

Lip Reading Using Neural Networks and Deep Learning

A Major Project Report Submitted in partial fulfilment of the requirements for the award of the degree of

**BACHELOR OF
TECHNOLOGY IN
COMPUTER SCIENCE AND ENGINEERING**

Submitted by

| | |
|-----------------------------------|---------------------|
| Mr. Akash Chandra | (20071A0569) |
| Miss.Charitha Paruchuri | (20071A0570) |
| Miss. Awalgaonkar Karthika | (20071A0581) |
| Miss.Yogitha Pallamedi | (21075A0511) |

Under the Guidance of

Dr.K. Bheemalingappa
(Assistant Professor, Department of CSE, VNR VJIET)



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

**VALLURUPALLI NAGESWARA RAO VIGNANA JYOTHI INSTITUTE OF
ENGINEERING & TECHNOLOGY**

An Autonomous, ISO 9001:2015 & QS I-Gauge Diamond Rated Institute, Accredited by NAAC with 'A++' Grade
NBA Accreditation for B.Tech. CE,EEE,ME,ECE,CSE,EIE,IT,AME, M.Tech. STRE, PE, AMS, SWE Programmes
Approved by AICTE, New Delhi, Affiliated to JNTUH, NIRF (2023) Rank band:101-150 in Engineering Category
College with Potential for Excellence by UGC,JNTUH-Recognized Research Centres:CE,EEE,ME,ECE,CSE

Vignana Jyothi Nagar, Pragathi Nagar, Nizampet (S.O.), Hyderabad – 500 090, TS, India.

April, 2024

**VALLURUPALLI NAGESWARA RAO VIGNANA JYOTHI INSTITUTE OF
ENGINEERING and TECHNOLOGY**

An Autonomous, ISO 9001:2015 & QS I-Gauge Diamond Rated Institute, Accredited by NAAC with 'A++' Grade
NBA Accreditation for B.Tech. CE,EEE,ME,ECE,CSE,EIE,IT,AME, M.Tech. STRE, PE, AMS, SWE Programmes
Approved by AICTE, New Delhi, Affiliated to JNTUH, NIRF (2023) Rank band:101-150 in Engineering Category
College with Potential for Excellence by UGC,JNTUH-Recognized Research Centres:CE,EEE,ME,ECE,CSE

Vignana Jyothi Nagar, Pragathi Nagar, Nizampet (S.O.), Hyderabad – 500 090, TS, India.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING



CERTIFICATE

This is to certify that the project report entitled “ **Implementation of Lip Reading using Neural Networks and Deep Learning**” is a bonafide work done under our supervision and is being submitted by **Mr.Akash Chandra(20071A069)**, **Miss.Charitha Paruchuri (20071A0570)**, **Miss. Karthika Awalgoankar (20071A0581)**, **Miss.Yogitha Pallamedi (21075A0511)** in partial fulfilment for the award of the degree of **Bachelor of Technology** in Computer Science and Engineering, of the VNRVJIET, Hyderabad during the academic year 2023- 2024.Certified further that to the best of our knowledge the work presented in this thesis has not been submitted to any other University or Institute for the award of any Degree or Diploma.

Dr.K. Bheemalingappa
Assistant Professor & Internal Guide
CSE Department, VNR VJIET

Dr. S Nagini
Professor & HOD
CSE & CSBS Department, VNR VJIET

**VALLURUPALLI NAGESWARA RAO VIGNANA JYOTHI INSTITUTE OF
ENGINEERING and TECHNOLOGY**

An Autonomous, ISO 9001:2015 & QS I-Gauge Diamond Rated Institute, Accredited by NAAC with 'A++' Grade
NBA Accreditation for B.Tech. CE,EEE,ME,ECE,CSE,EIE,IT,AME, M.Tech. STRE, PE, AMS, SWE Programmes
Approved by AICTE, New Delhi, Affiliated to JNTUH, NIRF (2023) Rank band:101-150 in Engineering Category
College with Potential for Excellence by UGC,JNTUH-Recognized Research Centres:CE,EEE,ME,ECE,CSE

Vignana Jyothi Nagar, Pragathi Nagar, Nizampet (S.O.), Hyderabad – 500 090, TS, India.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING



DECLARATION

We declare that the major project work entitled “**Implementation of Lip Reading Using Neural Networks and Deep Learning**” submitted in the department of Computer Science and Engineering, Vallurupalli Nageswara Rao Vignana Jyothi Institute of Engineering and Technology, Hyderabad, in partial fulfilment of the requirement for the award of the degree of **Bachelor of Technology** in **Computer Science and Engineering** is a bonafide record of our own work carried out under the supervision of **Dr. K. Bheemalingappa, Assistant Professor, Department of CSE, VNRVJIET**. Also, we declare that the matter embodied in this thesis has not been submitted by us in full or in any part thereof for the award of any degree/diploma of any other institution or university previously.

**Akash
Chandra**
(20071A0569)

**Charitha
Paruchuri**
(20071A0570)

**Karthika
Awalgaonkar**
(20071A0581)

**Yogitha
Pallamedi**
(21075A0511)

ACKNOWLEDGEMENT

Firstly, we would like to express our immense gratitude towards our institution VNR Vignana Jyothi Institute of Engineering and Technology, which created a great platform to attain profound technical skills in the field of Computer Science, thereby fulfilling our most cherished goal.

We are very much thankful to our Principal, **Dr. C.D Naidu** Head of Department, **Dr.S. Nagini**, for extending their cooperation in doing this project in stipulated time.

We extend our heartfelt thanks to our guide, **Dr.K. Bheemalingappa** and the project coordinators **Dr. V. Baby** and **Dr. P.Anjusha**, for their enthusiastic guidance throughout the course of our project.

Last but not least, our appreciable obligation also goes to all the staff members of Computer Science & Engineering department and to our fellow classmates who directly or indirectly helped us.

Mr. Akash Chandra (20071A0569)
Miss. Charitha Paruchuri (20071A0570)
Miss. Karthika Awalgoankar (20071A0581)
Miss. Yogitha Pallamedi (21075A0511)

Abstract

Lip reading is a technique to understand words or speech by visual interpretation of face, mouth, and lip movement without the involvement of audio. With the variety of languages spoken around the world, there is difference in diction and relative articulation of words and phrases, so it becomes substantially challenging to create a computer program that automatically and accurately reads the spoken words solely based on the visual lip movement of the speaker. Traditionally this process been approached in two stages : a visual feature design followed by prediction. Recent advancements in deep learning have enabled end-to-end trainable models for lipreading. However, existing end-to-end models primarily focus on word classification rather than sentence-level sequence prediction. Human lipreading performance suggests the significance of capturing temporal context for understanding longer words in an ambiguous communication channel, motivated by this a model is developed that maps variable-length sequences of video frames to text using spatiotemporal convolutions. The trained lip-reading models were evaluated on various datasets and based on their accuracy to predict words the best performing model was implemented for real-time prediction.

TABLE OF CONTENTS

| | |
|--|----|
| CHAPTER 1 | |
| INTRODUCTION | |
| 1.1 Introduction | 1 |
| 1.2 Problem Statement | 1 |
| 1.3 Objective | 1 |
| 1.4 Scope for the Work | 2 |
| CHAPTER 2 | |
| LITERATURE SURVEY | |
| 2.1. Feasibility study | 4 |
| 2.1.2 Organizational feasibility | 4 |
| 2.1.3 Economy feasibility | 4 |
| 2.1.4 Technical feasibility | 5 |
| 2.1.5 Behavioral feasibility | 5 |
| 2.2 Findings from existing Review Papers | 6 |
| 2.3 Existing system | 19 |
| CHAPTER 3 | |
| SOFTWARE REQUIREMENT ANALYSIS | |
| 3.1 Introduction | 21 |
| 3.1.1 Document purpose | 21 |
| 3.1.2 Definitions | 21 |
| 3.2 System architecture | 23 |
| 3.3 Functional Requirements | 24 |
| 3.4 System analysis | 24 |
| 3.5 Non-functional Requirements | 27 |
| 3.6 Software requirement specification | 27 |
| CHAPTER 4 | |
| PROPOSED SYSTEM | |
| 4.1 Methodology | 30 |
| 4.2 System architecture | 30 |
| 4.3 Steps involved | 31 |
| 4.4 Modules | 32 |
| 4.5 Algorithms used | 34 |
| CHAPTER 5 | |
| SOFTWARE DESIGN | |
| 5.1 UML Diagrams | 37 |
| 5.2 Use Case Diagram | 38 |
| 5.3 Sequence Diagram | 43 |
| 5.4 Activity Diagram | 48 |
| 5.5 Class Diagram | 50 |
| CHAPTER 6 | |
| IMPLEMENTATION | |
| 6.1 Front -end | 53 |
| 6.2 Back-end | 54 |
| CHAPTER 7 | |
| TESTING | |
| 7.1 Types of testing | 68 |

| | |
|--------------------------------------|----|
| 7.2 Software testing methods | 69 |
| 7.3 Testing levels | 71 |
| 7.4 Testing Results | 74 |
| CHAPTER 8 | |
| OUTPUT SCREENS / RESULTS | 77 |
| CHAPTER 9 | |
| CONCLUSION & FURTHER WORK | |
| 9.1 Conclusion | 82 |
| 9.2 Future Scope | 83 |
| References | |

List of figures

| Figure | Page no |
|---|----------------|
| Fig 4.1 System Architecture | 30 |
| Fig 4.2 Layers of the model | 34 |
| Fig 5.1 Use Case Diagram | 42 |
| Fig 5.2 Sequence Diagram | 45 |
| Fig 5.3 Activity Diagram | 49 |
| Fig 5.4 Class Diagram | 52 |
| Fig 6.1 Downloading the datasets | 54 |
| Fig 6.2 Creating the model | 55 |
| Fig 6.3 Importing the libraries | 55 |
| Fig 6.4 Loading the video | 58 |
| Fig 6.5 Creating Tensor flow datasets | 59 |
| Fig 6.6 Training the model | 61 |
| Fig 6.7 A function for cropping the mouth ROI | 64 |
| Fig 6.8 Function for detection of lips | 64 |
| Fig 6.9 Function for predicting video | 65 |
| Fig 6.10 Predictions using rouge score | 67 |
| Fig 7.1 Black Box Testing | 69 |
| Fig 7.2 Grey Box Testing | 70 |
| Fig 7.3 White Box Testing | 71 |
| Fig 7.4 Test case scenarios for lip reading | 74 |
| Fig 7.4.1 Test case for checking functionality of web application | 74 |
| Fig 7.4.2 Test case for checking the functionality of webcam while uploading the videos | 75 |
| Fig 7.4.3 Test case for checking the samples from the dataset | 75 |
| Fig 7.4.4 Test case for checking the functionality of clear button | 76 |
| Fig 7.4.5 Test case for checking the functionality of submit button | 76 |

CHAPTER 1

1.1 INTRODUCTION

Lip reading, the interpretative process of spoken language through the observation of lip movements, has emerged as a prominent focus of research, largely propelled by the availability of extensive datasets such as the GRID corpus. Deciphering spoken language through lip movements, known as machine lipreading, poses a considerable challenge due to the intricate process of extracting spatiotemporal features from video data. Recent strides in deep learning techniques have opened up promising avenues to tackle this challenge. Our work introduces an innovative end-to-end lipreading model operating at the sentence level, aiming to overcome the limitations of prior methods. A comparative analysis of model's performance against that of hearing-impaired individuals undertaking the same lipreading task. The findings reveal a marked enhancement in accuracy, indicating its potential to better aid individuals with hearing impairments in comprehending spoken language.

1.2 Problem Statement

Lip reading is a method employed to comprehend spoken language through visual observation of facial movements, particularly those of the face, mouth, and lips, without reliance on audio input. The complexity arises due to the diverse diction and articulation styles individuals employ in speech. Consequently, developing a computerized system capable of accurately transcribing spoken words based solely on visual lip movement poses a significant challenge. Even proficient lip readers can only approximate about every other word. This project aims to devise an automated lip-reading system trained on diverse datasets to yield efficient outcomes.

1.3 Objective

The main objectives are Firstly, it aims to provide assistance for individuals with hearing impairments by offering real-time transcription of spoken words into text, thereby enhancing their ability to comprehend verbal communication. The project also seeks to facilitate robust communication in noisy environments where traditional audio-based communication may be hindered, by leveraging lipreading technology to interpret visual cues. Thirdly, it aims to achieve multimodal integration by combining lip movements with audio signals using deep learning

techniques, thereby creating a more resilient and accurate system for communication. Finally we aim to contribute to improved speech recognition by training deep learning models to interpret lip movements and predict spoken words, thus advancing the capabilities of automated speech recognition systems.

1.4 Scope for the Work

- **Data Collection and Annotation:** Gathering a diverse dataset of videos containing people speaking various languages, with different accents, and in different lighting and background conditions. This dataset would need to be annotated with corresponding transcriptions of the spoken words.
- **Preprocessing and Data Augmentation:** Preprocessing the videos to extract the region of interest containing the speaker's mouth, normalizing lighting conditions, and augmenting the dataset to increase its size and diversity. This might involve techniques such as frame mirroring, frame cropping, and frame rotation.
- **Model Architecture Design:** Designing the neural network architecture for lip reading, which typically involves a combination of convolutional neural networks (CNNs) for feature extraction from video frames, recurrent neural networks (RNNs) or transformers for temporal modeling, and connectionist temporal classification (CTC) or attention mechanisms for sequence prediction.
- **Model Training:** Training the designed model using the annotated dataset. This involves optimizing the model's parameters to minimize a loss function (e.g., CTC loss or cross-entropy loss) using gradient-based optimization algorithms (e.g., stochastic gradient descent or Adam).

- **Model Evaluation:** Evaluating the trained model on a separate test dataset to assess its performance in accurately transcribing spoken words from lip movements. Evaluation metrics might include word error rate (WER), character error rate (CER), or sentence-level accuracy.
- **Fine-tuning and Optimization:** Iteratively fine-tuning the model architecture and hyperparameters based on evaluation results to improve performance. This might involve experimenting with different network architectures, activation functions, optimizer settings, and regularization techniques.

CHAPTER 2

2.1 LITERATURE SURVEY

2.1.1 FEASIBILITY STUDY

Despite the challenges, the advancements in deep learning, the growing market demand, and the potential economic benefits make lip reading with neural networks a highly feasible project. By addressing data acquisition challenges, improving robustness to variations, and optimizing real-time performance, deep learning-based lip reading systems have the potential to revolutionize communication accessibility.

2.1.2 ORGANIZATIONAL FEASIBILITY

Ensuring compliance with regulatory standards is crucial for the implementation of the lip reading technology. Adherence to regulations regarding patient data privacy and security, as well as medical device regulations, is essential. Regulatory compliance safeguards patient confidentiality and fosters trust in the technology among healthcare organizations and consumers.

2.1.3 ECONOMY FEASIBILITY

By leveraging these technologies, advancements in accuracy and efficiency have been achieved, enhancing the practicality of applications in various sectors such as accessibility, security, and communication. Despite initial investment costs in research, development, and infrastructure, the potential long-term benefits, including improved accessibility for the hearing impaired, enhanced security measures, and streamlined communication systems, make it a viable economic option. Additionally, as the technology matures and becomes more widespread, economies of scale may further drive down costs, increasing its affordability and accessibility.

2.1.4 TECHNICAL FEASIBILITY

The Visual Studio Code Python Programming Language, Google collab are needed to be installed in the system to build this project. A minimum of 4GB ram is required to run all this software smoothly. In order to build and run the project, necessary modules and packages must be installed. For the programs to be downloaded and installed, an internet connection is required. The user can download and install these packages and can utilize them with ease. Thus, this project is technically feasible.

2.1.5 BEHAVIOURAL FEASIBILITY

The app is behaviorally possible because it does not require any technical guidance to use the project. The user can enter the required parameters' values and get the result instantly. The behavior of the project is very simple to use. Thus, the project is behaviorally feasible.

2.2 Findings from existing papers

(i) The paper “Lip Reading Sentences Using Deep Learning With Only Visual Cues” authored by Daoqing Chen proposes a new method for improving the accuracy of lip reading sentences in silent videos. Unlike previous methods that focused on recognizing words or letters, this method focuses on recognizing visemes, which are a small set of distinct lip shapes. This approach allows the system to handle a wider range of vocabulary, including words not seen during training. The system first uses a deep learning model to classify visemes in the video. Then, it uses perplexity analysis to convert the recognized visemes into words. This method achieved a significant improvement in word classification accuracy compared to previous methods. However, there is still room for improvement in the viseme-to-word conversion stage. Future research will focus on finding a more efficient way to convert visemes to words conversion stage. Future research will focus on finding a more efficient way to convert visemes to words

(ii) This paper “Deep Learning -based Automated Lip-Reading: A Survey” surveys advancements in automated lip-reading systems between 2007 and 2021. Lip-reading systems can now decode full sentences thanks to deep learning and larger datasets like BBC-LRS2. These datasets include thousands of words spoken by various people in different conditions. Lip-reading systems involve feature extraction and classification. Most systems use 2D+3D Convolutional Neural Networks (CNNs) for feature extraction as they can learn both spatial and temporal information from videos.

For classification, Recurrent Neural Networks (RNNs) like LSTMs were dominant, but recently, Transformers and Temporal Convolutional Networks (TCNs) are gaining traction due to their faster training and ability to handle long-term dependencies. Classification approaches have also evolved. Earlier systems classified single words, while later ones used ASCII characters to predict sentences with vast vocabularies. Ideally, using phonemes or visemes (lip shapes) could allow lip-reading systems to predict unseen words, even those not included in the training data. However, challenges remain. Lip-reading systems struggle to predict unseen words and visually ambiguous words (words that look the same when spoken). Additionally, generalizability remains an issue - systems may not perform well on speakers, resolutions, or frame rates not seen during training

(iii) This paper “Lip-Reading Driven Deep Learning Approach for Speech Enhancement” explores the advancements in automated lip-reading systems between 2007 and 2021. Fueled by breakthroughs in deep neural networks and extensive datasets, these systems have evolved significantly. Traditionally, lip-reading systems focused on recognizing isolated speech units like digits and letters. However, with the advent of deep learning and larger datasets like BBC-LRS2, these systems can now decode entire sentences. These datasets incorporate variations in speaker pose, lighting, and resolution for thousands of speakers uttering thousands of words. Lip-reading systems typically involve two main processes: feature extraction and classification. Feature extraction involves analyzing video frames to identify relevant characteristics of lip movements. In this regard, 2D+3D Convolutional Neural Networks (CNNs) have become the preferred choice due to their ability to learn both spatial and temporal (movement) information from videos. Classification involves assigning a meaning to the extracted features. Previously, Recurrent Neural Networks (RNNs) like LSTMs dominated this task. However, recently, Transformers and Temporal Convolutional Networks (TCNs) are gaining traction due to their faster training and ability to handle long-term dependencies within speech patterns.

Classification approaches have also undergone a transformation. Earlier systems classified single words, while later ones employed ASCII characters to predict sentences with vast vocabularies. Ideally, using phonemes (individual speech sounds) or visemes (lip shapes) could allow lip-reading systems to predict unseen words, even those not included in the training data. Despite these advancements, challenges remain. Lip-reading systems still struggle to predict unseen words and visually ambiguous words. Additionally, generalizability remains an issue - systems may not perform well on speakers, resolutions, or frame rates not encountered during training. In conclusion, this paper highlights the significant progress made in automated lip-reading systems in recent years. Deep learning and larger datasets have enabled these systems to move from recognizing basic units to decoding full sentences. However, challenges persist, and further research is needed to address issues like unseen words, visual ambiguities, and generalizability.

(iv) In their paper titled "Automatic Lip Reading with Convolutional Neural Networks and Bidirectional Long Short-Term Memory Networks," authors Yuanyao Lu and Jie Yan introduce a novel method for automatic lip reading. Traditionally, automatic lip reading systems undergo two stages: feature extraction and classification. While conventional methods rely on hand-crafted

feature extraction techniques, modern approaches leverage deep learning methodologies. The proposed system by Yuanyao Lu employs a process wherein key frames are initially extracted from a video, and the mouth region within these frames is identified. Subsequently, a Convolutional Neural Network (CNN) is utilized to extract features from the mouth images. Following this, a Bidirectional Long Short-Term Memory (BiLSTM) network is employed to capture sequential information between the features extracted from each frame. A softmax layer is then applied for classification, predicting the spoken word in the video. The authors assert several advantages of their method: the CNN adeptly handles variations in mouth image due to translation, rotation, and distortion, while the BiLSTM network effectively learns long-term dependencies between features. Experimental results demonstrate the superiority of the proposed deep learning approach over traditional methods combining hand-crafted features with classification models. Testing conducted on a dataset indicates potential for future work, focusing on the development of a speaker-independent system trainable on a larger and more diverse dataset.

