RAJALAKSHMI ENGINEERING COLLEGE
An AUTONOMOUS Institution
Affiliated to ANNA UNIVERSITY, Chennai

VISUAL SPEECH RECOGNITION

Submitted by

MADHUMITHA M (221501072)
MENAKAA V (221501077)

AI19541 FUNDAMENTALS OF DEEP LEARNING

Department of Artificial Intelligence and Machine Learning

RAJALAKSHMI ENGINEERING COLLEGE,THANDALAM.

RAJALAKSHMI
ENGINEERING COLLEGE

BONAFIDE CERTIFICATE

NAME …………………………………………………………………………………..……..…

ACADEMIC YEAR……………….………SEMESTER…………BRANCH………………

UNIVERSITY REGISTER No.

Certified that this is the bonafide record of work done by the above students in the Mini Project titled **"VISUAL SPEECH RECOGNITION : A LIP READING TECHNOLOGY"** in the subject **AI19541 – FUNDAMENTALS OF DEEP LEARNING** during the year **2024 - 2025.**

Signature of Faculty – in – Charge

Submitted for the Practical Examination held on ________________

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ABSTRACT

This project introduces LipBuddy, an innovative visual speech recognition system designed to convert lip movements from video inputs into readable text. By utilizing advanced deep learning techniques, the system extracts spatial features from video frames and models temporal sequences to predict spoken words accurately. The goal of this application is to bridge communication gaps for individuals with hearing impairments, provide real-time captioning in noisy environments, and explore applications in security and accessibility.

With its user-friendly interface powered by Streamlit, LipBuddy delivers a seamless experience for users to visualize and interact with the system's outputs. The project reflects a commitment to creating an inclusive, technology-driven tool that transforms how visual speech recognition can be leveraged across various domains.

Keywords:

- Temporal Sequence Modeling
- Spatial Feature Extraction
- Visual Speech Recognition

TABLE OF CONTENTS

CHAPTER NO	TITLE	PAGE NO
	ABSTRACT	**III**
1.	**INTRODUCTION**	**1**
2.	**LITERATURE REVIEW**	**2**
3.	**SYSTEM REQUIREMENTS**	**3**
	3.1 HARDWARE REQUIREMENTS	
	3.2 SOFTWARE REQUIREMENTS	
4.	**SYSTEM OVERVIEW**	**6**
	4.1 EXISTING SYSTEM	
	4.2 PROPOSED SYSTEM	
	4.2.1. SYSTEM ARCHITECTURE DIAGRAM	
	4.2.2. DESCRIPTION	
5.	**IMPLEMENTATION**	**9**
	5.1 LIST OF MODULES	
	5.2 MODULE DESCRIPTION	
	5.2.1. ALGORITHMS	
6.	**RESULT AND DISCUSSION**	**11**
7.	**APPENDIX**	**12**
	SAMPLE CODE	
	OUTPUT SCREEN SHOT	
	IEEE PAPER	
8.	**REFERENCES**	**21**

CHAPTER 1
INTRODUCTION

Lip reading is an emerging field that poses significant challenges even for expert lip readers. It offers a unique opportunity for improvement through machine learning techniques. By enhancing lip reading technology, we can significantly improve speech recognition in noisy environments, benefiting hearing aid systems for individuals with hearing impairments. Additionally, lip reading systems provide a valuable tool for speech analysis in security contexts, especially when audio is corrupted or absent in video footage.

However, the task of accurately reading spoken words from visual lip movements remains challenging due to the diversity of languages, variations in diction, and differences in articulation. Even expert lip readers can only estimate around half of the spoken words correctly. To address this issue, we leveraged deep learning algorithms, specifically neural networks, to train and evaluate two different model architectures. The better-performing model was then fine-tuned for enhanced accuracy and integrated into a web application, providing a real-time lip-reading solution that translates lip movements into text.

