates(subset=['Text'],inplace=True)#dropping duplicates
    data.dropna(axis=0,inplace=True)#dropping na
```

Information about dataset

```
[ ] data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 88421 entries, 0 to 99999
Data columns (total 10 columns):
Id                      88421 non-null int64
ProductId               88421 non-null object
UserId                  88421 non-null object
ProfileName             88421 non-null object
HelpfulnessNumerator    88421 non-null int64
HelpfulnessDenominator  88421 non-null int64
Score                   88421 non-null int64
Time                    88421 non-null int64
Summary                 88421 non-null object
Text                    88421 non-null object
dtypes: int64(5), object(5)
memory usage: 7.4+ MB
```

Preprocessing

Performing basic preprocessing steps is very important before we get to the model building part. Using messy and uncleaned text data is a potentially disastrous move. So in this step, we will drop all the unwanted symbols, characters, etc. from the text that do not affect the objective of our problem.

```
[ ] ntraction_mapping = {"ain't": "is not", "aren't": "are not","can't": "cannot", "'cause": "because", "could've": "could have", "couldn't": "could not",
                         "didn't": "did not",  "doesn't": "does not", "don't": "do not", "hadn't": "had not", "hasn't": "has not", "haven't": "have not",
                         "he'd": "he would","he'll": "he will", "he's": "he is", "how'd": "how did", "how'd'y": "how do you", "how'll": "how will", "how's":
                         "I'd": "I would", "I'd've": "I would have", "I'll": "I will", "I'll've": "I will have","I'm": "I am", "I've": "I have", "i'd": "i w
                         "i'd've": "i would have", "i'll": "i will",  "i'll've": "i will have","i'm": "i am", "i've": "i have", "isn't": "is not", "it'd": "
                         "it'd've": "it would have", "it'll": "it will", "it'll've": "it will have","it's": "it is", "let's": "let us", "ma'am": "madam",
                         "mayn't": "may not", "might've": "might have","mightn't": "might not","mightn't've": "might not have", "must've": "must have",
                         "mustn't": "must not", "mustn't've": "must not have", "needn't": "need not", "needn't've": "need not have","o'clock": "of the clock
                         "oughtn't": "ought not", "oughtn't've": "ought not have", "shan't": "shall not", "sha'n't": "shall not", "shan't've": "shall not ha
                         "she'd": "she would", "she'd've": "she would have", "she'll": "she will", "she'll've": "she will have", "she's": "she is",
                         "should've": "should have", "shouldn't": "should not", "shouldn't've": "should not have", "so've": "so have","so's": "so as",
                         "this's": "this is","that'd": "that would", "that'd've": "that would have", "that's": "that is", "there'd": "there would",
                         "there'd've": "there would have", "there's": "there is", "here's": "here is","they'd": "they would", "they'd've": "they would have"
                         "they'll": "they will", "they'll've": "they will have", "they're": "they are", "they've": "they have", "to've": "to have",
                         "wasn't": "was not", "we'd": "we would", "we'd've": "we would have", "we'll": "we will", "we'll've": "we will have", "we're": "we a
                         "we've": "we have", "weren't": "were not", "what'll": "what will", "what'll've": "what will have", "what're": "what are",
                         "what's": "what is", "what've": "what have", "when's": "when is", "when've": "when have", "where'd": "where did", "where's": "where
                         "where've": "where have", "who'll": "who will", "who'll've": "who will have", "who's": "who is", "who've": "who have",
                         "why's": "why is", "why've": "why have", "will've": "will have", "won't": "will not", "won't've": "will not have",
                         "would've": "would have", "wouldn't": "would not", "wouldn't've": "would not have", "y'all": "you all",
                         "y'all'd": "you all would","y'all'd've": "you all would have","y'all're": "you all are","y'all've": "you all have",
                         "you'd": "you would", "you'd've": "you would have", "you'll": "you will", "you'll've": "you will have",
                         "you're": "you are", "you've": "you have"}
```

1.Convert everything to lowercase

2.Remove HTML tags

3.Contraction mapping

4.Remove (‘s)

5.Remove any text inside the parenthesis ( )

6.Eliminate punctuations and special characters

7.Remove stopwords

8.Remove short words

Let’s define the function:

Let's define the function:

```
[ ]  stop_words = set(stopwords.words('english'))

     def text_cleaner(text,num):
         newString = text.lower()
         newString = BeautifulSoup(newString, "lxml").text
         newString = re.sub(r'\([^)]*\)', '', newString)
         newString = re.sub('"','', newString)
         newString = ' '.join([contraction_mapping[t] if t in contraction_mapping else t for t in newString.split(" ")])
         newString = re.sub(r"'s\b","",newString)
         newString = re.sub("[^a-zA-Z]", " ", newString)
         newString = re.sub('[m]{2,}', 'mm', newString)
         if(num==0):
             tokens = [w for w in newString.split() if not w in stop_words]
         else:
             tokens=newString.split()
         long_words=[]
         for i in tokens:
             if len(i)>1:                                             #removing short word
                 long_words.append(i)
         return (" ".join(long_words)).strip()
```

```
[ ] #call the function
    cleaned_text = []
    for t in data['Text']:
        cleaned_text.append(text_cleaner(t,0))
```

```
[ ] cleaned_text[:5]
```

```
['bought several vitality canned dog food products found good quality product looks like stew processed meat smells better labrador finicky appreciates
product better',
 'product arrived labeled jumbo salted peanuts peanuts actually small sized unsalted sure error vendor intended represent product jumbo',
 'confection around centuries light pillowy citrus gelatin nuts case filberts cut tiny squares liberally coated powdered sugar tiny mouthful heaven chewy
flavorful highly recommend yummy treat familiar story lewis lion witch wardrobe treat seduces edmund selling brother sisters witch',
 'looking secret ingredient robitussin believe found got addition root beer extract ordered made cherry soda flavor medicinal',
 'great taffy great price wide assortment yummy taffy delivery quick taffy lover deal']
```

```
[ ] #call the function
    cleaned_summary = []
    for t in data['Summary']:
        cleaned_summary.append(text_cleaner(t,1))
```

```
[ ] cleaned_summary[:10]
```

```
['good quality dog food',
 'not as advertised',
 'delight says it all',
 'cough medicine',
 'great taffy',
 'nice taffy',
 'great just as good as the expensive brands',
 'wonderful tasty taffy',
 'yay barley',
 'healthy dog food']
```

```
[ ]  cleaned_summary[:10]
```

```
['good quality dog food',
 'not as advertised',
 'delight says it all',
 'cough medicine',
 'great taffy',
 'nice taffy',
 'great just as good as the expensive brands',
 'wonderful tasty taffy',
 'yay barley',
 'healthy dog food']
```

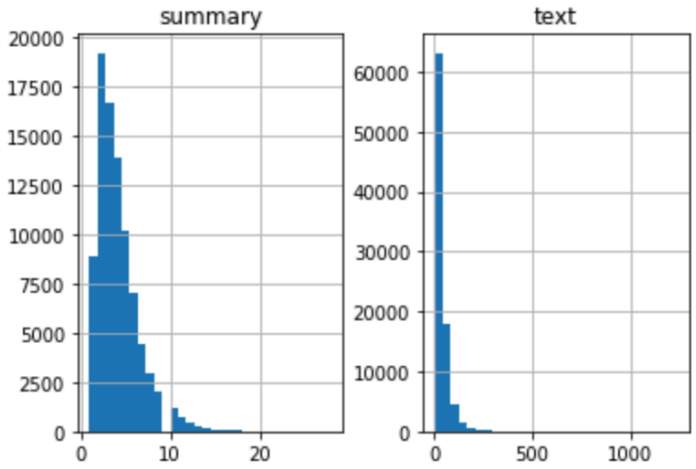
```
[ ]  data['cleaned_text']=cleaned_text
     data['cleaned_summary']=cleaned_summary
```

Drop empty rows

```
[ ]  data.replace('', np.nan, inplace=True)
     data.dropna(axis=0,inplace=True)
```

Understanding the distribution of the sequences

```
import matplotlib.pyplot as plt

text_word_count = []
summary_word_count = []

# populate the lists with sentence lengths
for i in data['cleaned_text']:
      text_word_count.append(len(i.split()))

for i in data['cleaned_summary']:
      summary_word_count.append(len(i.split()))

length_df = pd.DataFrame({'text':text_word_count, 'summary':summary_word_count})

length_df.hist(bins = 30)
plt.show()
```




```
[ ]  cnt=0
     for i in data['cleaned_summary']:
         if(len(i.split())<=8):
             cnt=cnt+1
     print(cnt/len(data['cleaned_summary']))
```