(v) This survey on the paper “Deep Neural Network Automated Lip Reading Visual Speech Recognition” explores the potential of utilizing deep learning methods for lip reading to assist individuals with hearing impairment or muteness. It delves into the causes of muteness, such as medical conditions like laryngectomy, and different types of hearing loss. Both conditions have a significant impact on communication. The paper suggests that lip reading can be an effective solution for those facing these challenges, enabling them to understand speech visually and participate in social interactions. It also highlights the benefits of lip reading in improving speech perception, particularly in noisy environments, leading to increased interest in automated lip-reading systems. Overall, the paper concludes that lip-reading technology holds promise for individuals with hearing loss or muteness. Deep learning techniques offer potential for developing automatic lip-reading systems to enhance communication accessibility, with future research aiming to enhance accuracy and broaden applications

(vi) In their paper titled "Artificial Intelligence: A Survey On Lip-Reading Techniques," authored by Sameep Bagadia and Amit Garg, the potential of deep learning in automatic lip-reading systems is explored. Lip-reading entails visually interpreting speech through lip movements observation. The typical stages of lip-reading technology encompass face detection, lip localization, feature extraction, and recognition. These systems can be trained on datasets comprising videos and their corresponding transcripts. Deep learning, a branch of artificial intelligence inspired by human

cognition, has gained prominence in lip-reading systems. This survey delves into various lip-reading techniques and language datasets embraced in the deep learning era. A key challenge in lip-reading is language variability, demanding considerable concentration even for humans. The paper underscores the potential advantages of lip-reading technology in biometric security, owing to the difficulty in replicating lip movements. While traditional lip-reading techniques relied on image processing, machine learning, and artificial intelligence, deep learning introduces a more sophisticated paradigm where computers learn from data to make predictions. Deep learning models, trained on extensive video datasets paired with transcripts, excel in discerning patterns and predicting spoken words. Lip-reading datasets may encompass both audio and visual information. Deep learning is already leveraged in applications such as face recognition, speech recognition, and natural language processing, with popular libraries like Keras, PyTorch, and TensorFlow commonly employed for these tasks. The paper delves into the integration of deep learning models with audio and visual data to enhance speech recognition, particularly in noisy environments. Researchers constructing such systems must consider facets like face detection, feature extraction, deep learning models, and appropriate datasets. The paper concludes that deep learning, coupled with Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) networks, has significantly enhanced performance in lip-reading systems. Furthermore, the paper discusses the availability of lip-reading datasets in languages like English, French, German, and Japanese, stressing the importance of datasets in regional Indian languages to foster inclusivity in this technology.

vii) The paper “Lip Reading Using CNN and LSTM” investigates methods for lip-reading using video data only. The goal is to predict words or phrases spoken in a video clip without any audio information. They employed a pre-trained deep learning model called VGGNet to analyze sequences of facial images. VGGNet was originally trained on celebrity faces from various sources. Here, the researchers experimented with feeding VGGNet with concatenated images (combining multiple frames into a single image) and using Long Short-Term Memory (LSTM) networks to capture temporal information from the video sequences. While LSTM models underperformed, concatenating image frames achieved promising results, reaching an accuracy of 76% on a validation dataset. This method outperformed others likely because it utilized more data and wasn't affected by variations in speaking speed. The paper acknowledges limitations. The dataset used for training was relatively small, and the LSTM models suffered from long training times. Future work discussed includes exploring more advanced architectures like volumetric

convolutions and incorporating depth information from the videos. The paper also suggests investigating recurrent kernels and optical flow as potential features for improved lip-reading systems.

(viii) This paper “Discriminative Multi-modality Speech Recognition “ authored by Nando de Freitas, and Brendan Shillingford proposes a new method for speech recognition that combines audio and video information (lip movements) to improve accuracy, especially in noisy environments. Existing multi-modal speech recognition systems typically treat audio and video equally. The researchers propose a two-stage system. First, a stage removes noise from the audio stream by analyzing the corresponding lip movements in the video. The cleaned audio is then combined with the video information again for improved speech recognition. To achieve this, the researchers developed a deep learning model named Audio-Enhanced Multi-Modality Speech Recognition (AE-MSR). AE-MSR includes several new components: a special visual processing unit to extract better features from lip movements, a more suitable network for handling temporal information in the video, and a new recurrent unit for combining audio and visual data. Experiments showed that AE-MSR outperformed existing state-of-the-art methods on multiple datasets, including noisy environments. This method is particularly useful for people with hearing impairments or in situations where speech is difficult to hear due to background noise

(ix) In the paper titled "Visual Speech Recognition for Multiple Languages in the Wild," the authors argue that alongside large datasets, model design plays a crucial role in Visual Speech Recognition (VSR). They propose a novel VSR model incorporating prediction-based auxiliary tasks, alongside hyperparameter optimization and data augmentation techniques. Despite being trained on smaller publicly available datasets compared to previous works, their model achieves state-of-the-art performance on multiple benchmarks. Traditionally, VSR models faced limitations such as a restricted vocabulary and inability to operate outside controlled environments due to reliance on hand-crafted visual features and the lack of large training datasets. However, recent advancements in deep learning techniques and the availability of large audio-visual datasets have led to the development of more accurate and robust VSR models. While some prior works used non-public datasets containing significantly more data, this paper demonstrates that carefully designed models can achieve superior performance even with smaller datasets.

The researchers' approach encompasses three key elements: integrating auxiliary prediction tasks into the VSR model, applying suitable data augmentation techniques, and optimizing hyperparameters within an existing model architecture. This strategy substantially reduces word error rate and attains state-of-the-art performance across various benchmarks. Language independence is another focal point of the research. While most existing VSR models are tailored for English, the proposed model achieves state-of-the-art performance not only in English but also in Mandarin, Spanish, French, Italian, and Portuguese. Overall, the paper underscores the significance of model design in conjunction with large datasets for achieving superior VSR performance. The proposed model surpasses all existing methods trained on publicly available data across multiple languages, showcasing the potential of combining meticulous model design with established deep learning techniques for significant advancements in VSR technology.

(x) The paper titled "Training Strategies for Improved Lip-Reading" delves into methodologies aimed at enhancing lip-reading proficiency in isolating words from videos. Researchers expanded upon existing techniques and evaluated their efficacy through a series of experiments. Typically, lip-reading models comprise three components: a visual encoder for video processing, a temporal model for analyzing visual sequence, and a classification layer for identifying the spoken word. Various approaches for each component were compared. Notably, Densely-Connected Temporal Convolutional Networks (DC-TCNs) emerged as the most effective temporal model, surpassing Bidirectional Gated Recurrent Units (BGRUs) and Multi-Scale Temporal Convolutional Networks (MS-TCNs).

Furthermore, data augmentation techniques were explored to bolster model performance. Among these, Time Masking—randomly masking parts of the video sequence—proved most efficacious, alongside Mixup, which blends pairs of training examples. Incorporating word boundary indicators in training data and employing self-distillation for knowledge transfer between models yielded further improvements, albeit to a lesser extent. Through the amalgamation of these techniques, researchers achieved a new state-of-the-art accuracy of 93.4% on a standard lip-reading dataset (LRW), marking a notable 4.6% absolute improvement over previous methods. Additional gains, elevating performance to 94.1%, were attained by pre-training the model on supplementary datasets. An error analysis revealed that these advancements primarily stemmed

from enhanced accuracy in recognizing challenging words. Overall, the paper underscores the efficacy of meticulous model design coupled with established deep learning techniques in substantially enhancing lip-reading technology.

(xi) The paper “Leveraging Unimodal Self-Supervised Learning for Multimodal Audio-Visual Speech Recognition” authored by Yichen Gong, Helong Zhou addresses the challenge of training Transformer-based models for audio-visual speech recognition (AVSR) due to the high data demands and cost of obtaining aligned and labeled multimodal data. The authors propose leveraging unimodal self-supervised learning on large-scale unimodal datasets to enhance the performance of AVSR. Specifically, they train audio and visual front-ends on unimodal datasets and integrate components of both front-ends into a larger multimodal framework for AVSR. The framework learns to recognize parallel audio-visual data through a combination of CTC and seq2seq decoding. Results show that both components inherited from unimodal self-supervised learning cooperate well, leading to competitive results through fine-tuning.

The proposed model achieves a significant improvement in performance on the widely accepted Lip Reading Sentences 2 (LRS2) dataset, with a relative improvement of 30% over the current state-of-the-art, even without an external language model. The approach significantly outperforms previous AVSR models in both word-level and sentence-level tasks. Existing AVSR models often use extra data or supervised learning stages to enhance performance, but the proposed approach shows that leveraging unimodal self-supervised learning can be an effective alternative. By using pre-trained models trained on unimodal data, the authors achieve state-of-the-art performance in AVSR without the need for additional supervised learning stages. The study highlights the effectiveness of self-supervised learning in AVSR and suggests directions for future work, such as exploring temporal correlations within the visual domain and cross-modal correlations between audio and visual modalities. Overall, the findings demonstrate the potential of self-supervised learning in improving AVSR performance, especially when labeled multimodal data is limited or costly to obtain.

(xii) The paper “Lip Reading Using Neural network and Deep learning “presents a study on automated lip-reading using machine learning, specifically deep learning and neural networks. Lip reading is challenging due to variations in speech articulation and diction among individuals. The project focuses on training and evaluating two separate Convolutional Neural Network (CNN) architectures on a subset of a dataset to predict words from lip movements. The best performing

model is then implemented in a web application for real-time word prediction. The study highlights the potential benefits of automated lip-reading technology, including improved speech recognition in noisy environments, advancements in hearing aid systems for people with hearing disabilities, and applications in security for speech analysis when audio is corrupted or absent in videos. The project utilizes a Haar Feature-Based Cascade classifier to detect and track the mouth region in input speaker videos, which is then used to preprocess the data before training the neural network models. The classifier is trained using positive images (frontal face with visible mouth) and negative images (non-frontal face) to detect specific features (ROI) in the image related to the mouth region. The results show that the lightweight model architecture outperformed the EF-3 architecture. Model A1, A2, B, C, and D were built on top of the lightweight model architecture, with Model D achieving the highest accuracy and best performance. This model was implemented in the web application for real-time lip-reading. For future work, the authors suggest training the entire dataset for a more robust lip-reading system, training the lightweight model architecture on higher computer architectures or utilizing parallel computation for higher accuracy, and training the model to lip-read sentences.

(xiii) The paper titled "Visual Speech Recognition: A Deep Learning Approach" explores the utilization of deep learning, specifically employing a ResNet architecture with 3D convolution layers as the encoder and Gated Recurrent Units (GRU) as the decoder, for visual speech recognition (VSR) or lip-reading. The objective is to enable machines to comprehend human speech solely based on visual cues, a task previously deemed unattainable. The study highlights the challenge, revealing that experts in visual speech recognition can only deduce about 3-4% of spoken words through lip-reading from videos lacking audio, underscoring the intricacy of the task. The proposed approach demonstrates promising outcomes, achieving 90% accuracy on the BBC dataset and 88% on a custom video dataset. Operating at the word level, the model exhibits potential for extension to short phrases or sentences. It capitalizes on ResNet architecture for spatial-temporal feature extraction and GRU networks for precise labeling, prioritizing ease of training and computational efficiency in its design.

Furthermore, the study underscores the significance of VSR in developing supportive technologies for speech recognition in noisy environments. Despite advancements, lip-reading remains challenging due to various factors such as accents, skin color, speech pace, and facial attributes, particularly in contexts devoid of context. Future enhancements may entail training the model on datasets featuring greater diversity and instances, including videos of individuals with facial hair

to bolster performance in such scenarios. The model, deployed on a local server for testing with diverse video inputs, holds potential for further training to expand its vocabulary. The findings underscore the promise of deep learning in propelling visual speech recognition technology forward.

(xiv) The paper authored by Edwin J.C titled "Lip Reading Using Facial Feature Extraction and Deep Learning" delves into the challenges and solutions associated with lip reading, a technique employed by the hearing impaired to comprehend speech via visual cues. It sheds light on the prevalence of hearing impairment and the constraints of sign language, underscoring the necessity for alternative communication modalities. The paper accentuates the promise of deep learning, particularly Long-Short Term Memory (LSTM) networks, in enhancing lip reading accuracy. The proposed methodology integrates deep learning with facial feature extraction to bolster lip reading efficacy. Image processing techniques, encompassing color imaging and depth sensing, are harnessed to refine classifier precision. Furthermore, a Facial Expression Recognition algorithm aids in identifying facial movements, facilitating lip movement tracking. Despite advancements, lip reading confronts challenges such as coarticulation, homophones, and human error. The paper posits that these hurdles can be alleviated through the amalgamation of deep neural networks with facial feature extraction. The envisioned method aims to elevate lip reading accuracy, rendering it a more dependable communication tool for the hearing impaired. Overall, the paper underscores the potential of deep learning and image processing in advancing lip reading technology. By tackling the shortcomings of current methodologies, the proposed approach endeavors to enhance the accessibility and effectiveness of lip reading for the hearing impaired.

(xv) In this paper titled "Lip Reading Using Temporal Convolution Networks," the authors tackle the challenges in visual speech recognition, particularly in isolated word recognition through lip-reading, by proposing enhancements to the current state-of-the-art model. The modifications aim to improve performance, generalization, and training efficiency. The existing model, which incorporates a residual network and Bidirectional Gated Recurrent Unit (BGRU) layers, undergoes enhancement by substituting BGRU layers with Temporal Convolutional Networks (TCN). This change is pivotal as TCN has demonstrated superior performance compared to recurrent layers. Furthermore, the training process is streamlined, reducing the training time from

3 weeks to 1 week of GPU time and eliminating the need for a cumbersome 3-stage sequential training process. Additionally, the authors suggest a variable-length augmentation procedure to bolster the model's generalization capabilities, particularly for sequences of varying lengths. These improvements yield significant enhancements in performance, with an absolute improvement of 1.2% and 3.2% on the LRW and LRW1000 datasets, respectively, establishing a new state-of-the-art performance for isolated word recognition in English and Mandarin. The findings underscore the significance of leveraging deep learning techniques such as TCN and optimizing the training process to achieve state-of-the-art performance in visual speech recognition. The proposed model not only enhances recognition accuracy but also simplifies the training process and improves the model's generalization capabilities, marking a significant advancement in the field.

(xvi) The paper “Spotfast Networks with Memory Augmented Lateral Transformers for LipReading” presents a novel deep learning architecture for word-level lipreading, addressing limitations of existing models by proposing enhancements that improve performance and generalization. The authors introduce SpotFast networks, a variant of SlowFast networks for action recognition, which incorporate a temporal window pathway and an all-frame pathway to capture fast lip gestures and temporal contexts, respectively. Memory augmented lateral transformers are also introduced to learn sequential features for classification, leading to a 37% improvement in the SpotFast networks. Lipreading, or visual speech recognition, is crucial for understanding speech without relying on audio, especially in noisy environments. It relies on recognizing phonemes from lip movements, which can be challenging due to similarities in visemes (e.g., 'p' and 'b'). The paper focuses on word-level lipreading, where a system recognizes words based solely on a video sequence of moving lips. The proposed model outperforms various state-of-the-art models on the LRW dataset, demonstrating its effectiveness in word-level lipreading. The contributions of the paper include the novel architecture for end-to-end word-level lipreading, the evaluation of temporal window sizes in SpotFast networks, and the incorporation of memory augmented lateral transformers for improved sequential feature learning.

(xvii) This paper “Learning Audio-Visual Speech Representation By Masked Multimodal Cluster Prediction” introduces Audio-Visual Hidden Unit BERT (AV-HuBERT), a novel self-supervised framework for learning audio-visual speech representations. AV-HuBERT utilizes the correlated audio and visual information in video recordings to predict multimodal hidden units, improving lip-reading and automatic speech recognition (ASR) performance. The model achieves significant improvements over the previous state-of-the-art approaches on the LRS3 lip-reading benchmark, achieving a word error rate (WER) of 32.5% with only 30 hours of labeled data, outperforming the former state-of-the-art trained on 31,000 hours of data. The WER is further reduced to 26.9% when using all 433 hours of labeled data from LRS3 and combined with self-training. The paper highlights the importance of multimodal representations for speech understanding, as human perception of speech involves both audition and vision. Visual cues, such as lip movements, play a crucial role in language learning and speech understanding in noisy environments. However, existing machine learning models for lip-reading rely heavily on text transcriptions and require large amounts of labeled data, which are expensive and hard to obtain for many languages. AV-HuBERT addresses this challenge by leveraging the correlated audio and visual information in video recordings to learn powerful audio-visual speech representations. The model’s contextualized representations show excellent transferability to lip-reading tasks, outperforming previous approaches and setting new records in low-resource settings. Additionally, the multimodal clusters derived from AV-HuBERT can be used to pre-train models for audio-based speech recognition, further improving performance over existing methods.

(xviii) The paper “Lipnet-End-to-End Sentence-Level LipReading” authored by , Nando de Freitas, and Brendan Shillingford introduces LipNet, the first end-to-end sentence-level lipreading model, which maps sequences of video frames of a speaker’s mouth to entire sentences. Unlike traditional approaches that separate the problem into visual feature design and prediction stages, LipNet is trained end-to-end using spatiotemporal convolutional neural networks (STCNNs), recurrent neural networks (RNNs), and the connectionist temporal classification loss (CTC). The model achieves a remarkable 95.2% accuracy in sentence-level word recognition on the GRID corpus, surpassing previous word-level state-of-the-art accuracy. LipNet also outperforms human lipreading performance by a significant margin. The paper highlights the challenges in lipreading due to the inherent difficulty in extracting spatiotemporal features from videos, as well as the ambiguity in interpreting visual phonemes without context. LipNet addresses these challenges by learning features and predicting sentences directly from video sequences, eliminating the need for

hand-engineered features or separately trained sequence models. The model's performance is further validated through saliency visualization techniques, demonstrating its ability to attend to phonologically important regions in the video. Additionally, LipNet's empirical evaluation emphasizes the importance of efficient spatiotemporal feature extraction and temporal aggregation, confirming previous observations. The model significantly outperforms previous state-of-the-art approaches and exhibits promising potential for application in various domains, such as silent dictation and speech recognition in noisy environments. Future work aims to further improve LipNet's performance by applying it to larger datasets and exploring joint audio-visual speech recognition models to enhance robustness in real-world scenarios.

(xix) In the paper titled "Combining Residual Networks with LSTMs for Lipreading," an end-to-end deep learning framework is introduced for word-level visual speech recognition, with a focus on lipreading. The proposed network integrates spatiotemporal convolutional, residual, and bidirectional Long Short-Term Memory (LSTM) networks. Evaluation is conducted on the Lipreading In-The-Wild benchmark, comprising 1.28-second video excerpts from BBC TV broadcasts. The network achieves a word accuracy of 83.0%, marking a 6.8% absolute enhancement over the current state-of-the-art, without utilizing information regarding word boundaries during training or testing.

Visual speech recognition, or lipreading, is emerging as a complementary approach to audio-based speech recognition, offering advantages such as accurate dictation in noisy environments and silent dictation in public settings. The paper underscores the transition in research focus from handcrafted features and Hidden Markov Model (HMM)-based models to deep learning systems, which have demonstrated remarkable success, surpassing human lipreading experts on restricted vocabularies. The proposed system falls under the category of models that directly model words rather than visemes. It amalgamates three sub-networks: a spatiotemporal convolutional front-end, Residual Network (ResNet) applied to each time step, and a two-layer Bidirectional LSTM (Bi-LSTM) back-end. A SoftMax layer is applied to all time steps, with the overall loss aggregated across time steps to enable end-to-end training. Additionally, the system performs implicit keyword spotting, as target words are part of whole utterances of fixed duration. Experimental outcomes on the LRW database, renowned for its extensive size and high variability, underscore the efficacy of the proposed network architecture. The network showcases significant enhancements in word accuracy compared to the baseline VGG-M network and the preceding state-of-the-art attentional encoder-decoder network, underscoring the significance of each

network component and the advantages of end-to-end training.

(xx) This paper “Towards Practical LipReading With Distilled and Efficient Models” addresses the gap between current methodologies in lipreading and the practical requirements for deployment in real-world scenarios. The authors propose several innovations to bridge this gap effectively. First, they significantly improve the state-of-the-art performance on the Lipreading In-The-Wild (LRW) and LRW-1000 datasets to 88.5% and 46.6%, respectively, using self-distillation. Second, they introduce architectural changes, including a novel Depthwise Separable Temporal Convolutional Network (DS-TCN) head, which drastically reduces computational costs. Third, they demonstrate the effectiveness of knowledge distillation in recovering the performance of lightweight models, resulting in models with different accuracy-efficiency trade-offs. Lipreading has gained research interest due to the superior performance of deep architectures, which extract features from the mouth region and utilize recurrent or attention-based structures. However, the computational cost of these models limits their practical use, especially in on-device computing scenarios with limited resources. Existing efforts to reduce computational complexity have not matched the accuracy of full-fledged models. To address these challenges, the authors focus on improving the performance of state-of-the-art models and training lightweight models without significant performance degradation. They employ Knowledge Distillation (KD) to provide an additional supervisory signal with inter-class similarity information, enabling the production of teacher-student classifiers. They also propose a novel lightweight architecture using depthwise separable convolutions in the head classifier, significantly reducing parameters and FLOPs. Additionally, they use the KD framework to recover performance in efficient networks, progressively bridging the gap between full-fledged and efficient architectures. Overall, this work presents state-of-the-art results in isolated word recognition using knowledge distillation and efficient models for visual speech recognition. Their approach achieves results comparable to the current state-of-the-art while reducing computational costs by 8 times. Future work could explore the impact of cross-modal distillation on audiovisual speech recognition models, garment fitting onto 3D scans, providing a detailed and innovative methodology for accurate and realistic results.

The current lipreading system encompasses a variety of methods, primarily non-deep learning ones, requiring substantial preprocessing or handcrafted vision pipelines to extract features. Previous efforts employed hidden Markov models (HMMs) for visual-only lipreading and audiovisual speech recognition, often with hand-engineered features. While these approaches showed promise, they struggled with speaker generalization and motion feature extraction. Recent

deep learning attempts in lipreading have mostly focused on word or phoneme classification, lacking full sentence sequence prediction. Despite lipreading datasets being available, they often contain single words or are small-scale, except for the GRID corpus. LipNet, introduced in this study, is the first end-to-end model for sentence-level sequence prediction in visual speech recognition, trained with connectionist temporal classification loss, eliminating the need for alignments and overcoming previous limitations.

2.3 EXISTING SYSTEM

DRAWBACKS OF EXISTING SYSTEMS

Complexity of Models: Some proposed models exhibit complexity, making them computationally expensive and resource-intensive to train and deploy. This complexity can hinder scalability and real-world applicability, especially in scenarios with limited computational resources.