CHAPTER 2
LITERATURE REVIEW

1. Early Approaches to Lip Reading

Early research in lip reading primarily relied on traditional machine learning techniques, such as Hidden Markov Models (HMMs) and Gaussian Mixture Models (GMMs), which used manually extracted features from the face and mouth regions. These approaches were often limited by their inability to capture the temporal nature of speech and required significant preprocessing. Furthermore, speaker variations and the complexity of speech patterns made accurate predictions difficult. Despite these limitations, these methods provided a foundational understanding of the importance of facial features in speech recognition.

2. Deep Learning and the Rise of CNNs and RNNs

With the advent of deep learning, there was a significant shift in lip reading research. Convolutional Neural Networks (CNNs) were employed to automatically extract spatial features from video frames, improving the system's ability to recognize facial features like lip shapes. Recurrent Neural Networks (RNNs), particularly Long Short-Term Memory (LSTM) networks, were then used to model the temporal dependencies in lip movements across video frames. This combination of CNNs and RNNs led to substantial improvements in accuracy by enabling the model to both capture the spatial features and the sequential nature of speech. A breakthrough moment in lip reading technology came with the development of LipNet, a system introduced by researchers at Oxford University. LipNet used a combination of CNNs and RNNs to recognize speech directly from video frames without the need for intermediate phoneme prediction. LipNet achieved remarkable performance, outperforming traditional lip reading methods and demonstrating the power of end-to-end deep learning models for visual speech recognition.

3. Applications of Lip Reading

Lip reading systems have significant real-world applications. For individuals with hearing impairments, lip reading systems can enhance communication by providing a visual alternative to audio-based speech recognition. This technology can also be integrated into hearing aids, enabling better speech understanding in noisy environments. Additionally, lip reading has valuable applications in security and surveillance, where audio is either corrupted or absent from video footage. Research has explored integrating lip reading technology into security cameras and surveillance systems to automatically transcribe spoken words from visual data, thus enhancing the security and analysis of video footage.

4. Challenges and Limitations

Despite advances, lip reading remains a challenging task. Visual speech data is often ambiguous, with different phonemes or words sometimes appearing visually similar on the lips. Factors such as variations in articulation, accent, and facial expressions further complicate lip reading systems. Environmental factors, including lighting conditions, camera angles, and occlusions (e.g., hands covering the mouth), can also hinder the accuracy of lip reading systems. Additionally, the lack of large-scale, annotated datasets remains a significant limitation, restricting the ability of models to generalize across different speakers, languages, and real-world conditions. Another notable challenge is the computational complexity of lip reading models. Real-time lip reading systems require high computational power to process video frames quickly enough to interpret speech in real time. As a result, optimizing deep learning models for both accuracy and efficiency is a critical area of ongoing research.

5. Recent Developments and Innovative Approaches

Recent studies have focused on addressing the challenges associated with lip reading by incorporating new techniques such as 3D Convolutional Neural Networks (3D CNNs) to capture both spatial and temporal features of lip movements more effectively. Attention mechanisms have also been explored to improve the model’s focus on the most relevant parts of the face, such as the lips, during the recognition process. Additionally, hybrid models combining CNNs with other techniques, like transformers or Generative Adversarial Networks (GANs), have shown promise in improving the robustness and accuracy of lip reading systems. Multi-modal systems, where visual data is combined with other inputs like facial expressions or head movements, are being developed to further enhance accuracy.

CHAPTER 3

SYSTEM REQUIREMENTS

3.1 HARDWARE REQUIREMENTS:

- CPU: Intel Core i5 or equivalent (minimum)
- GPU: NVIDIA GTX 1080 or higher (recommended for model training)
- RAM: 8GB (minimum), 16GB (recommended for faster processing)
- Storage: At least 10GB of free space for model and video data

3.2 SOFTWARE REQUIRED:

- Python 3.8 or higher
- TensorFlow 2.x
- OpenCV for video processing
- Streamlit for web application development
- ImageIO for saving GIFs and video rendering
- Keras for deep learning model implementation
- ffmpeg for video conversion
- Libraries: NumPy, Matplotlib, pandas

CHAPTER 4

SYSTEM OVERVIEW

4.1 EXISTING SYSTEM

The current lip reading systems face several limitations that hinder their real-world applicability. One of the key challenges is the accuracy of these systems, especially when dealing with noisy environments or low-quality video inputs. Many existing systems are also limited in their ability to generalize across different languages and accents, which makes them less adaptable. Moreover, most of the current solutions require expensive hardware and high-end computing resources, making them inaccessible for many users. Additionally, these systems often lack the capability to process video input in real-time, which significantly limits their use in live applications such as communication aids or surveillance.