```
0.9424907471335922
```

```
[ ]  max_text_len=30
     max_summary_len=8
```

```
[ ]  cnt=0
     for i in data['cleaned_summary']:
         if(len(i.split())<=8):
             cnt=cnt+1
     print(cnt/len(data['cleaned_summary']))
```

```
0.9424907471335922
```

```
[ ]  max_text_len=30
     max_summary_len=8
```

Let us select the reviews and summaries whose length falls below or equal to **max_text_len** and **max_summary_len**

```
cleaned_text =np.array(data['cleaned_text'])
cleaned_summary=np.array(data['cleaned_summary'])

short_text=[]
short_summary=[]

for i in range(len(cleaned_text)):
    if(len(cleaned_summary[i].split())<=max_summary_len and len(cleaned_text[i].split())<=max_text_len):
        short_text.append(cleaned_text[i])
        short_summary.append(cleaned_summary[i])

df=pd.DataFrame({'text':short_text,'summary':short_summary})
```

```
[ ] df['summary'] = df['summary'].apply(lambda x : '<START>'+ x + ' <END>')
```

We are getting closer to the model building part. Before that, we need to split our dataset into a training and validation set. We'll use 90% of the dataset as the training data and evaluate the performance on the remaining 10% (holdout set):

```
[ ] from sklearn.model_selection import train_test_split
    x_tr,x_val,y_tr,y_val=train_test_split(np.array(df['text']),np.array(df['summary']),test_size=0.1,random_state=0,shuffle=True)
```

Preparing the Tokenizer

A tokenizer builds the vocabulary and converts a word sequence to an integer sequence.

Text Tokenizer

```
[ ] from keras.preprocessing.text import Tokenizer
    from keras.preprocessing.sequence import pad_sequences

    #prepare a tokenizer for reviews on training data
    x_tokenizer = Tokenizer()
    x_tokenizer.fit_on_texts(list(x_tr))
```

Rarewords and its Coverage

Here, I am defining the threshold to be 4 which means word whose count is below 4 is considered as a rare word

```
thresh=4

cnt=0
tot_cnt=0
freq=0
tot_freq=0

for key,value in x_tokenizer.word_counts.items():
    tot_cnt=tot_cnt+1
    tot_freq=tot_freq+value
    if(value<thresh):
        cnt=cnt+1
        freq=freq+value

print("% of rare words in vocabulary:",(cnt/tot_cnt)*100)
print("Total Coverage of rare words:",(freq/tot_freq)*100)
```

```
% of rare words in vocabulary: 66.12339930151339
Total Coverage of rare words: 2.953684513790566
```

Remember:

- **tot_cnt** gives the size of vocabulary (which means every unique words in the text)
- **cnt** gives me the no. of rare words whose count falls below threshold
- **tot_cnt - cnt** gives me the top most common words

```
[ ] #prepare a tokenizer for reviews on training data
    x_tokenizer = Tokenizer(num_words=tot_cnt-cnt)
    x_tokenizer.fit_on_texts(list(x_tr))

    #convert text sequences into integer sequences
    x_tr_seq    =   x_tokenizer.texts_to_sequences(x_tr)
    x_val_seq   =   x_tokenizer.texts_to_sequences(x_val)

    #padding zero upto maximum length
    x_tr    =   pad_sequences(x_tr_seq,  maxlen=max_text_len, padding='post')
    x_val   =   pad_sequences(x_val_seq, maxlen=max_text_len, padding='post')

    #size of vocabulary ( +1 for padding token)
    x_voc   =  x_tokenizer.num_words + 1
```

```
[ ] x_voc

    8440
```

Summary Tokenizer

```
[ ] #prepare a tokenizer for reviews on training data
    y_tokenizer = Tokenizer()
    y_tokenizer.fit_on_texts(list(y_tr))
```

```
[ ] #prepare a tokenizer for reviews on training data
    y_tokenizer = Tokenizer(num_words=tot_cnt-cnt)
    y_tokenizer.fit_on_texts(list(y_tr))

    #convert text sequences into integer sequences
    y_tr_seq    =   y_tokenizer.texts_to_sequences(y_tr)
    y_val_seq   =   y_tokenizer.texts_to_sequences(y_val)