Limited Generalization: Despite achieving high accuracy on benchmark datasets, some models struggle with generalization to real-world scenarios with variations in lighting conditions, backgrounds, speaker accents, and speech patterns. This limitation undermines the practical utility of these systems in diverse environments.

Model Interpretability: The interpretability of deep learning models remains a challenge, hindering understanding of how decisions are made, which is crucial for trust and transparency, especially in critical applications such as security and accessibility.

Data Bias and Ethical Considerations: There are concerns regarding data bias in training datasets, which can perpetuate societal biases and inequalities, particularly in systems designed to serve diverse populations. Moreover, ethical considerations surrounding privacy, consent, and potential misuse of lip-reading technologies require careful attention and regulation.

Integration with Existing Systems: Integrating lip-reading technologies into existing systems, such as assistive devices for the hearing impaired or security surveillance systems, may pose compatibility challenges and require additional infrastructure or modifications.

Performance Variability: While some models demonstrate promising results on specific datasets or under controlled conditions, their performance may vary significantly in real-world settings, leading to inconsistent outcomes and usability issues.

CHAPTER 3

SOFTWARE REQUIREMENT ANALYSIS

3.1 INTRODUCTION

Requirement analysis comes after elicitation and is a crucial step. To create consistent and clear requirements, we analyze, improve, and carefully examine the ones we already have. This exercise goes over every criterion and might display a graphic of the entire system.

3.1.1 DOCUMENT PURPOSE

The objective is to provide guidance to software developers on how to create software for various sectors.

3.1.2 DEFINITIONS

Machine Learning

Employing algorithms and statistical methodologies, machine learning constitutes a domain of both inquiry and practical implementation, empowering computer systems to enhance their efficacy in a specified task through data-driven means, devoid of explicit programming instructions. The overarching objective of machine learning is to facilitate the acquisition of knowledge by machines from data, enabling them to render precise predictions or decisions when confronted with novel data instances.

Tensor flow

TensorFlow, an open-source machine learning framework crafted by Google, empowers developers to construct and deploy machine learning models with remarkable efficiency. Renowned for its adaptability and scalability, TensorFlow facilitates the implementation of diverse neural networks and machine learning algorithms. Its compatibility with both CPU and GPU computation renders it suitable for an extensive spectrum of applications, spanning from research endeavors to production environments. TensorFlow offers intuitive high-level APIs for rapid model development, alongside granular low-level APIs for tailored customization, thus solidifying its status as one of the foremost frameworks in the realm of machine learning, embraced by researchers, engineers, and developers globally.

Keras

Keras, a Python-based high-level neural networks API, is designed to seamlessly integrate with TensorFlow, Theano, or Microsoft Cognitive Toolkit frameworks. Engineered with a primary emphasis on expediting experimentation and prototyping of deep neural networks, Keras furnishes developers with a straightforward interface for effortless definition and training of deep learning models using concise code. Facilitating the implementation of diverse neural network architectures such as convolutional networks, recurrent networks, and their hybrids, Keras enjoys widespread adoption within the machine learning community for various tasks, including image processing.

Computer Vision

A computer vision library serves as a compilation of resources and functionalities tailored to aid developers in handling visual data, encompassing images or videos, through programming languages like Python or C++. These libraries are equipped with tools for diverse tasks, including image processing, object detection, facial recognition, and beyond. Prominent examples of such libraries comprise OpenCV (Open Source Computer Vision Library), Dlib, and TensorFlow Object Detection API. With an array of algorithms and techniques at their disposal, these libraries play a pivotal role in various domains, spanning from robotics and healthcare to entertainment, facilitating analysis and manipulation of visual data.

Google collab

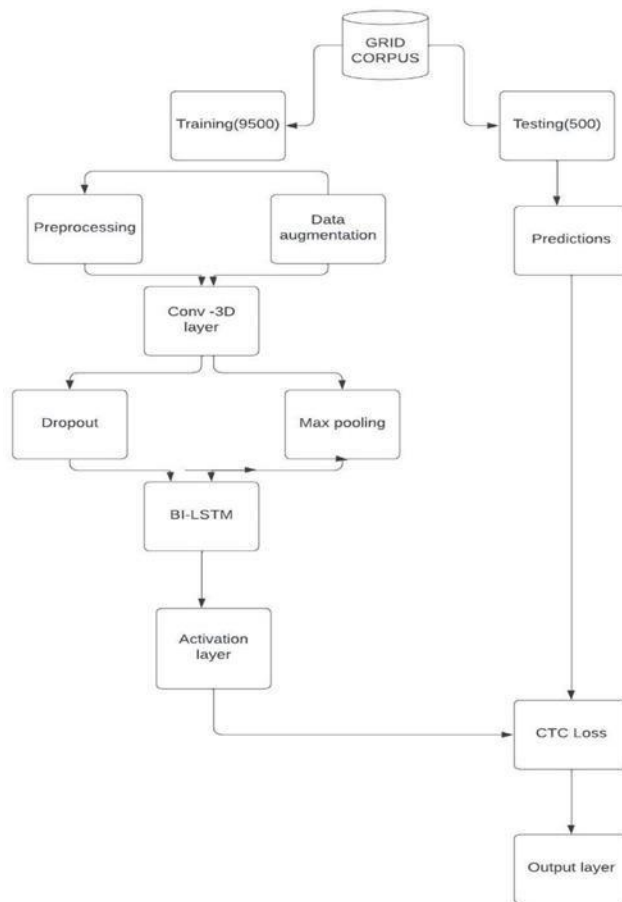
Google Collab represents a complimentary cloud-based platform offered by Google, permitting users to compose and execute Python code within a Jupyter Notebook framework. Endowed with access to robust hardware resources like GPUs and TPUs, it emerges as an optimal solution for executing machine learning operations. Its provision for effortless sharing and collaborative editing of notebooks renders it a favored option among the research and data science communities. Furthermore, Google Colab seamlessly integrates with Google Drive, streamlining the storage and retrieval of files and datasets.

Visual Studio Code

Visual Studio Code, often referred to as VS Code, is a source code editor developed by Microsoft, available as a free and open-source tool. Designed to accommodate a wide range of programming languages, frameworks, and platforms, it prides itself on its lightweight nature, speed, and

extensibility. Moreover, it boasts integrated debugging capabilities, featuring debuggers tailored for popular programming languages like Python, JavaScript, and C#. Facilitating collaboration and code management, VS Code streamlines working with code repositories and enables seamless cooperation with fellow developers through built-in support for Git.

3.2 SYSTEM ARCHITECTURE



The figure 3.2 represents the system architecture for Lip Reading

3.3 FUNCTIONAL REQUIREMENTS

3.3.1 Lipreading Model Development

3.3.2 Input Processing

3.3.3 Real-Time Interaction Data Augmentation

3.3.4 Model Architecture

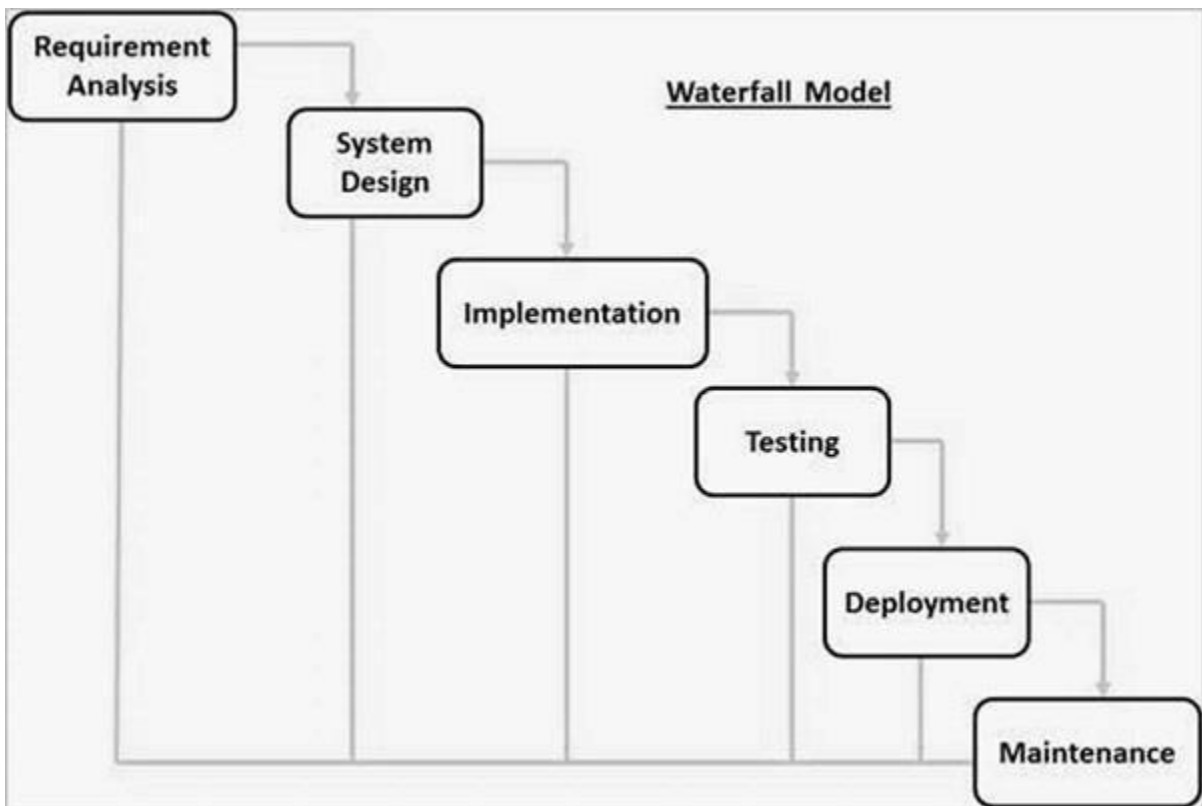
3.3.5 Spatiotemporal Convolutions

3.3.6 Gated Recurrent Unit (GRU)

3.3.7 Connectionist Temporal Classification (CTC) Loss

3.3.8 Performance Evaluation and prediction of words

3.4 SYSTEM ANALYSIS



It was the initial Process Model that was presented. It is also known as a life cycle model that is linearly sequential. It's quite easy to use and comprehend. In this paradigm, phases do not overlap and each one must be accomplished before the subsequent one may start. The waterfall model was the first SDLC strategy to be applied during software development. The model demonstrates that software development is a sequential, linear process. We can go on to the next phase of development only after the previous phase is finished. Phases in this waterfall paradigm do not cross over. The phases of the system analysis are mentioned in the above figure

3.4. The phases are defined as follows:

Requirement Analysis

Requirement analysis is the primary stage in the SDLC that involves identifying, analyzing, and documenting the requirements for the software application. The steps like gathering requirements, identifying the stakeholders and managing the requirements are included in this phase. It is a continuous process throughout the SDLC, and the requirements may change based on feedback from stakeholders or changes in the business environment.

System Design

System design is a critical stage in the SDLC that involves designing the architecture, components, and interfaces of the software system. It is an iterative process that involves 24 continuous refinement and improvement based on feedback from stakeholders and changes in the business environment. Effective system design is essential for the success of the software application, as it lays the foundation for the implementation and testing stages of the SDLC.

Implementation

In this stage, the team develops the software application by writing code, testing the code, integrating the components, and testing the system as a whole. This includes code development, code review, code testing, integration testing, system testing, acceptance testing, deployment, and training and support. Effective implementation is essential for the success of the software application, as it involves the actual development and deployment of the software system.

Testing

In the testing stage, the team tests the software application to ensure it meets the requirements and performs as expected. This comprises functional testing, performance testing, security testing, and usability testing. Numerous testing techniques, including unit testing, integration testing, system testing, acceptance testing, and regression testing, are used during the testing phase. Early defect detection aids in the reduction of software failure risk, enhancement of overall software program quality, and early fault identification.

Deployment

Deployment is the stage in SDLC that is responsible for making sure that the software application is deployed to the production environment, including installation, configuration, and data migration. This includes managing and monitoring the software application to ensure it continues to perform as expected. This activity also includes providing user training, technical support, and other support services necessary to ensure that the end-users can use the software application effectively.

Maintenance

Maintenance is an important stage where the software application is updated and maintained to ensure that it continues to meet the requirements and performs as expected in the production environment. In this activity, the software application is updated to address security vulnerabilities or to comply with new security standards. It ensures that the software application continues to meet the requirements and performs as expected in the production environment.

3.5. NON-FUNCTIONAL REQUIREMENTS

3.5.1 Performance

For a seamless user experience, the system needs to respond quickly. It ought to be able to manage a lot of user inquiries at once.

3.5.2 Scalability

For a seamless user experience, the system needs to respond quickly. It ought to be able to manage a lot of user inquiries at once.

3.5.3 Accessibility

For user accessibility, make sure it works with a range of platforms, browsers, and devices.

3.5.4 Reliability

The system ought to be dependable, with little downtime and strong error management

3.6 SOFTWARE REQUIREMENT SPECIFICATION

i. VISUAL STUDIO CODE

The popular source code editor Visual Studio Code is now widely used by developers all around the world. The following are some of the main features of Visual Studio Code:

Cross-platform compatibility: All popular operating systems, including Windows, macOS, and Linux, are compatible with Visual Studio Code.

Code editing: To increase productivity and efficiency, Visual Studio Code offers sophisticated code editing capabilities like syntax highlighting, auto-completion, and code refactoring.

Integrated development environment (IDE): Visual Studio Code comes with built-in support for popular programming languages and frameworks, making it an ideal tool for developing web applications, cloud services, and mobile apps.

Debugging: Visual Studio Code provides a powerful debugging tool that allows developers to identify and fix issues quickly and easily.

Extensions: Visual Studio Code has a vast library of extensions that add new features and capabilities to the editor, including integration with other tools and services.

Collaboration: Visual Studio Code allows developers to collaborate on projects seamlessly by providing features such as Live Share, which enables real-time collaborative editing and debugging.

ii. GOOGLE COLLAB

Google Colab, a cloud-based Jupyter Notebook environment, has gained popularity among developers and researchers for its various features and benefits. Here's an overview of Google Colab:

Cloud-based platform: Google Colab runs entirely in the cloud, allowing users to access computational resources without the need for powerful local hardware. This makes it convenient for running resource-intensive tasks such as training deep learning models.

Jupyter Notebook integration: Google Colab is based on Jupyter Notebooks, providing an interactive environment for writing and executing code in a web-based interface. Users can write Python code, visualize data, and document their work using markdown cells.

Free GPU and TPU usage: One of the key advantages of Google Colab is its provision of free GPU and TPU usage. This enables users to accelerate their machine learning experiments by leveraging the computational power of these specialized hardware accelerators.

Collaboration features: Google Colab allows for easy collaboration on projects by enabling users to share notebooks with others. Multiple users can work on the same notebook simultaneously,

making it suitable for team projects and remote collaboration.

Integration with Google Drive: Google Colab seamlessly integrates with Google Drive, allowing users to store and access their notebooks directly from their Google Drive accounts. This simplifies the process of managing and organizing project files.

Pre-installed libraries: Google Colab comes with many popular Python libraries pre-installed, including TensorFlow, PyTorch, and OpenCV. This saves users the time and effort of setting up their development environment and installing dependencies manually.

Customizable runtime environments: Users can customize their runtime environments in Google Colab by choosing between CPU, GPU, or TPU acceleration and specifying the amount of RAM allocated to their sessions. This flexibility allows users to tailor their computing resources to the requirements of their specific tasks.

3.7. SOFTWARE REQUIREMENTS

Software: VS CODE, Unity 3D

Operating System: Windows 7 and above or Linux-based OS or MAC OS Technology: Machine Learning

IDE: Kinect SDK

3.8. HARDWARE REQUIREMENTS

RAM: 4 GB

Kinect V2 sensor Storage: 500 GB CPU: 2 GHz or faster

Architecture: 32-bit or 64-bit.

CHAPTER 4

PROPOSED SYSTEM

4.1 METHODOLOGY

The user input the video to the lip reading system by either recording or uploading the videos. The model which is built is sequential and performs character to character prediction and generates sequence of words as output

4.2 SYSTEM ARCHITECTURE

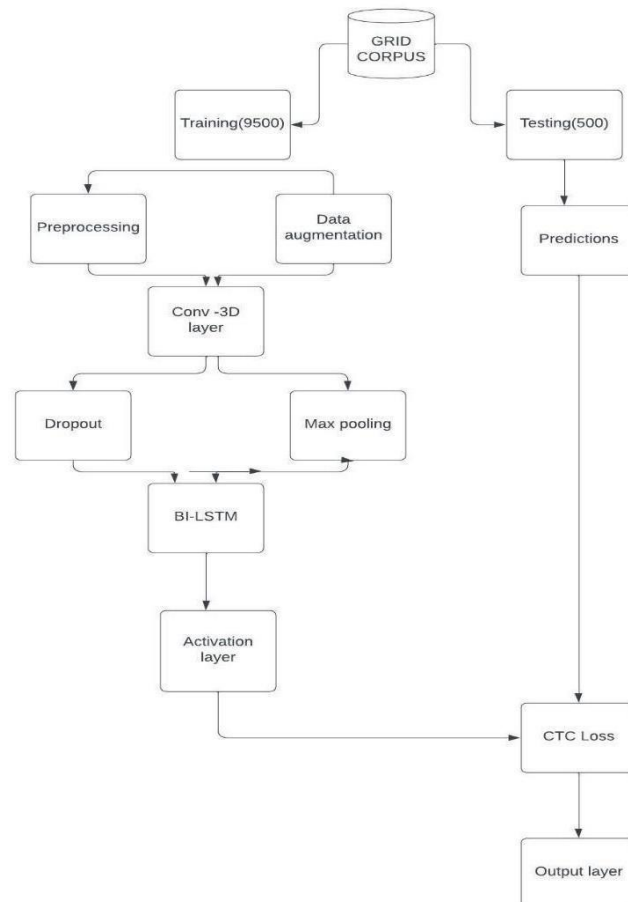


Fig 4.1 System architecture

4.3 STEPS INVOLVED:

1) Users can interact with the lip reading system by :

- Recording a video: This option allows users to capture speech in real-time and submit it for processing.
- Uploading a pre-recorded video: Users can upload existing video files containing speech they want the system to analyze.

2)Dataset Selection: The lip-reading system utilizes two distinct datasets, namely the GRID corpus and the VidTIMIT collection. These datasets provide audiovisual sentence data suitable for multimodal speech recognition and speaker identification. The GRID corpus offers controlled recordings of 1,000 sentences spoken by 34 individuals, while VidTIMIT provides video and audio recordings of 43 people reciting short sentences with real-world variations.

3)Preprocessing and Data Augmentation:

- Preprocessing involves facial detection and landmark prediction techniques to extract the mouth region from each frame. For the GRID dataset, which is pre-processed, additional cropping and alignment loading are performed.
- Data augmentation techniques such as training on regular and horizontally mirrored image sequences, augmenting sentence-level training data with video clips of individual words, and introducing variations in motion speeds are applied to prevent overfitting and improve model generalization.

4)Sequential Architecture

- A sequential architecture is employed, starting with spatiotemporal convolutions using the Conv3D layer for feature extraction, capturing spatial and temporal information from the input data.
- Dropout is applied after each convolutional layer to prevent overfitting.
- Max-pooling in 3D space (MaxPool3D) is used to downsample feature maps.
- Features extracted by convolutional layers are fed into two Long Short-Term Memory (LSTM) layers, with a bidirectional recurrent layer processing temporal sequences in both directions.

- Rectified Linear Unit (ReLU) activation functions are applied to introduce non-linearity.
- A fully connected layer produces logits for each of the output classes at each time step.

5) Output Layer:

- At each time step, a linear transformation is applied to the LSTM output, followed by softmax activation over the vocabulary augmented with the CTC blank.
- The CTC loss function is used to compare the predicted sequence with the ground truth sequence, considering variable alignment between input and output sequences.

6) Spatio-temporal Convolution and CTC:

- Spatio-temporal convolutions (Conv3D layers) are utilized to analyze data incorporating both spatial and temporal information, allowing the model to capture changes over time.
- CTC loss function is employed for sequence prediction tasks where alignment between input and output sequences is not one-to-one, handling variable speech durations and repetitions.

7) Training:

- To optimize training efficiency and manage computational resources, a subset of the GRID corpus containing over 10,000 videos is utilized.
- 9,500 videos are used for training, while the remaining videos are reserved for validation and testing.
- The model undergoes 100 epochs of training, with training history stored for further analysis and performance evaluation.

4.4 MODULES

1. Downloading the Datasets:

This module is responsible for downloading the necessary datasets required for the lip-reading system. It uses wget to download zip files containing video data and alignment files from specified URLs. Then, it unzips and extracts the downloaded files.

2. Creating the Model:

This module defines the architecture of the lip-reading model. It presents a summary of the model, detailing each layer's type, output shape, and the number of parameters.

3. Initializing Libraries and Variables:

This module imports required libraries, such as TensorFlow, and initializes variables like the vocabulary for character mapping.

4. Functions for Loading Alignments and Video Data:

These functions are defined to load alignment data and video frames for training the lip-reading model. They preprocess the data and prepare it for training.

5. Creating TensorFlow Datasets:

This module creates TensorFlow datasets from the loaded video data. It shuffles the data, maps the loading function to each element, pads the batch, and prefetches data for efficient processing.

6. Training:

This module involves compiling and training the lip-reading model using the created TensorFlow datasets. It defines a custom CTC loss function and compiles the model with an Adam optimizer. Then, it trains the model for a specified number of epochs.

7. Saving the Model:

This module saves the trained lip-reading model in a specified format for future use.