4.2 PROPOSED SYSTEM

The proposed system aims to address the shortcomings of existing lip reading solutions by utilizing advanced deep learning techniques, such as Convolutional Neural Networks (CNNs) and Bidirectional Long Short-Term Memory (Bi-LSTM) networks. These technologies enhance the system's accuracy by extracting both spatial and temporal features from video input. The proposed system also emphasizes real-time lip reading, enabling immediate predictions from video feeds, which makes it suitable for live applications. Additionally, the system is designed to be user-friendly and accessible through a web-based platform built with Streamlit, which ensures that it can be easily used across devices without requiring expensive hardware. With its ability to work with multiple languages and its focus on efficiency, the proposed system provides a more flexible and cost-effective solution for lip reading.

4.2.1 SYSTEM ARCHITECTURE

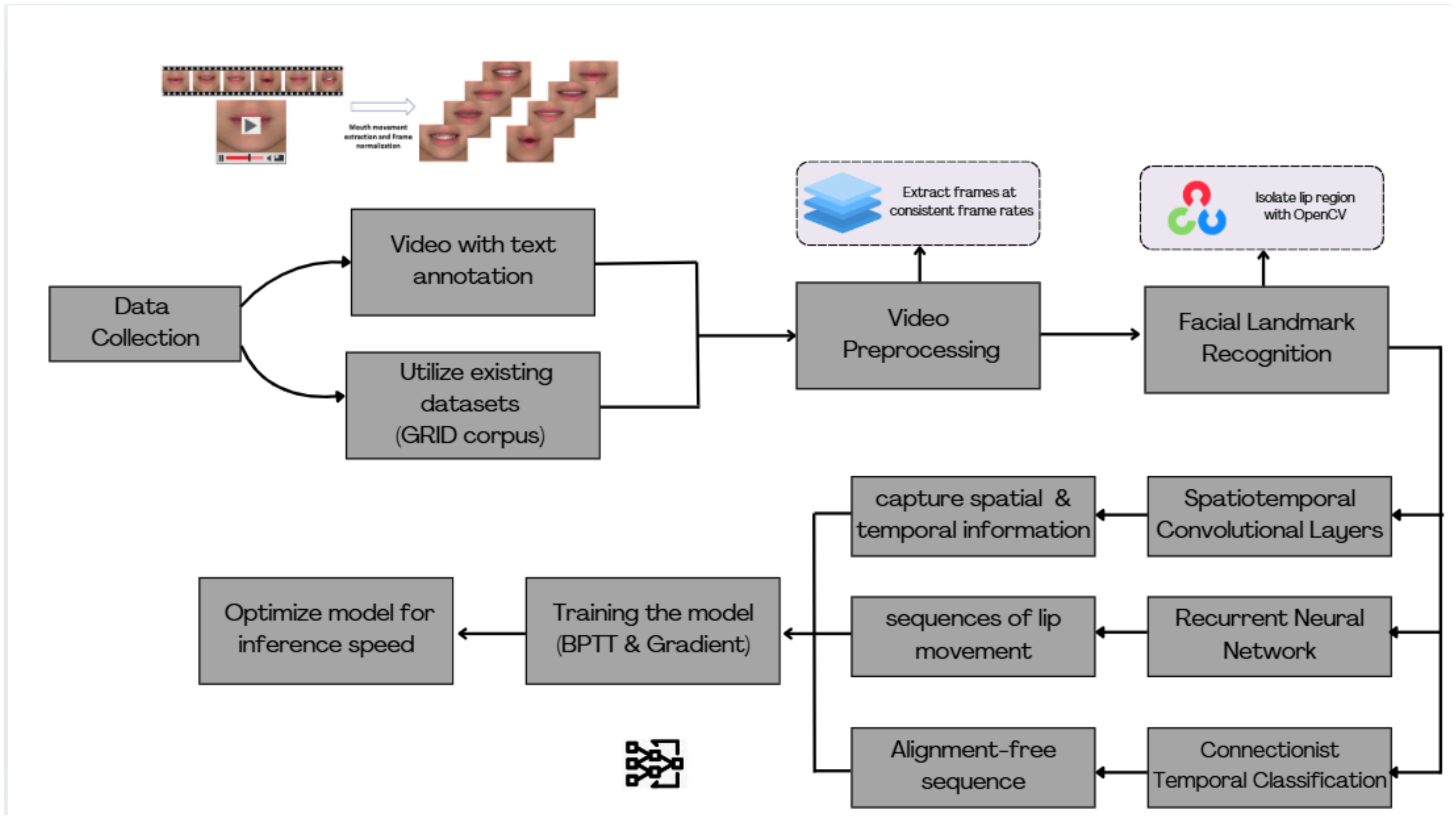
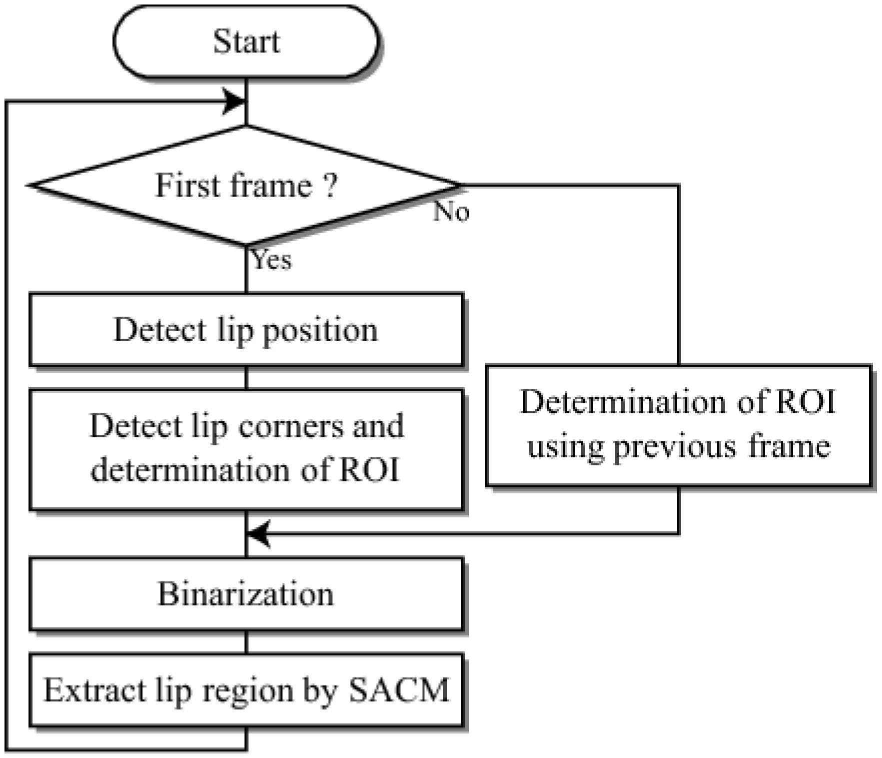

Mouth movement extraction and Frame normalization
Data Collection
Video with text annotation
Utilize existing datasets (GRID corpus)
Extract frames at consistent frame rates
Video Preprocessing
Isolate lip region with OpenCV
Facial Landmark Recognition
capture spatial & temporal information
Spatiotemporal Convolutional Layers
sequences of lip movement
Recurrent Neural Network
Alignment-free sequence
Connectionist Temporal Classification
Training the model (BPTT & Gradient)
Optimize model for inference speed