    #padding zero upto maximum length
    y_tr    =   pad_sequences(y_tr_seq, maxlen=max_summary_len, padding='post')
    y_val   =   pad_sequences(y_val_seq, maxlen=max_summary_len, padding='post')

    #size of vocabulary
    y_voc  =   y_tokenizer.num_words +1
```

Let us check whether word count of start token is equal to length of the training data

```
y_tokenizer.word_counts['sostok'],len(y_tr)
```

```
(42453, 42453)
```

Here, I am deleting the rows that contain only **START** and **END** tokens

```
[ ]  ind=[]
     for i in range(len(y_tr)):
         cnt=0
         for j in y_tr[i]:
             if j!=0:
                 cnt=cnt+1
         if(cnt==2):
             ind.append(i)

     y_tr=np.delete(y_tr,ind, axis=0)
     x_tr=np.delete(x_tr,ind, axis=0)
```

```
     ind=[]
     for i in range(len(y_val)):
         cnt=0
         for j in y_val[i]:
             if j!=0:
                 cnt=cnt+1
         if(cnt==2):
             ind.append(i)

     y_val=np.delete(y_val,ind, axis=0)
     x_val=np.delete(x_val,ind, axis=0)
```

Model building

Return Sequences = True: When the return sequences parameter is set to True, LSTM produces the hidden state and cell state for every timestep

Return State = True: When return state = True, LSTM produces the hidden state and cell state of the last timestep only

Initial State: This is used to initialize the internal states of the LSTM for the first timestep

Stacked LSTM: Stacked LSTM has multiple layers of LSTM stacked on top of each other. This leads to a better representation of the sequence. I encourage you to experiment with the multiple layers of the LSTM stacked on top of each other

```
from keras import backend as K
K.clear_session()

latent_dim = 300
embedding_dim=100

# Encoder
encoder_inputs = Input(shape=(max_text_len,))

#embedding layer
enc_emb =  Embedding(x_voc, embedding_dim,trainable=True)(encoder_inputs)

#encoder lstm 1
encoder_lstm1 = LSTM(latent_dim,return_sequences=True,return_state=True,dropout=0.4,recurrent_dropout=0.4)
encoder_output1, state_h1, state_c1 = encoder_lstm1(enc_emb)

#encoder lstm 2
encoder_lstm2 = LSTM(latent_dim,return_sequences=True,return_state=True,dropout=0.4,recurrent_dropout=0.4)
encoder_output2, state_h2, state_c2 = encoder_lstm2(encoder_output1)

#encoder lstm 3
encoder_lstm3=LSTM(latent_dim, return_state=True, return_sequences=True,dropout=0.4,recurrent_dropout=0.4)
encoder_outputs, state_h, state_c= encoder_lstm3(encoder_output2)

# Set up the decoder, using `encoder_states` as initial state.
decoder_inputs = Input(shape=(None,))
```

```
#embedding layer
dec_emb_layer = Embedding(y_voc, embedding_dim,trainable=True)
dec_emb = dec_emb_layer(decoder_inputs)

decoder_lstm = LSTM(latent_dim, return_sequences=True, return_state=True,dropout=0.4,recurrent_dropout=0.2)
decoder_outputs,decoder_fwd_state, decoder_back_state = decoder_lstm(dec_emb,initial_state=[state_h, state_c])

# Attention layer
attn_layer = AttentionLayer(name='attention_layer')
attn_out, attn_states = attn_layer([encoder_outputs, decoder_outputs])

# Concat attention input and decoder LSTM output
decoder_concat_input = Concatenate(axis=-1, name='concat_layer')([decoder_outputs, attn_out])

#dense layer
decoder_dense =  TimeDistributed(Dense(y_voc, activation='softmax'))
decoder_outputs = decoder_dense(decoder_concat_input)