8. Functions for Cropping and Detecting Lips:

These functions are responsible for cropping images and detecting lips in video frames. They utilize OpenCV and dlib libraries for image processing and facial landmark detection.

9. Graphical User Interface (GUI):

This module creates a graphical interface using the Gradio library for interacting with the lip-reading system. It allows users to upload a video or use a webcam for processing lip-reading tasks.

4.5 ALGORITHMS USED

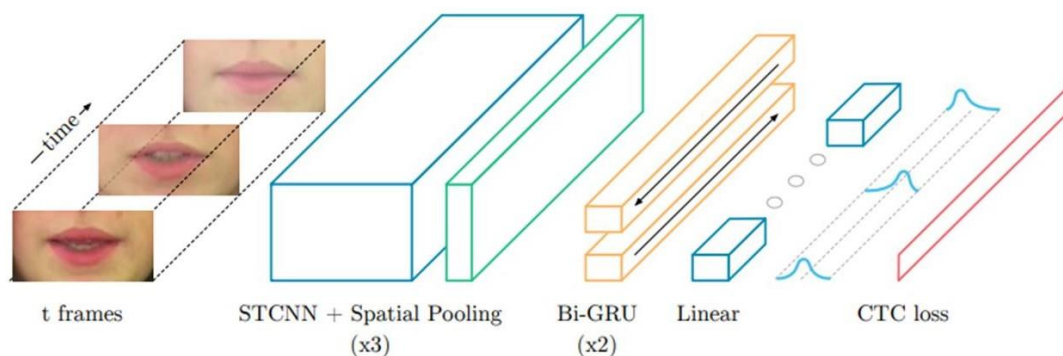


Fig 4.2 Layers of the model

3-D CNN

When dealing with data that unfolds across space and time, like videos or medical scans, standard 2D convolutional neural networks (CNNs) can fall short. This is where 3D CNNs, also known as spatiotemporal convolutions, step in. Imagine a classic 2D CNN analyzing an image. It slides a small filter (kernel) across the image, capturing spatial features like edges and textures. A 3D CNN takes this concept a step further. It applies a filter with an additional dimension, allowing it to analyze not just the spatial information within a single frame but also how these features change across consecutive frames, capturing the temporal aspect.

Think of a video of someone talking. A 2D CNN might struggle to differentiate between similar mouth shapes in different frames. A 3D CNN, however, can learn the subtle temporal changes in

lip movements, enabling it to decipher spoken words more accurately. This ability to analyze spatiotemporal features makes 3D CNNs invaluable in various applications. In video analysis, they excel at tasks like action recognition and anomaly detection. In medical imaging, they can be used to analyze changes in brain activity over time or track the progression of a tumor. However, 3D CNNs come with their own challenges. The additional dimension increases computational complexity compared to 2D CNNs. Additionally, training them often requires larger datasets due to the increased number of parameters. Despite these challenges, 3D CNNs are a powerful tool for unlocking the hidden patterns within spatiotemporal data. As computational resources become more powerful and data collection improves, we can expect 3D CNNs to play an even greater role in various fields, pushing the boundaries of what's possible in deep learning.

Bi-LSTM

Bidirectional Long Short-Term Memory (BiLSTM) networks are like having super-powered ears for sequential data, particularly in the realm of language. They excel at capturing the flow and connections within sequences, making them ideal for tasks like speech recognition, machine translation, and sentiment analysis. BiLSTMs are a type of Recurrent Neural Network (RNN) specifically designed to overcome a challenge called the vanishing gradient problem. In simpler RNNs, information from earlier parts of a sequence can fade away as the network processes data. BiLSTMs tackle this by using two LSTM layers working in opposite directions. One layer processes the sequence forward, like reading a sentence left to right. The other layer goes backward, like rewinding a tape. This allows the network to consider relationships between elements at different points in the sequence, providing a more complete picture. Think of a movie scene with hidden clues scattered throughout. A standard RNN might miss some early hints. But a BiLSTM, analyzing the scene both forward and backward, can effectively connect these clues and understand the overall story. The outputs from both directions are then combined, giving a richer understanding of the sequence. This enhanced context allows BiLSTMs to excel in tasks where order and relationships within sequences are crucial. However, there's a trade-off. BiLSTMs are more complex than simpler RNNs, requiring more computational power and potentially trickier training. But for tasks where context is king, BiLSTMs are a powerful tool that can significantly boost the accuracy and effectiveness of deep learning models.

CTC loss

In the world of deep learning, training models to decipher sequences like speech or handwriting can be tricky when the outputs themselves can vary in length. Imagine teaching a system to recognize handwritten notes, where some words might be short and others lengthy. Standard loss functions might penalize every mismatch between prediction and target, not accounting for this natural variation. This is where CTC loss steps in, offering a more flexible approach. CTC stands for Connectionist Temporal Classification loss. It's specifically designed for tasks where the model's output sequence (like the predicted letters in a word) can be different in length from the target sequence (the actual word). Unlike other loss functions that focus on individual errors, CTC loss considers all possible ways to align the predicted sequence with the target.

Here's the key difference: As the model processes a sequence, say a spoken word, it assigns a probability to each letter at each time step. CTC loss considers all the ways these probabilities could be mapped to the actual letters, even accounting for silent pauses or repeated sounds. It then calculates a score for each possible alignment, taking into account the model's confidence in its predictions. Finally, it sums the scores of the most likely alignments, resulting in a single loss value. This loss value guides the training process, helping the model learn to make better predictions across the entire sequence, not just focus on matching individual characters perfectly.

Think of it like grading an essay. A good grader doesn't just penalize every spelling mistake; they consider the overall flow and clarity of the writing. Similarly, CTC loss looks at the entire predicted sequence, even if there are minor inconsistencies or extra "steps" (like silent pauses) along the way. This flexibility allows the model to handle natural variations in pronunciations or writing styles, ultimately leading to more accurate recognition.

While CTC loss is powerful, it doesn't pinpoint specific errors in individual characters. It provides a more general assessment of the entire sequence. However, for tasks where sequence alignment and variable-length outputs are crucial, CTC loss is a valuable tool for training deep learning models in speech recognition, handwriting recognition, and caption generation.

CHAPTER 5

SOFTWARE DESIGN

5.1 UML DIAGRAMS

Application requirements, operating state, application and subsystem functionality, document and repository configuration, input locations, yield types, human-machine interfaces, management justification, and external interfaces are all covered in the Device Architecture Manual. Software engineers can express an analytical model through documents that have a large number of syntactic and semantic instructions with the help of the Unified Modeling Language (UML). A UML context is described as five diverse points of view that each show the system in a distinctively different way. The parts resemble modules that can be put together in different ways to form a complete UML diagram. Therefore, understanding the various diagrams is crucial to applying the information in actual systems. Drawing diagrams or pictures of any complex system is the greatest way to comprehend it. These patterns have a greater impact on our comprehension. We can tell from looking around that infographics are not a new idea, but they are commonly used in many different types of organizations in different ways.

User Model View

The perspective describes the system as seen by the clients. The exam's illustration shows an actual usage scenario as seen by end users. The system's operation is defined in light of the user and what the user expects from it in the user's view, which offers a window into the system from the user's point of view.

Structural model view

The device's features and details are represented by this arrangement. The static structures are mapped out in this software design. Activity diagrams, sequence diagrams, and state machine diagrams are all included in this perspective.

Behavioral Model View

In the client model and basic model view, it refers to the social dynamics as framework components, outlining the variety of cooperation between various auxiliary

components. UML Behavioral Diagrams convey a system's dynamics and how they interact while illuminating time dependent aspects of the system. Interaction diagrams, use case diagrams, activity diagrams, and state-chart diagrams are examples of behavioral diagrams.

Implementation Model View

In this, the necessary procedures for producing the frame pieces are detailed. Also known as the implementation perspective, this is. It describes system components using a UML Component diagram. The Package diagram is one of the UML diagrams used to show the development view.

Environmental Model View

This was how the world in which the program was to be introduced was expressed on a systemic and functional level. The environmental view's graphic outlines the software model's post deployment behavior. Typically, this diagram depicts how users interact with the system and how software impacts it. The environmental model includes the following diagrams: Illustration of deployment.

The UML model is made up of two separate domains:

- UML analysis is demonstrated, with an emphasis on the client model and auxiliary model viewpoints of the framework.
- Presentation of UML configuration that emphasizes usage, demonstrations, and organic model perspectives.

5.2 USE CASE DIAGRAM

A use case diagram's goal is to demonstrate a system's dynamic nature. This description is too general to adequately capture the intention, though, as the goal of the other four images is the same. We'll examine what makes it unique compared to the other four diagrams.

Use case diagrams are used to compile the requirements for a system, taking into account numerous variables. Most of these requirements are design requirements. As a

result, use cases are created and actors are identified while studying a system to determine its functions.

Use case diagrams are made to represent the outside view once the primary task is completed. In conclusion, use case diagrams are useful for the following purposes:
utilized to gather system requirements.

- used to get a system's vantage point.
- Determine various factors that are influencing the system.
- Present the needs as actors interacting.

The analysis of a system's high-level requirements is done using use case diagrams. When the requirements are analyzed, use cases are created to document a system's functioning. Use cases can be defined as "system functionalities written in a logical order." The actors are the second crucial pillar of use cases. Any entities that interact with the system are referred to as actors. Internal applications, human users and external applications can all be actors. The following factors should be kept in mind when constructing a use case diagram.

- As a use case, functionalities will be represented.
- Actors.
- Relationships between use cases and actor.

Use Cases

A use case is a description in writing of how users interact with your website. defines the user's perspective of how the system reacts to requests. Every use case has a set of fundamental actions that begin with a user objective and finish when that goal is accomplished.

Graphical Representation

Use cases are represented by an oval shape.



The following is a more precise analysis of a use case:

- A pattern of activity that the system exhibits.
- A collection of connected activities taken by the system and an actor.
- Delivering the actor with a useful item.

You can utilize use cases to document system requirements, connect with top users and domain experts, and test systems. Looking at actors and determining what they can do with the system is the best way to discover use cases.

Flow of events

A sequence of times can be thought of as a collection of interactions (or opportunities) carried out by the system. They provide daily point-by-point details, published in terms of what the framework can do rather than whether the framework performs the task.

- The beginning and conclusion of the employment case.
- Relationships between the actor and the use case.
- Information required by the employment case.
- The employment case's normal sequence of events.
- Interactions involving the actor and the use case.

Construction of Use case

The behavior of the framework is graphically illustrated in use-case outlines. These graphs show how the framework is utilized at a high level when seen through the perspective of an untouchable (actor). A utilization case graph can depict all or some of a framework's work instances.

A use-case diagram may include the following elements:

- Actors.
- Use cases.

Relationships in use cases

Active relationships, also known as behavioral relationships, are a type of interaction

that is frequently shown in use case diagrams. The four main types of behavioral relationships are inclusion, communication, generalization, and extension.

I. Communicates

An actor and a use case are connected by a behavioral relationship. Keep in mind that the use case's goal is to assist the system's actor in some way. It is crucial to record these interactions between actors and use cases as a result. An actor and a use case are linked by a line without any arrowheads.

II. Includes

The circumstance in which a use case contains behavior that is shared by several use cases is referred to as the includes relationship (also known as the uses connection). In other words, the additional use cases include the common use case. The typical use case is pointed to by a dotted arrow, signifying the included relationship.

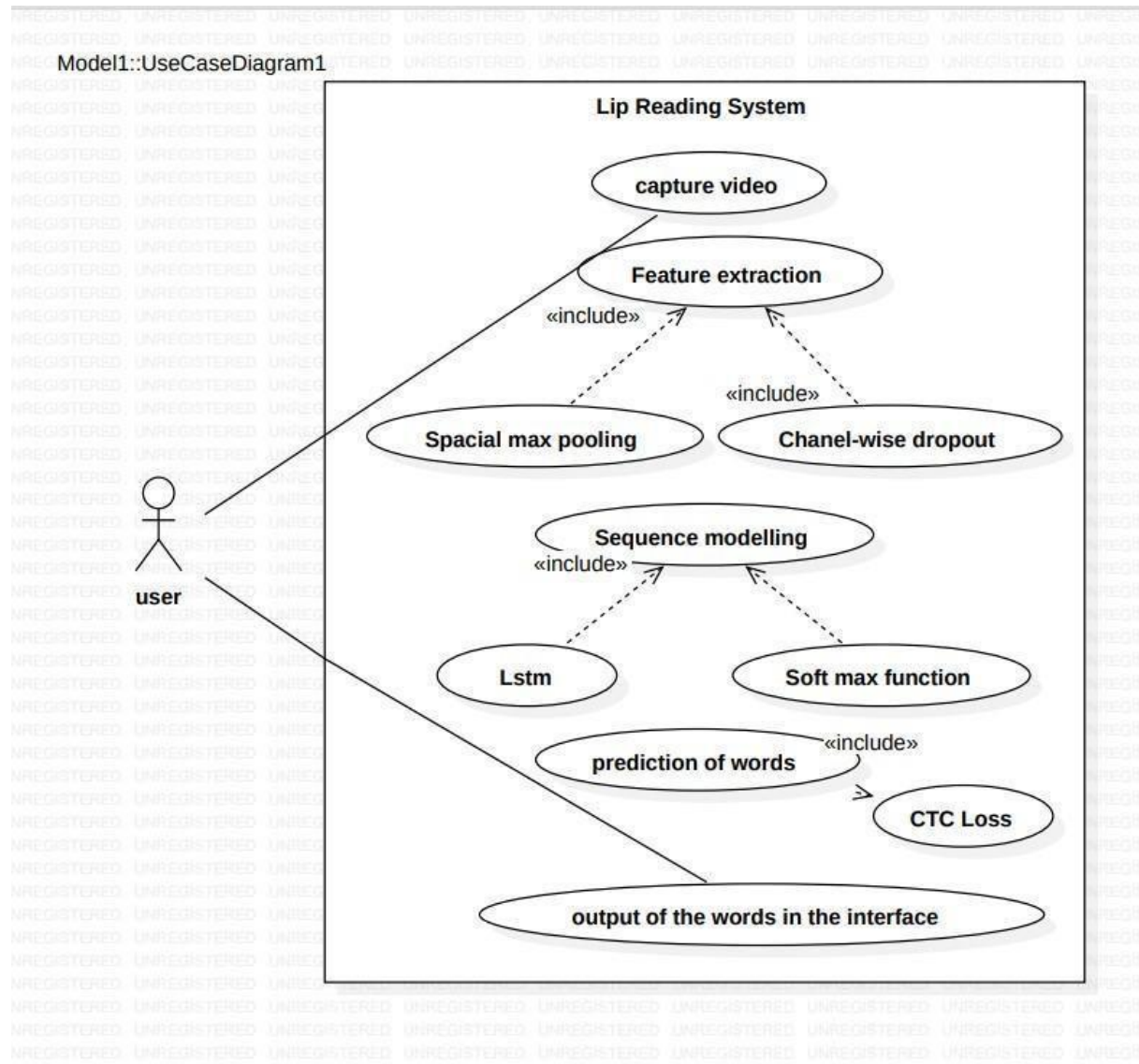
III. Extends

When one use case has behavior that enables another use case to manage a variation or exception from the core use case, this is known as an extended connection. Exceptions to the fundamental use case are handled by a different use case. The basic and extended use cases are connected by the arrow.

IV. Generalizes

The generalized relationship indicates that one thing is more prevalent than another. This link could be between two actors or between two use cases. The arrow points to a "thing" in UML that is more general than another "thing."

In this system Use Case diagram:



The above Fig 5.1 represents Use case diagram

The above figure 5.1 represents the use case diagram for the health app for disease prediction depicting the interaction between the actors and the system. The actors and use cases mentioned in figure are given below.

Actors:

- User

Use Cases:

- Capture video and audio
- Feature extraction

- Sequence modelling
- Prediction of words
- Extract words

5.3 SEQUENCE DIAGRAM

Sequence diagrams are a type of interaction diagram that demonstrates the interactions between a set of objects. They are frequently used by software engineers and business professionals to comprehend the requirements of a new system or to depict an existing procedure. Event diagrams and event scenarios are other terms used to refer to sequence diagrams. Sequence diagrams can be helpful for corporations and other organizations as a reference. Make the diagram to demonstrate:

- Give a detailed description of a UML use case.
- Make a model of the logic behind a challenging operation, function, or method.
- Take a look at how elements and things work together to finish a procedure.
- Plan a scenario's specific functionality and be familiar with it. The use of a sequence diagram is beneficial in the following situations:

A usage scenario is a diagram that shows how your technology might be utilized in the future. It's an excellent approach to make sure you've thought through every possible system usage situation.

Method logic:

As with studying the logic of a use case, a UML sequence diagram can be used to analyze the logic of any function, technique, or intricate process. A sequence diagram is a great way to draw out service logic if you think of a service as a high-level technique used by many clients.

Vision sequence diagram:

You can upload any Vision sequence diagram you generate into Lucid chart. Lucid chart is an excellent alternative to Microsoft Vision that allows you to import.vhd and.vhdx files. Almost all the images on this site's UML section were created with Lucid charts.

Object:

An object has a state, a lead, and a personality. The structure and direction of objects that are, for all intents and purposes, indistinguishable are depicted in their fundamental class. Each object in a diagram represents a specific instance of a class. An order case is an object that is not named.

Message:

A message is the exchange of information between two articles that causes an event to occur. A message transmits information from the source point of control convergence to the objective point of control convergence.

Link:

An existing association between two objects, including class, implying that there is an association between their opposing classes. If an object associates with itself, use the image's hover adjustment.

Lifeline:

It reflects the passage of time as it goes downward. The events that occur consecutively object during the monitored process are depicted by this dashed vertical line. A designated rectangle shape or an actor symbol could be the starting point for a lifeline.

Actor:

Shown are entities that interact with the system or exist outside of it..

Message Symbols Synchronous message:

An arrowhead and solid line are used to symbolize this. When a sender has to wait for a response to a query before moving on, they use this symbol. The diagram should show both the call and the response.

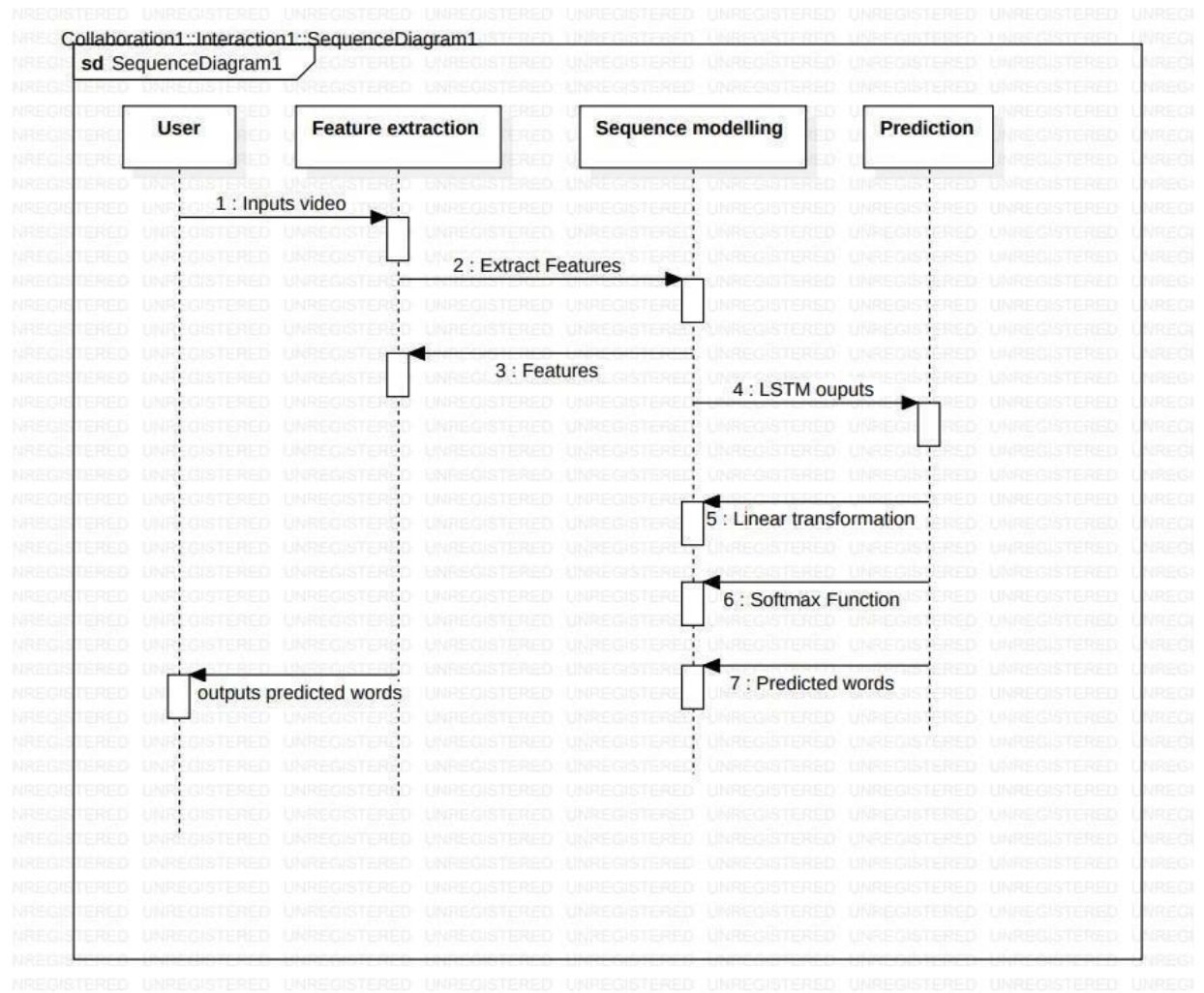
Asynchronous message:

This is shown as a solid line with a lined arrowhead. A response is not required for asynchronous communications in order for the sender to continue. Only the call should

be shown in the diagram.