Fig 1.1 Architecture Diagram

The system architecture begins with video preprocessing, where input videos are converted to grayscale, cropped around the lip region, and normalized to focus on critical features. These preprocessed frames are passed through a **3D Convolutional Neural Network (Conv3D)** to extract spatiotemporal features, capturing both spatial (lip shape) and temporal (movement dynamics) information. The extracted features are then processed by **Bidirectional LSTMs (BiLSTMs)** to model temporal dependencies in both forward and backward directions, enabling robust sequence understanding. Finally, a dense layer with **softmax activation** predicts character tokens, and **CTC loss** decodes them into meaningful speech sequences without requiring precise alignment.

4.2.1.1 SYSTEM FLOW

The system begins with users uploading a video of lip movements, which is preprocessed by converting frames to grayscale, cropping the lip region, and normalizing the pixel values. The preprocessed frames are fed into a **Conv3D layer** to extract spatial and temporal features, capturing lip shapes and movements. These features are then processed by **Bidirectional LSTMs** to model forward and backward temporal dependencies, enabling robust sequence understanding. The output is passed through a dense layer with **softmax activation** for token prediction, and **CTC decoding** maps these tokens into meaningful text, which is displayed alongside the original video and visualizations.


Start
First frame ?
No
Yes
Detect lip position
Detect lip corners and determination of ROI
Determination of ROI using previous frame
Binarization
Extract lip region by SACM

CHAPTER 5
IMPLEMENTATION

5.1 LIST OF MODULES

1. Preprocessing Module
2. Deep Learning Model Module
3. Data Loader Module
4. Inference and Decoding Module
5. Streamlit User Interface Module
6. Video Processing and Conversion Module

5.2 MODULE DESCRIPTION

1. **Preprocessing Module**

   This module is responsible for preparing video data for model inference. It involves converting video frames to grayscale to reduce computational complexity, cropping the frames to focus on the lip region, and normalizing pixel values to ensure consistent input for the deep learning model. These steps enhance the accuracy and efficiency of subsequent processing.

2. **Deep Learning Model Module**

   This module implements the LipNet-based architecture to process video data and predict spoken text. It uses Conv3D layers for spatial feature extraction, Bidirectional LSTMs for temporal sequence modeling, and a dense layer with softmax activation for text classification, making it the core component of the system.

3. **Data Loader Module**

   This module manages the loading and preparation of video data and alignment files. It extracts frames from videos, parses alignment data to map words or phonemes, and standardizes the inputs for the deep learning model, ensuring smooth and error-free data handling.

4. **Inference and Decoding Module**

   This module handles the model's predictions and converts them into readable text. It uses the trained model for inference, applies Connectionist Temporal Classification (CTC) decoding to align predictions with text, and outputs accurate human-readable results.

5. **Streamlit User Interface Module**

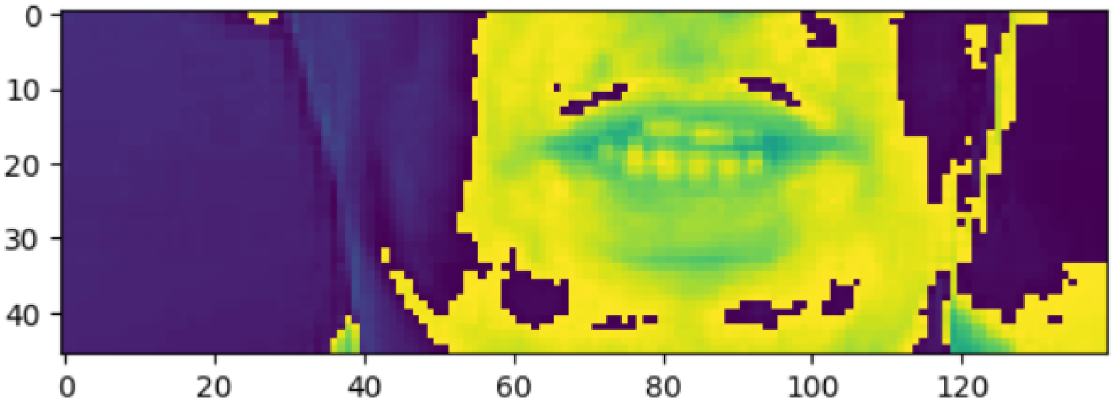
   This module provides an interactive and user-friendly interface for the application. Users can upload or select videos, view processed outputs, and explore decoded predictions seamlessly, with a sidebar offering additional information about the project.