# Define the model
model = Model([encoder_inputs, decoder_inputs], decoder_outputs)

model.summary()
```

```
Layer (type)                 Output Shape              Param #     Connected to
==================================================================================================
input_1 (InputLayer)         (None, 30)                0
__________________________________________________________________________________________________
embedding (Embedding)        (None, 30, 100)           844000      input_1[0][0]
__________________________________________________________________________________________________
lstm (LSTM)                  [(None, 30, 300), (N 481200      embedding[0][0]
__________________________________________________________________________________________________
input_2 (InputLayer)         (None, None)              0
__________________________________________________________________________________________________
lstm_1 (LSTM)                [(None, 30, 300), (N 721200      lstm[0][0]
__________________________________________________________________________________________________
embedding_1 (Embedding)      (None, None, 100)         198900      input_2[0][0]
__________________________________________________________________________________________________
lstm_2 (LSTM)                [(None, 30, 300), (N 721200      lstm_1[0][0]
__________________________________________________________________________________________________
lstm_3 (LSTM)                [(None, None, 300),  481200      embedding_1[0][0]
                                                              lstm_2[0][1]
                                                              lstm_2[0][2]
__________________________________________________________________________________________________
attention_layer (AttentionLayer [(None, None, 300),  180300      lstm_2[0][0]
                                                              lstm_3[0][0]
__________________________________________________________________________________________________
concat_layer (Concatenate)   (None, None, 600)         0           lstm_3[0][0]
                                                              attention_layer[0][0]
__________________________________________________________________________________________________
time_distributed (TimeDistribut (None, None, 1989)    1195389     concat_layer[0][0]
==================================================================================================
Total params: 4,823,389
Trainable params: 4,823,389
Non-trainable params: 0
__________________________________________________________________________________________________
```

I am using sparse categorical cross-entropy as the loss function since it converts the integer sequence to a one-hot vector on the fly. This overcomes any memory issues.

```
model.compile(optimizer='rmsprop', loss='sparse_categorical_crossentropy')
```

```
[ ]  es = EarlyStopping(monitor='val_loss', mode='min', verbose=1,patience=2)
```

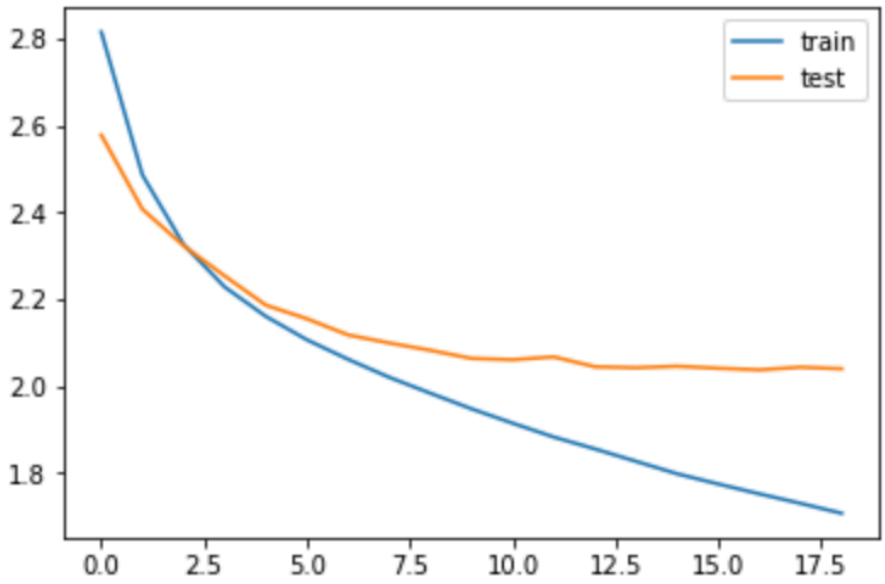
We'll train the model on a batch size of 128 and validate it on the holdout set (which is 10% of our dataset):

```
history=model.fit([x_tr,y_tr[:,:-1]], y_tr.reshape(y_tr.shape[0],y_tr.shape[1], 1)[:,1:] ,epochs=50,callbacks=[es],batch_size=128, validation_data=([x_val,y
```