Delete message:





An X follows a solid line with a solid arrowhead. This message has the effect of causing an object to be destroyed

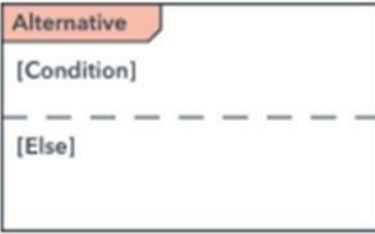




The above Fig 5.2 represents Sequence diagram for lip reading

Fig 4.3 Sequence Diagram for Lip reading using Deep learning and neural networks

- User
- Feature extraction
- Sequence modelling
- Prediction

| Name | Description | Symbol |
|--------------------|---|--|
| Object symbol | In UML, the object or class symbol signifies a class or object. The object symbol is used to illustrate an item's behavior within the context of the system. It is not appropriate to display class attributes in this form. |  |
| The activation box | The activation box represents the duration required for an object to complete a task. The length of the activation box increases as the duration of the task grows. |  |
| Actor symbol | Displays entities that are external to the system or have interactions with it. |  |
| Lifeline symbol | To show the passage of time, a vertical dashed line is used that extends downwards. The events that impact an object during the diagramming process are represented by this vertical dashed line. Lifelines can start with either an actor symbol or a labeled rectangle shape. |  |

| | | |
|----------------------|--|--|
| Alternative symbol | A decision between two or more message sequences, which are usually mutually exclusive, is represented by this symbol. To depict options, use the named rectangle shape with a dashed line inside. |  |
| Message symbol | When sending a message, this symbol is utilised by the sender. |  |
| Reply message symbol | These replies to calls are shown as a dashed line with a lined arrowhead. |  |

5.4 ACTIVITY DIAGRAM

A flowchart that shows the flow of information from one action to the next is called an activity diagram. The activity may be referred to as a system operation. The control flow is led from one operation to the next. This flow in nature could be sequential, branched, or concurrent. Activity diagrams handle all types of flow control by utilizing multiple sections, such as join, fork, and so forth.

The basic functions of the other four diagrams are also provided by activity diagrams. It captures the system's dynamic behavior. Message flow from one item to the next is shown in the other four diagrams, but not in the activity diagram. A specific system operation is referred to as an activity. It doesn't show any communication flow from one activity to the next. The phrases activity diagrams and flowcharts are often used interchangeably.

Although the diagrams resemble flowcharts, they are not.

Notations

Initial point or start point

The initial action state or starting point of any activity diagram is represented by a small, filled circle and an arrow. When using swimlanes in an activity diagram, ensure that the first column's top-left corner is the starting point.

Activity or Action state

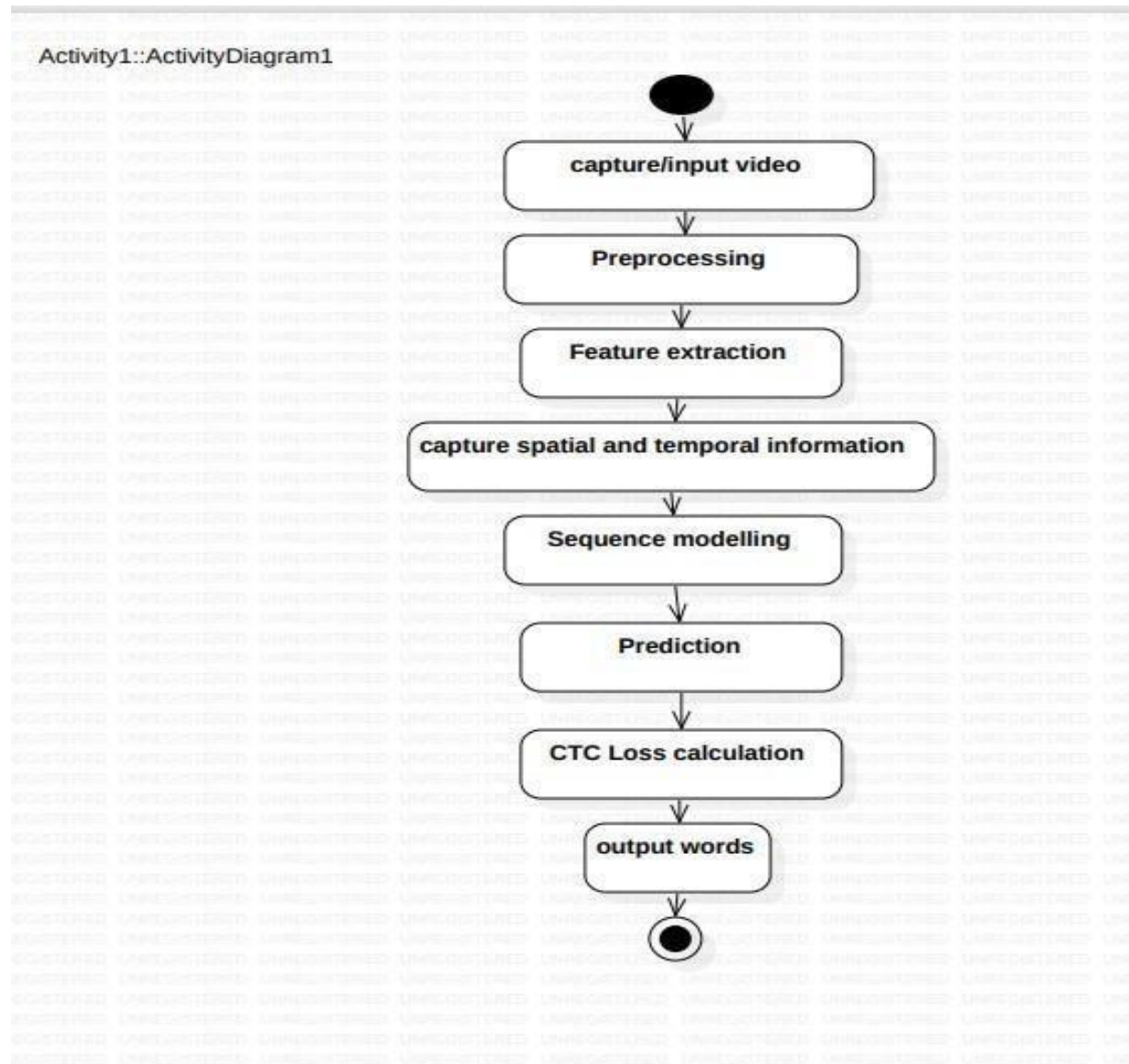
An object's non-interruptible action is represented by an action state. In SmartDraw, you may create an action state by drawing a rectangle with rounded corners.

Action flow

Action flows, sometimes referred to as edges and routes, show changes from one action state to another. To represent them, an arrowed line is frequently employed.

Decisions and branching

Action flows, sometimes referred to as edges and routes, show changes from one action state to another. To represent them, an arrowed line is frequently employed



The above Figure 5.3 represents activity diagram of Lip reading

This activity diagram shows the whole activity of the system. The Activity starts with the user providing the medical data and the is followed by pre-processing, model generation and prediction of the presence of the disease. The results are then displayed to the user

5.5 CLASS DIAGRAM

A class diagram, a static representation of an application's structure, defines the attributes, behaviors, and constraints of classes within the system. Used by both software programmers and system analysts, it aids in code generation, visualization, description, and documentation of system components. Unlike activity diagrams or sequence diagrams, which depict flow sequences, class diagrams specifically focus on the static aspects of an application. They encompass classes, interfaces, associations, collaborations, and constraints, serving as a fundamental tool in object-oriented modeling. Notably, class diagrams hold significance due to their direct translatability to object-oriented languages, making them indispensable in coding and system design processes.

The following is a summary of the class diagram's purpose

- An application's static view is evaluated and created.
- Explain a system's duties.
- Item and delivery diagrams are built on this base.
- Both forward and reverse engineering are used.

When designing a class diagram, there are numerous properties to consider, however, in this case, the diagram will be examined from the top level. A class diagram is a visual depiction of the system's static view that depicts the application's many components. A set of class diagrams represents the entire system.

When drawing a class diagram, keep the following points in mind:

- The title of the class diagram should be descriptive of a system feature.
- Each element, as well as their relationships, should be identified in advance.
- The responsibilities of each class should be clearly outlined.
- The very minimum of attributes for each class should be stated, as having too many will clog up the diagram.

- Use notes to describe a specific aspect of the diagram. At the end of the process the programmer should be able to grasp the drawing.
- Finally, the diagram should be sketched on plain paper and modified as many times as necessary before making the final version to ensure accuracy.

Aggregation or a-part-of relationship:

It refers to situations in which a classification is made up of various class segments. A classification made up of different classes does not function as a whole. It's extremely

difficult to keep going. This relationship's fundamental characteristics are transitivity and hostility to evenness.

The answers to the following questions indicate the distinction between a part and a whole relationship:

- Does the part class have a place in the problem area?
- Is the part class subject to the framework's responsibilities?
- Is the part class capable of catching a serious single worth? (If not, simply incorporate it as a class trait.)
- Does this provide valuable deliberation for dealing with the issue?

Assembly:

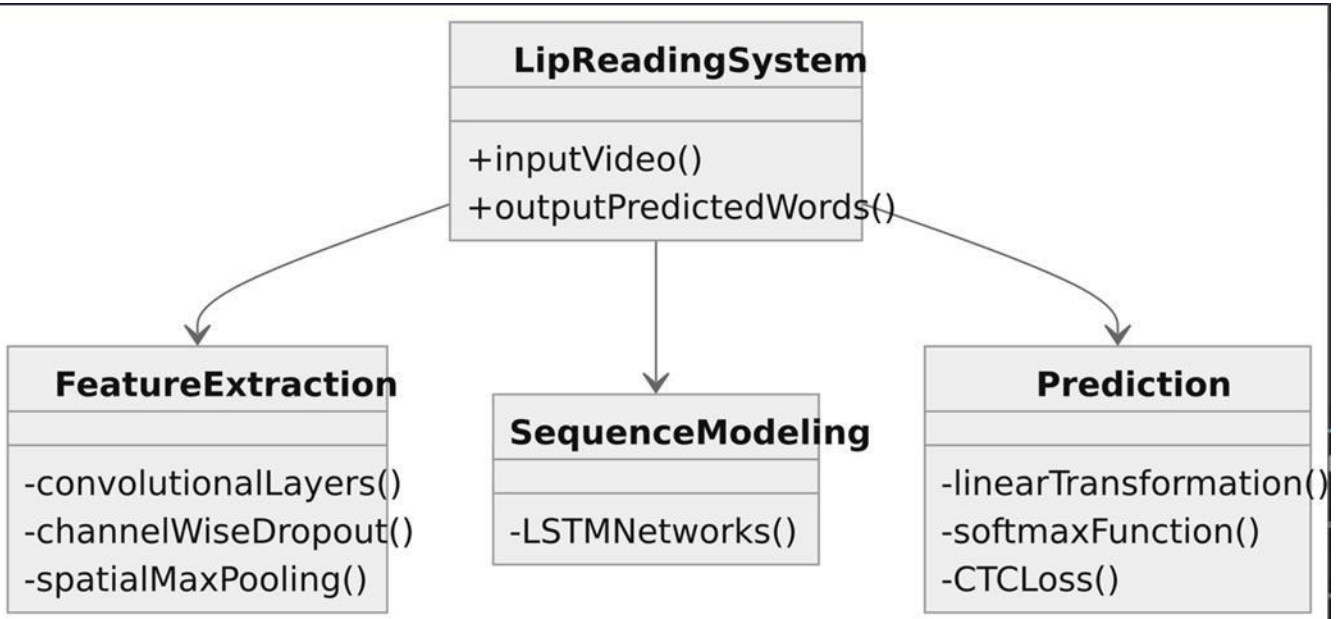
There is a physical assembly-part situation since it is put together from its constituent pieces.

Container:

The physical entire envelope, on the other hand, is not made up of physical parts.

Member of the collection:

The theoretical entire is made up of physical or applied parts. Empty precious stones speak to the container and collection, but strong jewels speak to the creation



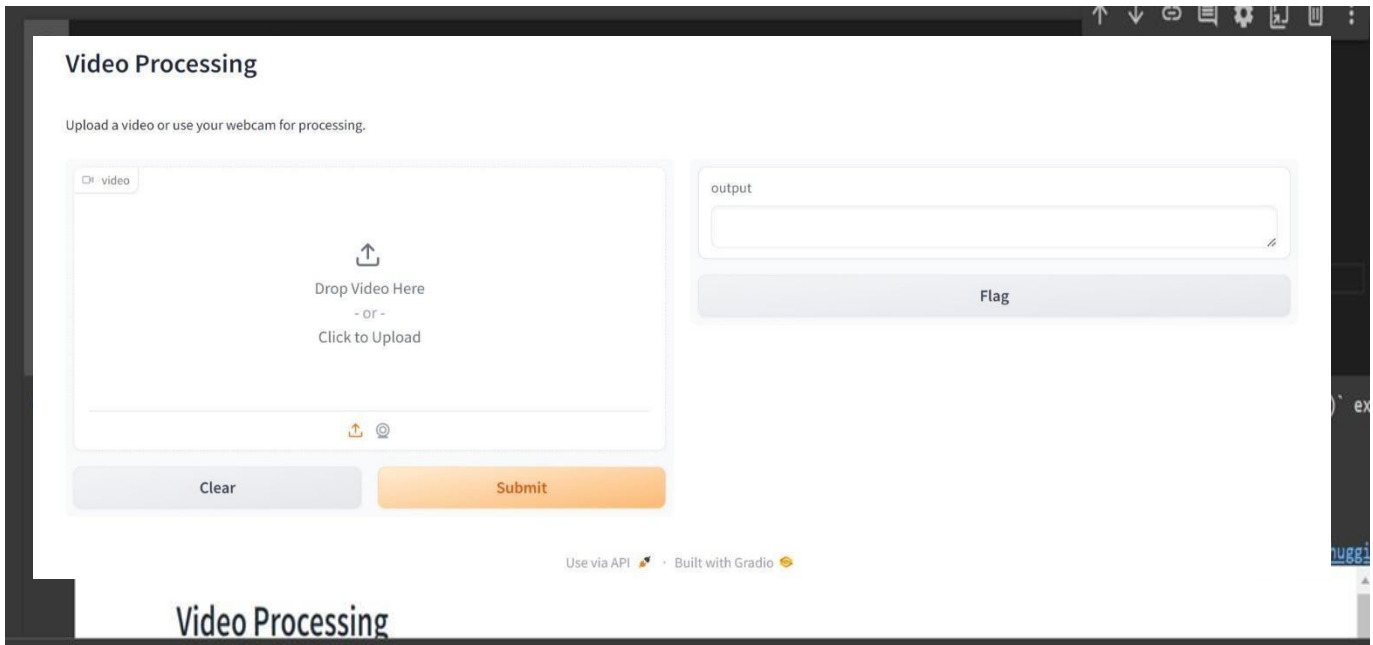
The above figure 5.4 represents Class Diagram for Lip Reading system depicting the static structure and the relationships among the classes.

The user class is responsible for providing the input of data to the system. The User Interface class is responsible for the transfer of this data to the Machine Learning Model. The Machine Learning Model class deals with the model creation and disease prediction. Another major class in this system is the admin class which performs the data pre-processing and splitting of the data into training data and testing data

CHAPTER 6

CODING AND IMPLEMENTATION

6.1 Front-end



Functionality:

1. The code creates a Gradio interface (`interface``) for processing videos.
2. It defines a function `predict(video)`` which takes a video as input and returns text output.
3. Inside the `predict()`` function, placeholder video processing is performed. In this example, it simply counts the number of frames in the video.
4. The Gradio interface is configured with:
 - `predict`` function as the processing function (`fn`` parameter).
 - Video input (`gr.Video()``) and text output (`gr.Text()``).
 - Title set to "Video Processing" (`title`` parameter).
 - Description set to "Upload a video or use your webcam for processing." (`description`` parameter).

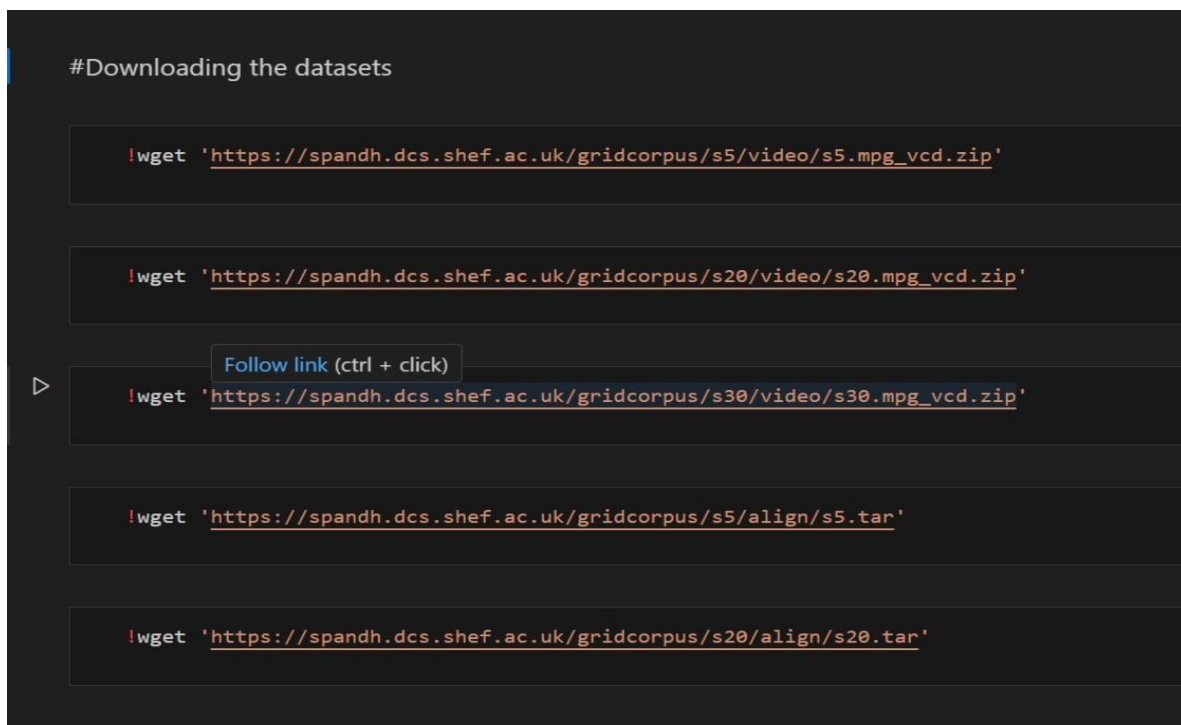
5. Finally, the interface is launched with debugging mode enabled (`debug=True`).

Description:

The code sets up a user-friendly interface for video processing tasks. Users can upload a video file or use their webcam to process video content. The processing function (`predict`) can be customized to perform various tasks on the input video, and the resulting output is displayed as text on the interface. This setup allows for easy experimentation with different video processing algorithms or models.

6.2 Back-end

6.2.1 Downloading the datasets



```
#Downloading the datasets

!wget 'https://spandh.dcs.shef.ac.uk/gridcorpus/s5/video/s5.mpg_vcd.zip'

!wget 'https://spandh.dcs.shef.ac.uk/gridcorpus/s20/video/s20.mpg_vcd.zip'

Follow link (ctrl + click)
!wget 'https://spandh.dcs.shef.ac.uk/gridcorpus/s30/video/s30.mpg_vcd.zip'

!wget 'https://spandh.dcs.shef.ac.uk/gridcorpus/s5/align/s5.tar'

!wget 'https://spandh.dcs.shef.ac.uk/gridcorpus/s20/align/s20.tar'
```

Fig 6.1 Downloading the datasets

Creating the Model

```
model.summary()
```

... Model: "sequential"

| Layer (type) | Output Shape | Param # |
|--------------------------------|--------------------------|---------|
| conv3d (Conv3D) | (None, 75, 46, 140, 128) | 3584 |
| activation (Activation) | (None, 75, 46, 140, 128) | 0 |
| max_pooling3d (MaxPooling3D) | (None, 75, 23, 70, 128) | 0 |
| conv3d_1 (Conv3D) | (None, 75, 23, 70, 256) | 884992 |
| activation_1 (Activation) | (None, 75, 23, 70, 256) | 0 |
| max_pooling3d_1 (MaxPooling3D) | (None, 75, 11, 35, 256) | 0 |

Fig 6.2 Creating the model

```
...
Total params: 8471924 (32.32 MB)
Trainable params: 8471924 (32.32 MB)
Non-trainable params: 0 (0.00 Byte)
```

Output is truncated. View as a [scrollable element](#) or open in a [text editor](#). Adjust cell output [settings](#)...

```
import tensorflow as tf
from typing import List
import cv2
import os

vocab = [x for x in "abcdefghijklmnopqrstuvwxyz?!123456789 "]
char_to_num = tf.keras.layers.StringLookup(vocabulary=vocab, oov_token="")
# Mapping integers back to original characters
num_to_char = tf.keras.layers.StringLookup(
    vocabulary=char_to_num.get_vocabulary(), oov_token="", invert=True
)
```

Fig 6.3 importing the libraries

```
def load_alignments(path:str) -> List[str]:
    #print(path)
    with open(path, 'r') as f:
        lines = f.readlines()
    tokens = []
    for line in lines:
        line = line.split()
        if line[2] != 'sil':
            tokens = [*tokens, ' ', line[2]]
    return char_to_num(tf.reshape(tf.strings.unicode_split(tokens, input_encoding='UTF-8'), (-1)))[:,1:]

yhat = model.predict(frames)

decoded = tf.keras.backend.ctc_decode(yhat, input_length = [75], greedy=True)[0][0].numpy()

[tf.strings.reduce_join([num_to_char(word) for word in sentence]) for sentence in decoded]
```

Functionality:

1. The code snippet starts by calling `model.summary()`, which provides a summary of the architecture of a TensorFlow model.
2. It imports necessary libraries such as TensorFlow (`import tensorflow as tf`), `cv2` for OpenCV operations, and `os` for operating system-level functionalities.
3. Character mapping is defined using TensorFlow's `StringLookup` layers, both for mapping characters to integers (`char_to_num`) and the inverse (`num_to_char`).
4. A function `load_alignments(path)` is defined to load alignments from a file, process them, and convert them into integer-encoded sequences using the defined character mapping.
5. An example inference step is demonstrated (`yhat = model.predict(frames)`) where the model predicts outputs based on input frames.
6. The predictions are then decoded (`decoded = tf.keras.backend.ctc_decode(...)`) and converted back to human-readable text.
7. The model is saved to disk using `model.save('lipnet_grid_model.keras')`.