```
Train on 41346 samples, validate on 4588 samples
WARNING:tensorflow:From /opt/conda/lib/python3.6/site-packages/tensorflow/python/ops/math_ops.py:3066: to_int32 (from tensorflow.python.ops.math_ops) is depr
Instructions for updating:
Use tf.cast instead.
Epoch 1/50
41346/41346 [==============================] - 85s 2ms/sample - loss: 2.8152 - val_loss: 2.5780
Epoch 2/50
41346/41346 [==============================] - 79s 2ms/sample - loss: 2.4859 - val_loss: 2.4072
Epoch 3/50
41346/41346 [==============================] - 81s 2ms/sample - loss: 2.3259 - val_loss: 2.3232
Epoch 4/50
41346/41346 [==============================] - 80s 2ms/sample - loss: 2.2281 - val_loss: 2.2534
Epoch 5/50
41346/41346 [==============================] - 79s 2ms/sample - loss: 2.1604 - val_loss: 2.1862
Epoch 6/50
41346/41346 [==============================] - 80s 2ms/sample - loss: 2.1065 - val_loss: 2.1549
Epoch 7/50
41346/41346 [==============================] - 80s 2ms/sample - loss: 2.0616 - val_loss: 2.1177
Epoch 8/50
41346/41346 [==============================] - 80s 2ms/sample - loss: 2.0202 - val_loss: 2.0992
Epoch 9/50
41346/41346 [==============================] - 79s 2ms/sample - loss: 1.9835 - val_loss: 2.0822
Epoch 10/50
```

Understanding the Diagnostic plot

```
from matplotlib import pyplot
pyplot.plot(history.history['loss'], label='train')
pyplot.plot(history.history['val_loss'], label='test')
pyplot.legend()
pyplot.show()
```



Inference

Set up the inference for the encoder and decoder:

```
# Encode the input sequence to get the feature vector
encoder_model = Model(inputs=encoder_inputs,outputs=[encoder_outputs, state_h, state_c])

# Decoder setup
# Below tensors will hold the states of the previous time step
decoder_state_input_h = Input(shape=(latent_dim,))
decoder_state_input_c = Input(shape=(latent_dim,))
decoder_hidden_state_input = Input(shape=(max_text_len,latent_dim))

# Get the embeddings of the decoder sequence
dec_emb2= dec_emb_layer(decoder_inputs)
# To predict the next word in the sequence, set the initial states to the states from the previous time step
decoder_outputs2, state_h2, state_c2 = decoder_lstm(dec_emb2, initial_state=[decoder_state_input_h, decoder_state_input_c])

#attention inference
attn_out_inf, attn_states_inf = attn_layer([decoder_hidden_state_input, decoder_outputs2])
decoder_inf_concat = Concatenate(axis=-1, name='concat')([decoder_outputs2, attn_out_inf])

# A dense softmax layer to generate prob dist. over the target vocabulary
decoder_outputs2 = decoder_dense(decoder_inf_concat)

# Final decoder model
decoder_model = Model(
    [decoder_inputs] + [decoder_hidden_state_input,decoder_state_input_h, decoder_state_input_c],
    [decoder_outputs2] + [state_h2, state_c2])
```

```
def decode_sequence(input_seq):
    # Encode the input as state vectors.
    e_out, e_h, e_c = encoder_model.predict(input_seq)

    # Generate empty target sequence of length 1.
    target_seq = np.zeros((1,1))

    # Populate the first word of target sequence with the start word.
    target_seq[0, 0] = target_word_index['sostok']

    stop_condition = False
    decoded_sentence = ''
    while not stop_condition:

        output_tokens, h, c = decoder_model.predict([target_seq] + [e_out, e_h, e_c])

        # Sample a token
        sampled_token_index = np.argmax(output_tokens[0, -1, :])
        sampled_token = reverse_target_word_index[sampled_token_index]

        if(sampled_token!='eostok'):
            decoded_sentence += ' '+sampled_token

        # Exit condition: either hit max length or find stop word.
        if (sampled_token == 'eostok'  or len(decoded_sentence.split()) >= (max_summary_len-1)):
            stop_condition = True

        # Update the target sequence (of length 1).
        target_seq = np.zeros((1,1))
        target_seq[0, 0] = sampled_token_index

        # Update internal states
        e_h, e_c = h, c

    return decoded_sentence
```

Here are a few summaries generated by the model:

```
for i in range(0,100):
    print("Review:",seq2text(x_tr[i]))
    print("Original summary:",seq2summary(y_tr[i]))
    print("Predicted summary:",decode_sequence(x_tr[i].reshape(1,max_text_len)))
    print("\n")
```

```
Review: gave caffeine shakes heart anxiety attack plus tastes unbelievably bad stick coffee tea soda thanks
Original summary: hour
Predicted summary:  not worth the money


Review: got great course good belgian chocolates better
Original summary: would like to give it stars but
Predicted summary:  good