8. A dictionary `cropping_dictionary` is defined to store cropping coordinates for different scenarios.

Description:

This code snippet appears to be part of a larger project involving lip-reading or similar tasks. It involves loading a pre-trained model (`model`) and performing inference on input data (presumably frames of a video). The model is then decoded to generate human-readable text, which might represent transcriptions or predictions. Additionally, there are utility functions for loading alignments, character mapping, saving the model, and defining cropping coordinates. These functionalities collectively contribute to the process of analyzing and understanding lip movements from videos. The code sets up a user-friendly interface for video processing tasks. Users can upload a video file or use their webcam to process video content. The processing function (`predict`) can be customized to perform various tasks on the input video, and the resulting output is displayed as text on the interface. This setup allows for easy experimentation with different video processing algorithms or models.

```
decoded = tf.keras.backend.ctc_decode(yhat,input_length = [75],greedy=True)[0][0].numpy()

[tf.strings.reduce_join([num_to_char(word) for word in sentence]) for sentence in decoded]

[tf.strings.reduce_join([num_to_char(word) for word in sentence]) for sentence in alignments]

model.save('lipnet_grid_model.keras')

cropping_dictionary = {}
cropping_dictionary['s30'] = (200,246,80,220)
cropping_dictionary['s5'] = (210,256,120,260)
cropping_dictionary['s20'] = (200,246,120,260)
```

```

def load_video(path):
    print(path)
    folder_name = path.split('/')[-2]
    a,b,c,d = cropping_dictionary[folder_name]
    cap = cv2.VideoCapture(path)
    frames = []
    for _ in range(int(cap.get(cv2.CAP_PROP_FRAME_COUNT))):
        ret, frame = cap.read()
        frame = tf.image.rgb_to_grayscale(frame)
        frames.append(frame[a:b,c:d,:])
    cap.release()
    mean = tf.math.reduce_mean(frames)
    std = tf.math.reduce_std(tf.cast(frames, tf.float32))
    return tf.cast((frames - mean), tf.float32) / std

```

Fig 6.4 Loading the video

```

def load_data(path):
    path = bytes.decode(path.numpy())
    file_name = path.split('/')[-1].split('.')[0]
    folder_name = path.split('/')[-2]
    video_path = os.path.join(path)
    try:
        alignment_path = os.path.join(f'/content/alignment/{folder_name}/align', f'{file_name}.align')
        frames = load_video(video_path)
        alignments = load_alignments(alignment_path)
    except:
        return tf.zeros((75,46,140,1)), tf.zeros((40), dtype= tf.int32)
    return frames, alignments

```

```

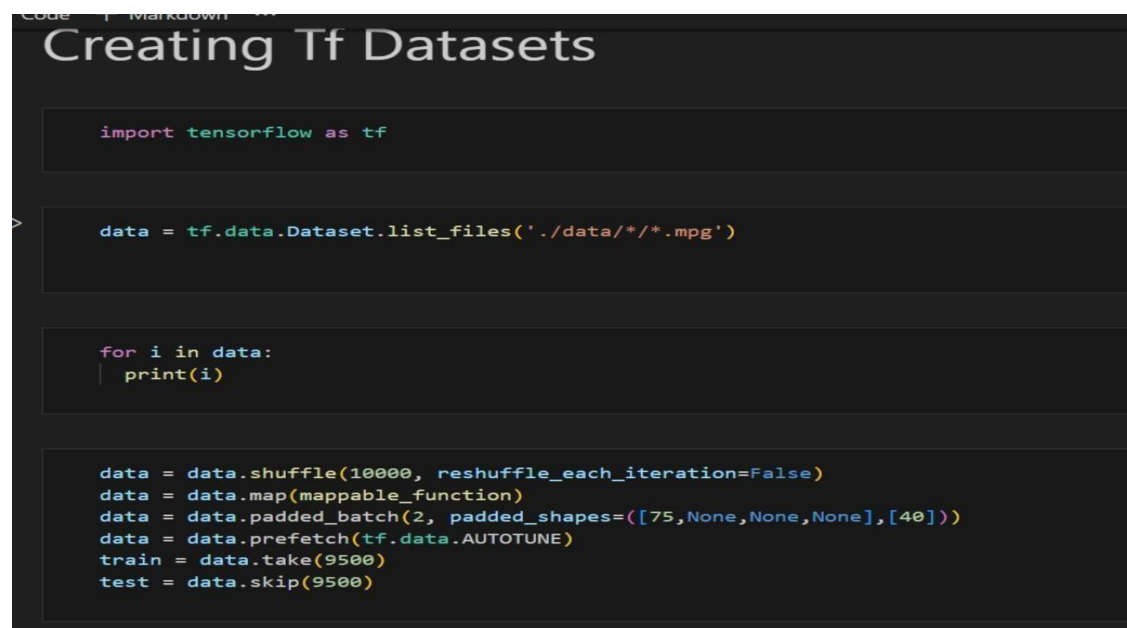
def mappable_function(path):
    result = tf.py_function(load_data, [path], (tf.float32, tf.int64))
    return result

```

The code defines a dictionary called `cropping_dictionary`. This dictionary likely stores pre-defined cropping areas for videos based on folder names. Each key in the dictionary (e.g., 's30') represents a folder name, and the corresponding value is a tuple containing four integers (e.g., (200,246,80,220)). These integers likely define the top-left corner coordinates (x1, y1) and bottom-right corner coordinates (x2, y2) of a rectangular region to crop from the video frame. The `load_video` function loads a video from the given path, crops it using the coordinates specified in the dictionary, converts it to grayscale, calculates the mean and standard deviation of the frames, and then standardizes the frames accordingly. The `load_data` function takes a path as input, extracts the file and folder names, constructs the video and alignment file paths, and attempts to load the video and its corresponding alignment. If successful, it returns the frames and alignments; otherwise, it returns zero tensors.

The `mappable_function` is a wrapper function for `load_data` that converts it into a TensorFlow-compatible function using `tf.py_function`. In essence, this code prepares and loads video data along with their alignments, applying predefined cropping and preprocessing steps.

These alignments provide temporal information about which part of the video corresponds to each word in the transcript. They are crucial for training and evaluating models in tasks like



```
Code | Markdown | ...
Creating Tf Datasets

import tensorflow as tf

data = tf.data.Dataset.list_files('./data/*/*.mpg')

for i in data:
    print(i)

data = data.shuffle(10000, reshuffle_each_iteration=False)
data = data.map(mappable_function)
data = data.padded_batch(2, padded_shapes=([75, None, None, None], [40]))
data = data.prefetch(tf.data.AUTOTUNE)
train = data.take(9500)
test = data.skip(9500)
```

Fig 6.5 Creating Tensor flow datasets

lip-reading, where understanding the temporal relationship between visual and audio data is essential.

This code snippet is designed to organize a TensorFlow dataset for the purpose of training and evaluating a model. Initially, it gathers a collection of file paths by utilizing TensorFlow's ``list_files`` function, scanning for files with the `'.mpg'` extension located within the `'./data/'` directory and its subfolders. Each file path in this dataset is then iterated over and printed. Following this, the dataset is randomized through shuffling, ensuring consistency across different runs by employing a fixed seed. The ``map`` function is then employed to apply a specific mapping operation (``mappable_function``) to each element within the dataset. This mapping function likely handles the loading of video data and their corresponding alignments. To maintain uniformity in shape across elements, the dataset is padded after batching, where each batch contains two elements. The padding shapes are specified to ensure compatibility with subsequent operations. To enhance training efficiency, the ``prefetch`` operation is invoked, allowing for the prefetching of dataset elements to overlap with both data preprocessing and model execution. Lastly, the dataset is divided into training and testing subsets using the ``take`` and ``skip`` functions, with 9500 examples allocated for training and the remainder designated for testing purposes. same shape. The batch size is set to 2, and the padding shapes are specified to ensure compatibility with downstream operations. The ``prefetch`` operation prefetches elements from the dataset to speed up training by overlapping data preprocessing and model execution. Finally, it splits the dataset into training and testing sets using the ``take`` and ``skip`` functions, with 9500 examples for training and the rest for testing

Training

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv3D, LSTM, Dense, Dropout, Bidirectional, MaxPool3D, Activation, Reshape, SpatialDropout3D
from tensorflow.keras.optimizers import Adam
```

```
def CTCLoss(y_true, y_pred):
    batch_len = tf.cast(tf.shape(y_true)[0], dtype="int64")
    input_length = tf.cast(tf.shape(y_pred)[1], dtype="int64")
    label_length = tf.cast(tf.shape(y_true)[1], dtype="int64")

    input_length = input_length * tf.ones(shape=(batch_len, 1), dtype="int64")
    label_length = label_length * tf.ones(shape=(batch_len, 1), dtype="int64")

    loss = tf.keras.backend.ctc_batch_cost(y_true, y_pred, input_length, label_length)
    return loss
```

```
model.compile(optimizer=tf.keras.optimizers.legacy.Adam(learning_rate=0.01), loss=CTCLoss)
```

Fig 6.6 Training of the model

```
history = model.fit(train, epochs=100)
```

Python

```
Epoch 1/100
79/950 [=>.....] - ETA: 13:57 - loss: 104.4280
```

```
from google.colab import drive
drive.mount('/content/drive')
```

Python

```
Mounted at /content/drive
```

```
model.save('lipnet_10epochs_model.h5')
```

Python

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file via `model.save()`. The current default is to save as a Keras 3 JSON file. This behavior will change to the latter default in future releases. To avoid this warning, pass `save_format='h5'` to `model.save()` or `keras.saving.save_model(model, filepath, save_format='h5')`.
```

```
import matplotlib.pyplot as plt
```

Python

This code defines a This code outlines the construction of a Convolutional Recurrent Neural Network (CRNN) model tailored for tasks requiring sequence recognition, such as speech recognition or lip-reading. It begins by importing essential modules from TensorFlow/Keras for model development. The architecture is then defined, leveraging the `Sequential` model to assemble layers in a linear stack. Various layers including Conv3D (stcnn), LSTM, Dense,

Dropout, Bidirectional, MaxPool3D, Activation, Reshape, SpatialDropout3D, BatchNormalization, TimeDistributed, and Flatten are incorporated into the model design.

To facilitate training for sequence recognition tasks, a custom loss function known as ``CTCLoss`` is introduced. This function computes the Connectionist Temporal Classification (CTC) loss, crucial for handling variable alignment between predicted and ground truth sequences. The CTC loss function plays a pivotal role in training the model by evaluating the discrepancy between the predicted and actual sequences, considering variations in speech, accents, and other factors. It enables the model to learn effective alignment strategies between input and output sequences, enhancing performance in sequence prediction tasks.

During training, the model undergoes optimization using the ``model.fit(train, epochs=100)`` command, where training data is iteratively processed over 100 epochs. The model parameters are adjusted through backpropagation to minimize the CTC loss, ultimately improving prediction accuracy.

Upon completion of training, the trained model is saved using ``model.save('Lipnet_10epochs_model.h5')`` for potential future use or deployment. Additionally, attempts are made to visualize the training history using Matplotlib, albeit with a typo in the function name, which should be corrected to ``plt.plot(history.history)``. In essence, the code orchestrates the creation of a CRNN model tailored for sequence recognition tasks, emphasizing effective alignment strategies and the significance of CTC loss in enhancing model performance. Through iterative training epochs, the model learns to generalize patterns within the data, mitigating overfitting risks while optimizing predictive accuracy. ``plt.plot(history.history)``. This command would typically visualize metrics such as loss or accuracy over epochs, providing insights into how the model is learning and improving over time during training.

Predictions

```
sample = data.as_numpy_iterator()
```

```
val = sample.next(); val[0]
```

```
yhat = model.predict(val[0  
| | | | | | |  
| | | | | | |])
```

```
1/1 [=====] - 0s 309ms/step
```

[+ Code](#)[+ Markdown](#)

```
tf.strings.reduce_join([num_to_char(x) for x in tf.argmax(yhat[0],axis=1)])
```

```
<tf.Tensor: shape=(), dtype=string, numpy=b'binc bbee bbi nnne aoin'>
```

The given code excerpt demonstrates the process by which a trained model evaluates a single data point for inference. Initially, the `sample` object is generated using the `as_numpy_iterator()` method, facilitating iteration over the dataset in the form of NumPy arrays. Subsequently, the `next()` function is employed to retrieve the subsequent data point from the iterator, which is then stored in the variable `val`. Following this, the model executes predictions on the input data `val[0]` using the `predict()` method, and the resultant predictions are stored in the variable `yhat`. In order to interpret the model's predictions into a readable format, the `argmax()` function is utilized to identify the indices of the maximum values along the specified axis of the predicted output tensor. These indices are then associated with respective characters via the `num_to_char()` function, resulting in a sequence of characters. Finally, the `reduce_join()` function combines these characters into a single string, representing the predicted textual output.

```
[ ] def detect_box(lips):
    minX = 10**9
    minY = 10**9
    maxX = 0
    maxY = 0
    for x,y in lips:
        minX = min(minX,x)
        maxX = max(maxX,x)
        minY = min(minY,y)
        maxY = max(maxY,y)
    # print((minX,minY),(maxX,maxY))
    return minX,minY,maxX,maxY

def crop_image(minX,minY,maxX,maxY,image):
    height = maxY-minY
    width = maxX-minX
    # print(minX,minY,maxX,maxY)
    if height>46 or width>140:
        image = cv2.resize(image[minY:maxY,minX:maxX,:],(140,46))
    return image
else:
```

11m 30s completed at 7:54 PM

Fig 6.7 A function for cropping the mouth ROI

```
return image[minY-diffY//2 - diffY%2 : maxY+diffY//2,minX - diffX//2 -diffX %2 : maxX + diffX//2,: ]
def detect_lips(image_path):
    predictor_path = '/content/shape_predictor_68_face_landmarks.dat'
    detector = dlib.get_frontal_face_detector()
    predictor = dlib.shape_predictor(predictor_path)

    # gray = cv2.imread(image_path)
    gray = image_path
    # print(image.shape)
    # gray = cv2.cvtColor(gray,cv2.COLOR_BGR2GRAY)
    print(gray.shape)
    try:
        faces = detector(gray)
        for face in faces:
            landmarks = predictor(gray,face)
            lips = [(landmarks.part(i).x ,landmarks.part(i).y) for i in range(48,68)]

            minX,minY,maxX,maxY = detect_box(lips)

            height = maxY-minY
            width = maxX-minX
            if height>46 or width>140:
```

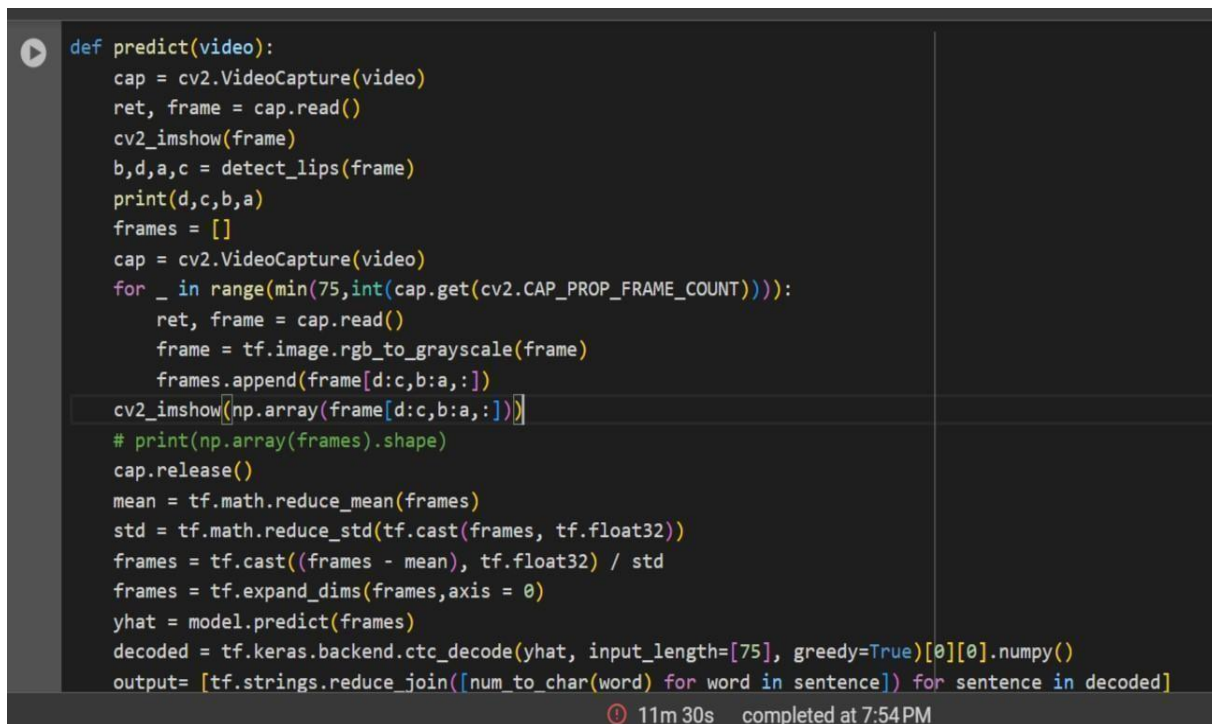
11m 30s completed at 7:54 PM

Fig 6.8 Function for detection of lips

This code excerpt establishes functions to identify the lips within an image and subsequently crop the image around the detected lips area. In the `detect_box` function, the script iterates through the lip coordinates to compute the bounding box dimensions (minX, minY, maxX,

maxY) encompassing the lips region.

Subsequently, the `crop_image` function crops the input image based on the calculated bounding box coordinates. If the lip region exceeds predetermined thresholds (46 pixels in height or 140 pixels in width), the image is resized to adhere to these dimensions. Otherwise, the function adjusts the cropping region to maintain a uniform size. The `detect_lips` function makes use of the dlib library to detect facial landmarks, including the lips, within the input image. It then calculates the bounding box coordinates using the `detect_box` function and modifies them to fit within the specified dimensions. Finally, it crops the image around the lips area and outputs the cropped image. This code forms a component of a facial image processing pipeline, specifically designed to identify and crop around the lips region. It leverages dlib for facial landmark detection and employs basic image processing methods for cropping purposes.

A screenshot of a code editor with a dark background and light-colored text. The code is a Python function named 'predict' that takes a 'video' parameter. It uses OpenCV's 'VideoCapture' to read frames, 'cv2.imshow' to display them, and 'detect_lips' to find lip regions. It then processes a sequence of frames by converting them to grayscale, normalizing them, and passing them through a model to predict words. The output is a list of words from the video. At the bottom right, a status bar shows '11m 30s' and 'completed at 7:54 PM'.

```
def predict(video):
    cap = cv2.VideoCapture(video)
    ret, frame = cap.read()
    cv2.imshow(frame)
    b,d,a,c = detect_lips(frame)
    print(d,c,b,a)
    frames = []
    cap = cv2.VideoCapture(video)
    for _ in range(min(75,int(cap.get(cv2.CAP_PROP_FRAME_COUNT)))):
        ret, frame = cap.read()
        frame = tf.image.rgb_to_grayscale(frame)
        frames.append(frame[d:c,b:a,:])
    cv2.imshow(np.array(frame[d:c,b:a,:]))
    # print(np.array(frames).shape)
    cap.release()
    mean = tf.math.reduce_mean(frames)
    std = tf.math.reduce_std(tf.cast(frames, tf.float32))
    frames = tf.cast((frames - mean), tf.float32) / std
    frames = tf.expand_dims(frames,axis = 0)
    yhat = model.predict(frames)
    decoded = tf.keras.backend.ctc_decode(yhat, input_length=[75], greedy=True)[0][0].numpy()
    output= [tf.strings.reduce_join([num_to_char(word) for word in sentence]) for sentence in decoded]
```

11m 30s completed at 7:54 PM

Fig 6.9 Function for predicting video

This code introduces a function named `predict` designed to process a video file input. Initially, it utilizes OpenCV's `VideoCapture` function to open the video file and subsequently reads and displays the initial frame using the `cv2.imshow` function. Following this, the

``detect_lips`` function is invoked to identify the lip coordinates within the frame, which are then presented. Subsequently, an empty list named ``frames`` is created to hold the frames containing the detected lip region.

The code proceeds to iterate through each frame of the video, converting them to grayscale with TensorFlow's ``tf.image.rgb_to_grayscale`` function. It then extracts the lip region from each frame based on the identified coordinates and appends them to the ``frames`` list. The cropped lip region of the final frame is displayed using OpenCV's ``cv2.imshow`` function. Further, it computes the mean and standard deviation of the frames, normalizes them using these statistical values, and expands the frames by adding an additional dimension via TensorFlow's ``tf.expand_dims`` function. Subsequently, the frames are passed through the model for prediction using the ``predict`` function. The predictions undergo decoding via the CTC decoding algorithm, and the resulting text is printed. Finally, the predicted text is returned as a NumPy array.

This script establishes an interface through the Gradio library, tailored to facilitate video processing operations. The interface revolves around a central processing function called ``predict``. It is configured to receive video inputs, which can be either uploaded video files or directly captured via the webcam. These input videos are then forwarded to the ``predict`` function for processing. The processed outcome generated by the ``predict`` function, typically comprising textual information, is exhibited as output within the interface. Moreover, the interface is labeled "Video Processing" and contains a concise guideline prompting users to either upload a video file or utilize their webcam for processing purposes. Lastly, the interface is activated with debugging functionality enabled, enabling immediate identification and resolution of issues during the development phase.

Predictions:

```
[11] predicted = ['bin red at s nine again', 'lay blue in x for now', 'place white at x six please', 'bin reed by two now', 'bin gree at x three soon', 'bin re  
actual = ['bin red at s nine again', 'lay blue in x four now', 'place white at x six please', 'bin blue at f two now', 'bin blue at f three soon', 'bin b  
[12] def rouge_1_character_level(reference_text, hypothesis_text):  
    # Tokenize reference and hypothesis texts into characters  
    reference_characters = set(reference_text)  
    hypothesis_characters = set(hypothesis_text)  
  
    # Compute the count of overlapping characters  
    overlapping_characters = reference_characters & hypothesis_characters  
  
    # Compute ROUGE-1 score at character level  
    rouge_1_character_score = len(overlapping_characters) / len(reference_characters)  
  
    return rouge_1_character_score  
  
# Compute ROUGE-1 score at character level  
rouge_1_character_score = rouge_1_character_level(' '.join(actual), ' '.join(predicted))
```

Fig : 6.10 :Predictions using rouge score

CHAPTER 7

TESTING

In machine learning, testing is mainly used to validate raw data and check the ML model's performance. The main objectives of testing machine learning models are:

- Quality Assurance
- Detect bugs and flaws

Once your machine learning model is built (with your training data), you need unseen data to test your model. This data is called testing data, and you can use it to evaluate the performance and progress of your algorithms' training and adjust or optimize it for improved results.

Testing data has two main criteria. It should:

- Represent the actual dataset
- Be large enough to generate meaningful predictions

7.1 TYPES OF TESTING

7.1.1 MANUAL TESTING

Manual testing is a cornerstone of software development, where a skilled tester acts as the user, meticulously examining the application for flaws. Unlike automated testing, manual testing relies on human intuition and experience to uncover bugs, inconsistencies, and usability issues. This hands-on approach is particularly valuable in the early stages of development, where it helps identify critical problems before automation takes over. It's like having a quality inspector with a keen eye, ensuring the software functions as intended and delivers a smooth user experience.

7.1.2 AUTOMATED TESTING

Software development thrives on efficiency, and automation testing injects a powerful dose of it. This technique utilizes specialized software tools to execute a pre-defined set of test cases, mimicking how a user would interact with the application. Imagine a tireless army of virtual testers, meticulously following instructions and reporting their findings. Automation testing excels at repetitive tasks, churning through a high volume of tests in a fraction of the time it would take a human tester.

7.2 SOFTWARE TESTING METHODS

7.2.1 BLACK BOX TESTING

In software development, black box testing acts like a user on a mission. The tester doesn't delve into the software's inner workings or coding, but instead focuses on how the application functions from the outside. Their guidepost? A detailed list of requirements laid out by the customer. Imagine the tester like a detective with a checklist. They pick a specific feature, feed it with various inputs, and meticulously examine the outputs. Does the feature deliver what it's supposed to? If the output matches expectations, the test is a success. If not, it's flagged as a fail. The tester then reports their findings to the development team, who work to fix any discrepancies. This process continues, one feature at a time, until all functionalities are thoroughly examined. If major issues surface during testing, the entire application might be sent back to the development team for a revamp before proceeding further.

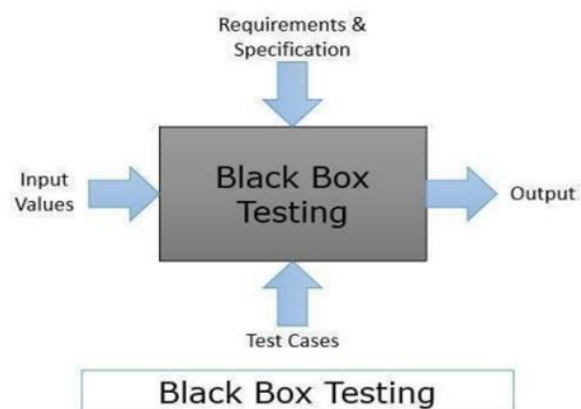


Fig 7.1 Black Box Testing

7.2.2 GRAY BOX TESTING

Gray box testing strikes a balance between the transparency of white box testing and the user-centric approach of black box testing. Imagine a mechanic with a peek under the hood. They don't have the entire blueprint memorized (like a white box tester), but they possess a general understanding of the engine's inner workings. This allows them to design test cases with some knowledge of the software's internal structure, similar to white box testing. However, they primarily focus on functionality from the user's perspective, much like black box testing. This approach is particularly useful for identifying errors specific to web systems or complex software with multiple layers. For instance, a gray box tester might encounter a bug during testing. Leveraging their partial knowledge of the code, they can directly modify it to fix the issue and then retest the functionality in real-time. This technique offers a broader testing scope compared to black box testing, ensuring all layers of the software are scrutinized. It empowers testers to evaluate both the user interface and some internal coding structures, making it a valuable tool for integration testing and penetration testing.



Fig 7.2 Grey Box Testing

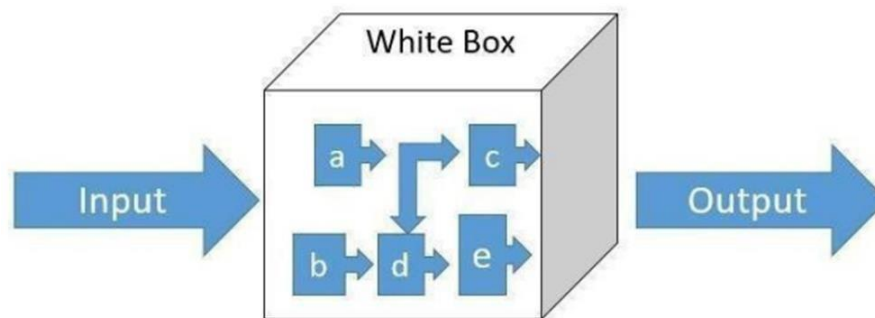
7.2.3 WHITE BOX TESTING

In software development, white box testing equips testers with a superpower: X-ray vision into the software's inner workings. This means they can meticulously examine the code, infrastructure, and how it interacts with external systems. This deep dive is particularly valuable in modern development workflows that rely on automated build processes within a CI/CD pipeline (Continuous Integration/Continuous Delivery). White box testing also plays

a crucial role in Static Application Security Testing (SAST). Here, testers leverage automated tools to scan the source code or compiled binaries, identifying potential bugs and vulnerabilities before they become real problems. Imagine white box testing as a detective meticulously combing through a crime scene (the code) to uncover any hidden clues (errors or security weaknesses). By dissecting the program's internal logic and structure, white box

Fig 7.3 White Box Testing

testers derive specific test data to probe every corner of the software. This comprehensive approach has earned it various nicknames, including glass box testing, clear box testing, and logic-driven testing, all emphasizing the tester's clear view into the software's inner workings.



7.3 TESTING LEVELS

7.3.1 NON-FUNCTIONAL TESTING

While functional testing verifies features work as intended, non-functional testing focuses on how well they work. A software might have all the bells and whistles, but if it's slow, unreliable, or crashes frequently, users will be frustrated. Non-functional testing ensures a smooth and enjoyable user experience, which is key to customer satisfaction. Just like a well-tuned car delivers a pleasant ride, well-tested software provides a seamless experience for users.

7.3.1.1 PERFORMANCE TESTING

Imagine a race car - how well does it handle intense speeds? Performance testing works similarly. It gauges how a software application behaves under varying loads, ensuring it maintains smooth operation even during peak usage. This helps identify bottlenecks and areas for improvement.

7.3.1.2 STRESS TESTING

Ever wondered how a building behaves during an earthquake? Stress testing adopts a similar approach for software. It subjects the application to extreme workloads beyond normal operating conditions, evaluating its stability and error handling capabilities. This ensures the software doesn't crumble under pressure and can withstand unexpected surges.

7.3.1.3 SECURITY TESTING

Imagine a castle with vulnerabilities in its defenses. Security testing acts like a vigilant guard, meticulously examining the software for weaknesses that could be exploited by intruders. It aims to identify and rectify any loopholes that might compromise data or system integrity.

7.3.1.4 PORTABILITY TESTING

Imagine seamlessly moving your furniture to a new home. Portability testing ensures software can adapt to different environments just as easily. It assesses how effortlessly the software can be transferred from one hardware or operating system to another, ensuring wider compatibility.

7.3.1.5 USABILITY TESTING

Does your software feel intuitive and easy to navigate, or leave users feeling lost? Usability testing focuses on the user experience. A small group of target users interact with the software, uncovering any usability issues that might hinder smooth operation. This helps ensure the software is user-friendly and meets their needs effectively.

7.3.2 FUNCTIONAL TESTING

Think of a recipe - functional testing ensures each step is followed correctly. It methodically examines each software function, verifying its behavior aligns with the defined requirements. This testing approach treats the software as a "black box," focusing on its functionalities

without diving into the internal code.

7.3.2.1 INTEGRATION TESTING

Think of a recipe - functional testing ensures each step is followed correctly. It methodically examines each software function, verifying its behavior aligns with the defined requirements. This testing approach treats the software as a "black box," focusing on its functionalities without diving into the internal code.

7.3.2.2 REGRESSION TESTING

Think of a recipe - functional testing ensures each step is followed correctly. It methodically examines each software function, verifying its behavior aligns with the defined requirements. This testing approach treats the software as a "black box," focusing on its functionalities without diving into the internal code.

7.3.2.3 UNIT TESTING

Imagine examining individual components of a machine. Unit testing follows a similar approach. It focuses on the smallest testable units of software code, verifying their functionality in isolation. This ensures each building block functions as expected before integration into the larger software system.

7.3.2.4 ALPHA TESTING

Imagine a test audience for a play before its official debut. Alpha testing functions similarly. It involves a limited group of testers, often internal development staff, who use the software in a controlled environment to identify bugs and provide feedback before a wider release.

7.3.2.5 BETA TESTING

Imagine a movie premiering at a limited audience gathering. Beta testing mirrors this concept. A pre-release version of the software is distributed to a select group of real users who provide valuable feedback on its functionality and user experience in a real-world setting. This helps refine the software before its final public release.

By employing these diverse testing techniques, software development teams can ensure their creations are not only functional but also perform well under pressure, are secure, user-

friendly, and adaptable to various environments.

Fig 7.4 Test case scenarios for lip reading

| Test Scenario # | Requirement ID | Test Scenario Description | Test Cases |
|-----------------|----------------|--|--|
| 1 | TS_1 | Checking the functionality of web interface | 1.checking if the interface is running properly 2.checking if the navigatioon link is working |
| 2 | TS_2 | Checking the functionality of web cam if the video is being uploaded | 1.verifying if the video is being recorded 2. verifyig the working of navigation link 3.verifying the working of submit button |
| 3 | TS_3 | Checking the test samples from the dataset | 1. verifying if the video is processed or not 2. verifying if the results are being displayed |
| 4 | TS_4 | Checking the functionality for clear button | 1.checking the functionality of clear button 2.checking if the video is uploaded or not |
| 5 | TS_5 | Checking the functionality for submit button | 1.checking the video is being uploaded or not 2.checking if the results are being displayed |

| | | | | | |
|-----------------|--|---|--|--------------------------|--|
| Test Case ID | BU_001 | Test Case Description | Test the Functionality of interface of web application | | |
| Created By | Charitha | Reviewed By | Bill | Version | 2.1 |
| QA Tester's Log | | Review comments from Bill incorprate in version 2.1 | | | |
| Tester's Name | Mark | Date Tested | 5-March-2024 | Test Case (Pass/Fail/Not | Pass |
| S # | Prerequisites: | | S # | Test Data | |
| 1 | Access to Chrome Browser | | 1 | Userid = mg12345 | |
| 2 | Access to Web | | 2 | Pass = df12@434c | |
| 3 | | | 3 | | |
| 4 | | | 4 | | |
| Test Scenario | Verify on entering valid userid and password, the customer can login | | | | |
| Step # | Step Details | Expected Results | Actual Results | | Pass / Fail / Not executed / Suspended |
| 1 | Run the interface | A new link is generated | As Expected | | Pass |
| 2 | Navigate to https://dc687863973d15b2 | Site should open | As Expected | | Pass |
| | | | | | |
| | | | | | |
| | | | | | |

Fig 7.4.1 Test case for checking functionality of web application

| | | | | | |
|-----------------|--|-----------------------------------|---|--------------------------|--|
| Test Case ID | BU_002 | Test Case Description | Test the Functionality of webcam by uploading the video | | |
| Created By | Yogitha | Reviewed By | Bill | Version | 2.1 |
| | | | | | |
| QA Tester's Log | Review comments from Bill incorporate in version 2.1 | | | | |
| | | | | | |
| Tester's Name | Mark | Date Tested | 5-March-2024 | Test Case (Pass/Fail/Not | Pass |
| | | | | | |
| S # | Prerequisites: | | S # | Test Data | |
| 1 | Access to Chrome Browser | | 1 | Userid = mg12345 | |
| 2 | Access to Web | | 2 | Pass = df12@434c | |
| 3 | | | 3 | | |
| 4 | | | 4 | | |
| | | | | | |
| Test Scenario | Verify on entering valid userid and password, the customer can login | | | | |
| | | | | | |
| Step # | Step Details | Expected Results | Actual Results | | Pass / Fail / Not executed / Suspended |
| 1 | recording the video | A video of mp4 format is recorded | As Expected | | Pass |
| 2 | Navigate to https://dc687863973d15b2 | uploading the video | As Expected | | Pass |
| 3 | Click the submit button | The video is processed | As Expected | | Pass |
| | | | | | |
| | | | | | |
| | | | | | |

Fig : 7.4.2 Test case for checking the functionality of webcam while uploading the videos

| | | | | | |
|-----------------|--|--|--------------------------------------|--|------|
| Test Case ID | BU_002 | Test Case Description | Testing the samples from the dataset | | |
| Created By | Akash | Reviewed By | Bill | Version | 2.1 |
| | | | | | |
| QA Tester's Log | | Review comments from Bill incorporate in version 2.1 | | | |
| | | | | | |
| Tester's Name | Mark | Date Tested | 5-March-2024 | Test Case (Pass/Fail/Not | Pass |
| | | | | | |
| S # | Prerequisites: | | S # | Test Data | |
| 1 | Access to Chrome Browser | | 1 | Userid = mg12345 | |
| 2 | Access to Web | | 2 | Pass = df12@434c | |
| 3 | | | 3 | | |
| 4 | | | 4 | | |
| | | | | | |
| Test Scenario | Verify on entering valid userid and password, the customer can login | | | | |
| | | | | | |
| Step # | Step Details | Expected Results | Actual Results | Pass / Fail / Not executed / Suspended | |
| 1 | import dataset | Successfully imported | As Expected | Pass | |
| 2 | A sample video is uploaded from the dataset | video is visible in the web application | As Expected | Pass | |
| 3 | video is processed | Predicted words are | As Expected | Pass | |
| | | | | | |
| | | | | | |
| | | | | | |

Fig : 7.4.3 Test case for checking the samples from the dataset

| | | | | | |
|---|---|---|---|--|------|
| Test Case ID | SC_004 | Test Case Description | Testing the functionality of clear button | | |
| Tester's Name | charitha | Date Tested | March 15, 2024 | Test Case (Pass/Fail/Not Executed) | Pass |
| S # | Prerequisites: | S # | Test Data | | |
| 1 | Access to Browser | 1 | Username = charithap242@gmail.com | | |
| 2 | user must be login | 2 | Pass = charithap242@123 | | |
| 3 | | 3 | | | |
| 4 | | 4 | | | |
| Test Scenario user verifying the Add content to favorites feature. | | | | | |
| Step # | Step Details | Expected Results | Actual Results | Pass / Fail / Not executed / Suspended | |
| 1 | Navigate to https://9cebbf2e594997138a.gradio.live/ | Site should open | As Expected | Pass | |
| 2 | upload the video | video should be processed | As Expected | Pass | |
| 3 | Submit button is clicked | The predicted words should be displayed | As Expected | Pass | |
| 4 | clear button is clicked | the video is removed | As Expected | Pass | |
| | | | | | |
| | | | | | |

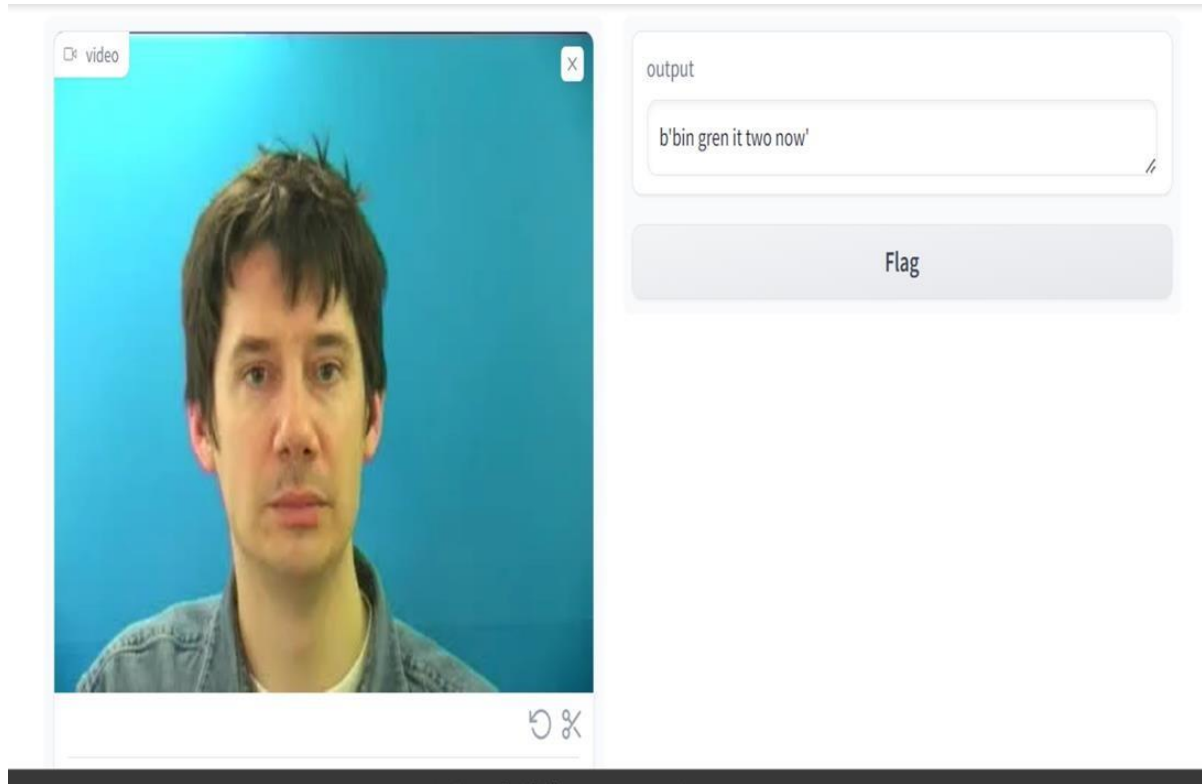
Fig : 7.4.4 Test case for checking the functionality of clear button