Review: one best flavored coffees tried usually like flavored coffees one great serve company love
Original summary: delicious
Predicted summary:  great coffee


Review: salt separate area pain makes hard regulate salt putting like salt go ahead get product
Original summary: tastes ok packaging
Predicted summary:  salt
```

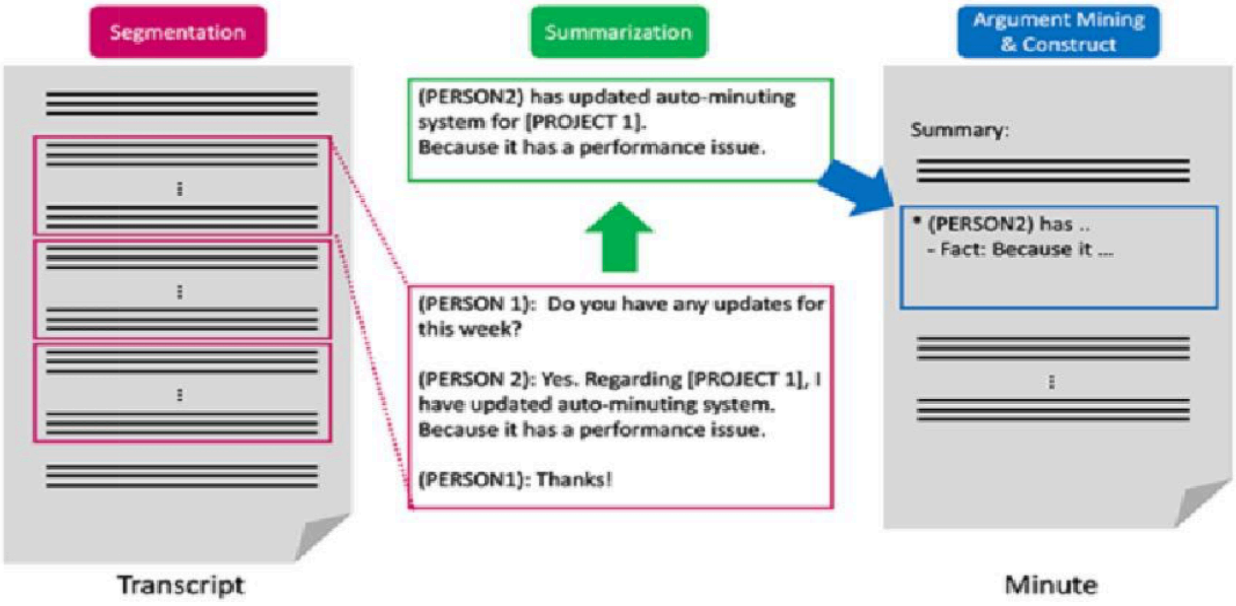
8. CONCLUSION

- In conclusion, SummaryPro is an innovative and powerful tool that uses the latest advancements in deep learning and NLP to transcribe and summarise audio files of business meetings. By combining extractive and abstractive summarization techniques, SummaryPro is able to provide a comprehensive and accurate summary of the meeting that captures the key information and concepts discussed. The tool also eliminates the need for manual minute-taking, reducing the risk of missing important information and improving the efficiency of the meeting management process.

- The development of SummaryPro was inspired by the AMBOC model, a well-established deep learning model for text summarization. However, our model has been modified and improved to suit the specific requirements of summarising business meetings. By using advanced techniques in NLP and deep learning, such as MFCC and concept clustering, SummaryPro is able to accurately identify speakers, extract key concepts, and generate a concise and informative summary in real-time.

- SummaryPro has the potential to revolutionize the process of meeting management by providing businesses with a reliable and efficient way to retain records of their meetings, reducing the chances of missing important information. Additionally, its ability to transcribe and summarise audio files in real-time can help increase productivity and reduce the cost of ineffective meetings, contributing to the growth of businesses.

- Overall, SummaryPro is a highly effective and efficient tool that can greatly improve the process of meeting management. Its advanced capabilities in deep learning and NLP make it a powerful solution for businesses looking to streamline their meeting processes and increase productivity.



9. BIBLIOGRAPHY

1) **Understanding LSTM's in depth**

2) **Learning MFCC**

3) **All about attention layer**

4) **Speech recognition through LSTM**