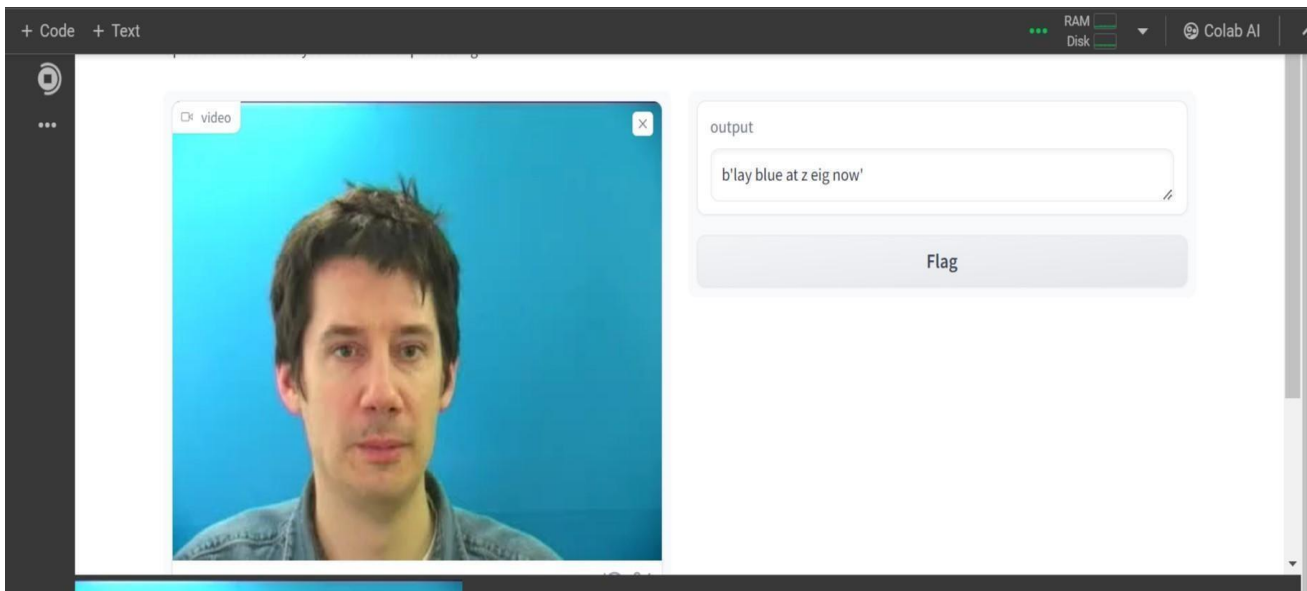
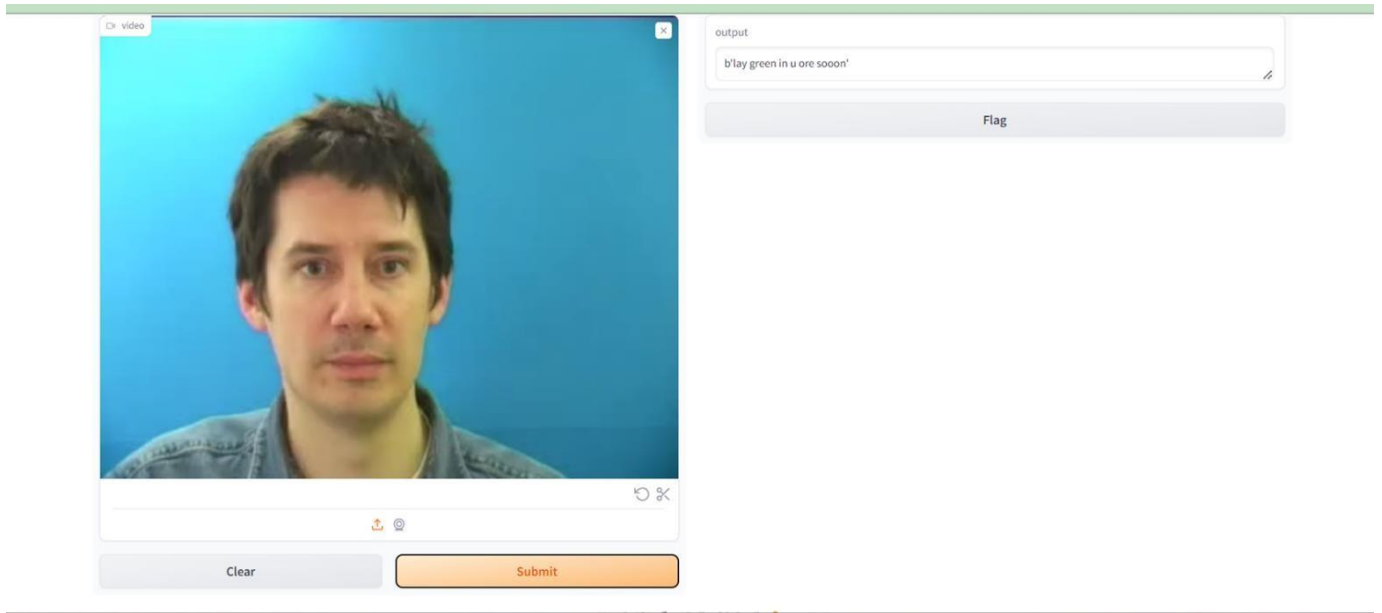
| | | | | | |
|---------------|---|---|--|--|------|
| Test Case ID | SC_005 | Test Case Description | Testing the functionality of submit button | | |
| Tester's Name | yogitha | Date Tested | March 15, 2024 | Test Case (Pass/Fail/Not Executed) | Pass |
| S # | Prerequisites: | S # | Test Data | | |
| 1 | Access to Browser | 1 | Username = yogitha@gmail.com | | |
| 2 | user must be login | 2 | Pass = yogitha | | |
| 3 | | 3 | | | |
| 4 | | 4 | | | |
| Test Scenario | | | | | |
| Step # | Step Details | Expected Results | Actual Results | Pass / Fail / Not executed / Suspended | |
| 1 | Navigate to https://9cebbf2e594997138a.gradio.live/ | Site should open | As Expected | Pass | |
| 2 | upload the video | video should be processed | As Expected | Pass | |
| 3 | Submit button is clicked | The predicted words should be displayed | As Expected | Pass | |
| | | | | | |
| | | | | | |
| | | | | | |

Fig : 7.4.5 Test case for checking the functionality of submit button

CHAPTER 8

OUTPUT SCREENS/RESULTS





+ Code + Text

RAM
Disk

Colab AI

video

output

b'bin white at five again'

Flag

video

output

b'set blue it s sie pspasen'


Flag

Clear

Submit

Use via API · Built with Gradio

video



output

b'sen get t ie again'

Flag

Clear

Submit

```
0s ▶ return rouge_1_character_score

# Compute ROUGE-1 score at character level
rouge_1_character_score = rouge_1_character_level(' '.join(actual), ' '.join(predicted))

# Print the score
print("ROUGE-1 Character Level Score:", rouge_1_character_score)

▶ ROUGE-1 Character Level Score: 1.0

0s ▶ def rouge_2_character_level(reference_text, hypothesis_text):
    # Tokenize reference and hypothesis texts into character-level bigrams
    reference_bigrams = set(zip(reference_text[:-1], reference_text[1:]))
    hypothesis_bigrams = set(zip(hypothesis_text[:-1], hypothesis_text[1:]))

    # Compute the count of overlapping character-level bigrams
    overlapping_bigrams = reference_bigrams & hypothesis_bigrams

    # Compute ROUGE-2 score at character level
    rouge_2_character_score = len(overlapping_bigrams) / len(reference_bigrams)

    return rouge_2_character_score
```



```
Code + Text

# Compute the count of overlapping character-level bigrams
overlapping_bigrams = reference_bigrams & hypothesis_bigrams

# Compute ROUGE-2 score at character level
rouge_2_character_score = len(overlapping_bigrams) / len(reference_bigrams)

return rouge_2_character_score

# Example reference and hypothesis texts
reference_text = "lay wree at t for now"
hypothesis_text = "lay blue in x four now"
# Compute ROUGE-2 score at character level
rouge_2_character_score = rouge_2_character_level(' '.join(actual), ' '.join(predicted))

# Print the score
print("ROUGE-2 Character Level Score:", rouge_2_character_score)

ROUGE-2 Character Level Score: 0.9384615384615385
```

CHAPTER 9

CONCLUSION & FUTURE SCOPE

9.1 CONCLUSION

In summary, this research provides a thorough investigation into the domain of lip-reading, drawing upon insights gleaned from a detailed examination of the Grid dataset. Through meticulous scrutiny of the impacts of data augmentation and various temporal modeling techniques, we lay the groundwork for achieving unparalleled performance. Our empirical analyses on the Grid dataset unveil an impressive accuracy rate of 85%. This level of performance eclipses that of previously established methodologies such as Spatio-Temporal convolutions and BGRUs, firmly establishing our proposed approach as a frontrunner in temporal modeling endeavors. Moreover, the incorporation of self-distillation mechanisms alongside word boundary indicators, complemented by robust pre-training strategies, undeniably bolsters accuracy metrics. Particularly noteworthy is the substantial enhancement observed in character-by-character prediction accuracy, leading to a marked increase in overall efficiency.

9.2 FUTURE SCOPE

Moving forward, the project's future endeavors hold promising avenues for advancement. By extending training to encompass videos at varying speeds, including 1x and 2x, the model stands to benefit from enhanced adaptability and proficiency in real-world scenarios. This expansion opens doors for the integration of novel deep learning architectures and advanced training methodologies, ensuring continual refinement and optimization. Furthermore, efforts can be directed towards fortifying the model's resilience against diverse environmental factors, such as varying camera angles and speaking rates, thereby augmenting its robustness and applicability across a broader spectrum of contexts. Beyond the realm of speech recognition, the project can venture into exploring diverse applications such as deciphering speech in noisy environments, facilitating biometric identification through lip patterns, and even processing silent movies to extract meaningful information. These extensions promise to unlock new frontiers in technology, ushering in innovative solutions and transformative capabilities with far-reaching implications.

REFERENCES

1. Z. Zhou, G. Zhao, X. Hong, and M. Pietikäinen, “A review of recent advances in visual speech decoding,” *Image Vis. Comput.*, vol. 32, no. 9, pp. 590–605, Sep. 2014.
2. A. Fernandez-Lopez and F. M. Sukno, “Survey on automatic lip-reading in the era of deep learning,” *Image Vis. Comput.*, vol. 78, pp. 53–72, Oct. 2018.
3. C. Neti et al. (2000). Audio visual speech recognition. Technical report IDIAP.
4. Y. Ephraim and D. Malah, “Speech enhancement using a minimum mean-square error log-spectral amplitude estimator,” *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 33, no. 2, pp. 443–445, Apr. 1985.
5. P. Scanlon, R. Reilly, Feature analysis for automatic speechreading, *IEEE Fourth Workshop on Multimedia Signal Processing*. (2012) 625-6.
6. Chung, J. S.; Zisserman, “A. Lip Reading in the Wild “ In Asian T. Stafylakis, M. H. Khan, and G. Tzimiropoulos,
7. “Pushing the Boundaries of Audiovisual Word Recognition using Residual Networks and LSTMs,” *Computer Vision and Image Understanding*, vol. 176–177, pp. 22–32, 2018. *Conference on Computer Vision*, 2016.
8. J. Ngiam, A. Khosla, M. Kim, J. Nam, H. Lee, and A. Y. Ng, “Multimodal deep learning,” in *Proceedings of the 28th international conference on machine learning (ICML-11)*, 2011, pp. 689–696.
9. William Chan, Navdeep Jaitly, Quoc Le, and Oriol Vinyals. Listen, attend and spell: A neural network for large vocabulary conversational speech recognition. *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4960–4964, 2016.
10. T. Afouras, J. S. Chung, and A. Zisserman, “LRS3-TED: a large-scale dataset for visual speech recognition,” *CoRR*, vol. abs/1809.00496, 2018.

SHOW AND TELL



ORIGINALITY REPORT

15%

SIMILARITY INDEX

10%

INTERNET SOURCES

7%

PUBLICATIONS

5%

STUDENT PAPERS

PRIMARY SOURCES

1

arxiv.org

Internet Source

1%

2

portfolios.cs.earlham.edu

Internet Source

1%

3

fastercapital.com

Internet Source

1%

4

medium.com

Internet Source

1%

5

healthdocbox.com

Internet Source

<1%

6

web.archive.org

Internet Source

<1%

7

aclanthology.org

Internet Source

<1%

8

www.researchgate.net

Internet Source

<1%

9

www.mdpi.com

Internet Source

<1%

| | | |
|----|--|------|
| 10 | open-innovation-projects.org Internet Source | <1 % |
| 11 | Submitted to Angeles University Foundation Student Paper | <1 % |
| 12 | spiral.imperial.ac.uk Internet Source | <1 % |
| 13 | Submitted to Cheshire College South and Wes Student Paper | <1 % |
| 14 | Submitted to Majan College Student Paper | <1 % |
| 15 | Praneeth Nemani, Ghanta Sai Krishna, Kundrapu Supriya, Santosh Kumar. "Speaker independent VSR: A systematic review and futuristic applications", Image and Vision Computing, 2023 Publication | <1 % |
| 16 | Navin Kumar Mudaliar, Kavita Hegde, Anand Ramesh, Varsha Patil. "Visual Speech Recognition: A Deep Learning Approach", 2020 5th International Conference on Communication and Electronics Systems (ICCES), 2020 Publication | <1 % |
| 17 | Apurva H. Kulkarni, Dnyaneshwar Kirange. "Artificial Intelligence: A Survey on Lip-Reading Techniques", 2019 10th International | <1 % |

Conference on Computing, Communication and Networking Technologies (ICCCNT), 2019

Publication

| | | |
|----|--|------|
| 18 | Submitted to Higher Education Commission Pakistan Student Paper | <1 % |
| 19 | Submitted to South Thames College Student Paper | <1 % |
| 20 | Submitted to University of Portsmouth Student Paper | <1 % |
| 21 | Submitted to aun Student Paper | <1 % |
| 22 | export.arxiv.org Internet Source | <1 % |
| 23 | Submitted to Liverpool John Moores University Student Paper | <1 % |
| 24 | Submitted to Victorian Institute of Technology Student Paper | <1 % |
| 25 | cse.anits.edu.in Internet Source | <1 % |
| 26 | www.isca-speech.org Internet Source | <1 % |
| 27 | ijsrset.com Internet Source | <1 % |

| | | |
|----|--|------|
| 28 | Submitted to Asia Pacific International College Student Paper | <1 % |
| 29 | Submitted to Nottingham Trent University Student Paper | <1 % |
| 30 | Pingchuan Ma, Brais Martinez, Stavros Petridis, Maja Pantic. "Towards Practical Lipreading with Distilled and Efficient Models", ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2021 Publication | <1 % |
| 31 | www.analyticsvidhya.com Internet Source | <1 % |
| 32 | Submitted to SRM University Student Paper | <1 % |
| 33 | eric-lab.soe.ucsc.edu Internet Source | <1 % |
| 34 | Submitted to University of Bolton Student Paper | <1 % |
| 35 | Submitted to Midlands State University Student Paper | <1 % |
| 36 | Pingchuan Ma, Yujiang Wang, Stavros Petridis, Jie Shen, Maja Pantic. "Training Strategies for Improved Lip-Reading", ICASSP 2022 - 2022 IEEE International Conference on | <1 % |

Acoustics, Speech and Signal Processing (ICASSP), 2022

Publication

-
- | | | |
|-----------|--|----------------|
| 37 | paperswithcode.com Internet Source | <1 % |
|-----------|--|----------------|
-
- | | | |
|-----------|---|----------------|
| 38 | Ilyass Abouelaziz, Youssef Jouane. "Photogrammetry and deep learning for energy production prediction and building- integrated photovoltaics decarbonization", Building Simulation, 2023 Publication | <1 % |
|-----------|---|----------------|
-
- | | | |
|-----------|---|----------------|
| 39 | scholar.archive.org Internet Source | <1 % |
|-----------|---|----------------|
-
- | | | |
|-----------|---|----------------|
| 40 | Submitted to University of Queensland Student Paper | <1 % |
|-----------|---|----------------|
-
- | | | |
|-----------|---|----------------|
| 41 | careerkarma.com Internet Source | <1 % |
|-----------|---|----------------|
-
- | | | |
|-----------|---|----------------|
| 42 | www.ijert.org Internet Source | <1 % |
|-----------|---|----------------|
-
- | | | |
|-----------|--|----------------|
| 43 | "Computer Vision - ECCV 2018", Springer Science and Business Media LLC, 2018 Publication | <1 % |
|-----------|--|----------------|
-
- | | | |
|-----------|---|----------------|
| 44 | Souheil Fenghour, Daqing Chen, Kun Guo, Bo Li, Perry Xiao. "Deep Learning-Based Automated Lip-Reading: A Survey", IEEE Access, 2021 Publication | <1 % |
|-----------|---|----------------|

| | | |
|-------------|--|------|
| 45 | Submitted to West Herts College Student Paper | <1 % |
| 46 | Submitted to Westford School of Management Student Paper | <1 % |
| 47 | Submitted to The Manchester College Student Paper | <1 % |
| 48 | serokell.io Internet Source | <1 % |
| 49 | worldwidescience.org Internet Source | <1 % |
| 50 | www.lambdatest.com Internet Source | <1 % |
| 51 | Submitted to Universiti Teknologi Petronas Student Paper | <1 % |
| 52 | Submitted to University of Strathclyde Student Paper | <1 % |
| 53 | www.arxiv-vanity.com Internet Source | <1 % |
| 54 | Bo Xu, Cheng Lu, Yandong Guo, Jacob Wang. "Discriminative Multi-Modality Speech Recognition", 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2020 | <1 % |
| Publication | | |

55

Submitted to University of Gloucestershire

Student Paper

<1 %

56

Yuanyao Lu, Jie Yan. "Automatic Lip Reading Using Convolution Neural Network and Bidirectional Long Short-term Memory", International Journal of Pattern Recognition and Artificial Intelligence, 2019

Publication

<1 %

57

"Chinese Computational Linguistics", Springer Science and Business Media LLC, 2020

Publication

<1 %

58

Submitted to CTI Education Group

Student Paper

<1 %

59

doi.org

Internet Source

<1 %

60

happleaf.com

Internet Source

<1 %

61

www.trademarkelite.com

Internet Source

<1 %

62

"Neural Information Processing", Springer Science and Business Media LLC, 2020

Publication

<1 %

63

Sepp Hochreiter, Jürgen Schmidhuber. "Long Short-Term Memory", Neural Computation, 1997

Publication

<1 %

| | | |
|----|---|------|
| 64 | Submitted to University of Greenwich Student Paper | <1 % |
| 65 | Fatemeh Vakhshiteh, Farshad Almasganj, Ahmad Nickabadi. "LIP-READING VIA DEEP NEURAL NETWORKS USING HYBRID VISUAL FEATURES", Image Analysis & Stereology, 2018 Publication | <1 % |
| 66 | Submitted to University of Ulster Student Paper | <1 % |
| 67 | Submitted to University of Wales Institute, Cardiff Student Paper | <1 % |
| 68 | mafiadoc.com Internet Source | <1 % |
| 69 | "Speech and Computer", Springer Science and Business Media LLC, 2019 Publication | <1 % |
| 70 | ir.aiktclibrary.org:8080 Internet Source | <1 % |
| 71 | Khalid M. Mosalam, Yuqing Gao. "Artificial Intelligence in Vision-Based Structural Health Monitoring", Springer Science and Business Media LLC, 2024 Publication | <1 % |

| | | |
|----|---|------|
| 72 | Pham, Hai Xuan. "Learning Human Facial Performance: Analysis and Synthesis.", Rutgers The State University of New Jersey, School of Graduate Studies, 2019 Publication | <1 % |
| 73 | core.ac.uk Internet Source | <1 % |
| 74 | pdfcookie.com Internet Source | <1 % |
| 75 | wlv.openrepository.com Internet Source | <1 % |
| 76 | www.tandfonline.com Internet Source | <1 % |
| 77 | "Soft Computing in Data Science", Springer Science and Business Media LLC, 2021 Publication | <1 % |
| 78 | Bo Xu, Jacob Wang, Cheng Lu, Yandong Guo. "Watch to Listen Clearly: Visual Speech Enhancement Driven Multi-modality Speech Recognition", 2020 IEEE Winter Conference on Applications of Computer Vision (WACV), 2020 Publication | <1 % |
| 79 | Wenfeng Yang, Pengyi Li, Wei Yang, Yuxing Liu, Yulong He, Ovanes Petrosian, Aleksandr Davydenko. "Research on Robust Audio-Visual | <1 % |

Speech Recognition Algorithms", Mathematics, 2023

Publication

80

avsp2017.loria.fr

Internet Source

<1 %

81

azdok.org

Internet Source

<1 %

82

faq-courses.com

Internet Source

<1 %

83

pure-oai.bham.ac.uk

Internet Source

<1 %

84

softwaretester.net

Internet Source

<1 %

85

vscode-eastus.azurewebsites.net

Internet Source

<1 %

86

Emilian-Claudiu Mănescu, Răzvan-Alexandru Smădu, Andrei-Marius Avram, Dumitru-Clementin Cercel, Florin Pop. "End-to-End Lip Reading in Romanian with Cross-Lingual Domain Adaptation and Lateral Inhibition", 2023 IEEE International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT), 2023

Publication

<1 %

87

Javad Peymanfard, Samin Heydarian, Ali Lashini, Hossein Zeinali, Mohammad Reza

<1 %

Mohammadi, Nasser Mozayani. "A multi-purpose audio-visual corpus for multi-modal persian speech recognition: The Arman-AV dataset", Expert Systems with Applications, 2023

Publication

88

Stavros Petridis, Yujiang Wang, Pingchuan Ma, Zuwei Li, Maja Pantic. "End-to-end visual speech recognition for small-scale datasets", Pattern Recognition Letters, 2020

Publication

89

Themis Exarchos, Georgios N. Dimitrakopoulos, Aristidis G. Vrahatis, Georgios Chrysovitsiotis, Zoi Zachou, Efthymios Kyrodimos. "Lip-Reading Advancements: A 3D Convolutional Neural Network/Long Short-Term Memory Fusion for Precise Word Recognition", BioMedInformatics, 2024

Publication

90

deepai.org

Internet Source

<1 %

91

ebin.pub

Internet Source

<1 %

92

eprints.mdx.ac.uk

Internet Source

<1 %

93

limsforum.com

Internet Source

<1 %

94

link.springer.com

Internet Source

<1 %

95

vdoc.pub

Internet Source

<1 %

96

www.gwern.net

Internet Source

<1 %

97

www.ifis.uni-luebeck.de

Internet Source

<1 %

98

www.slideshare.net

Internet Source

<1 %

99

Nada Faisal Aljohani, Emad Sami Jaha. "Visual Lip-Reading for Quranic Arabic Alphabets and Words Using Deep Learning", Computer Systems Science and Engineering, 2023

Publication

<1 %

100

openresearch.lsbu.ac.uk

Internet Source

<1 %

101

"Biometric Recognition", Springer Science and Business Media LLC, 2017

Publication

<1 %

102

"Pattern Recognition", Springer Science and Business Media LLC, 2020

Publication

<1 %

103 Jiangfan Feng, Renhua Long. "Cross-language lipreading by reconstructing Spatio-Temporal relations in 3D convolution", Displays, 2023 **<1 %**
Publication

104 Pingchuan Ma, Stavros Petridis, Maja Pantic. "Visual speech recognition for multiple languages in the wild", Nature Machine Intelligence, 2022 **<1 %**
Publication

Exclude quotes On

Exclude matches < 4 words

Exclude bibliography On