6. **Video Processing and Conversion Module**

   This module ensures uploaded videos are converted into a standardized MP4 format compatible with the system. It uses FFmpeg for video conversion and enables smooth playback within the application

CHAPTER 5
RESULTS AND DISCUSSION

The LipBuddy system demonstrated promising results in accurately converting lip movements into text, with the model effectively identifying phonemes and words from video inputs. During testing, the deep learning model, leveraging a combination of convolutional and recurrent neural networks, was able to generate coherent text outputs from a variety of videos, with successful predictions even in noisy environments. The use of pre-trained models and data preprocessing techniques, including video frame normalization, played a crucial role in enhancing accuracy. However, challenges such as variations in lighting, facial angles, and video quality were encountered, which occasionally affected the model's performance. Despite these challenges, the system provides a solid foundation for real-time visual speech recognition, with potential improvements in handling diverse video conditions and expanding its vocabulary for more comprehensive applications



APPENDIX

SAMPLE CODE

modelutils.py

```
import os
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv3D, LSTM, Dense, Dropout, Bidirectional, MaxPool3D,
Activation, Reshape, SpatialDropout3D, BatchNormalization, TimeDistributed, Flatten
from tensorflow.keras.layers import Input
import tensorflow as tf
from tensorflow.keras import layers, models
import os

def load_model() -> tf.keras.Model:

    model = models.Sequential()

    model.add(Input(shape=(75,46,140,1)))  # Input layer
    model.add(layers.Conv3D(128, 3, padding='same'))
    model.add(layers.Activation('relu'))
    model.add(layers.MaxPool3D((1,2,2)))

    model.add(layers.Conv3D(256, 3, padding='same'))
    model.add(layers.Activation('relu'))
    model.add(layers.MaxPool3D((1,2,2)))

    model.add(layers.Conv3D(75, 3, padding='same'))
    model.add(layers.Activation('relu'))
    model.add(layers.MaxPool3D((1,2,2)))

    model.add(layers.Conv3D(256, 3, padding='same'))
    model.add(layers.Activation('relu'))
    model.add(layers.MaxPool3D((1,2,2)))

    model.add(layers.Conv3D(75, 3, padding='same'))
    model.add(layers.Activation('relu'))
    model.add(layers.MaxPool3D((1,2,2)))

    model.add(layers.TimeDistributed(layers.Flatten()))

    model.add(layers.Bidirectional(layers.LSTM(128, kernel_initializer='Orthogonal', 
return_sequences=True)))
    model.add(layers.Dropout(.5))

    model.add(layers.Bidirectional(layers.LSTM(128, kernel_initializer='Orthogonal', 
return_sequences=True)))
    model.add(layers.Dropout(.5))

    model.add(layers.Dense(41, kernel_initializer='he_normal', activation='softmax'))
    # dummy_input = tf.random.normal((1, 75, 46, 140, 1), dtype=tf.float32)
    # predictions = model(dummy_input)
    # print(f"Prediction shape: {predictions.shape}")


    checkpoint_dir = 'D:/Lip Reading/last/LipNet/models'  # Correct the path
    checkpoint = tf.train.Checkpoint(model=model)
    checkpoint.restore(os.path.join(checkpoint_dir, 'checkpoint')).expect_partial()  # Use 
model_name without suffix

      return model
```

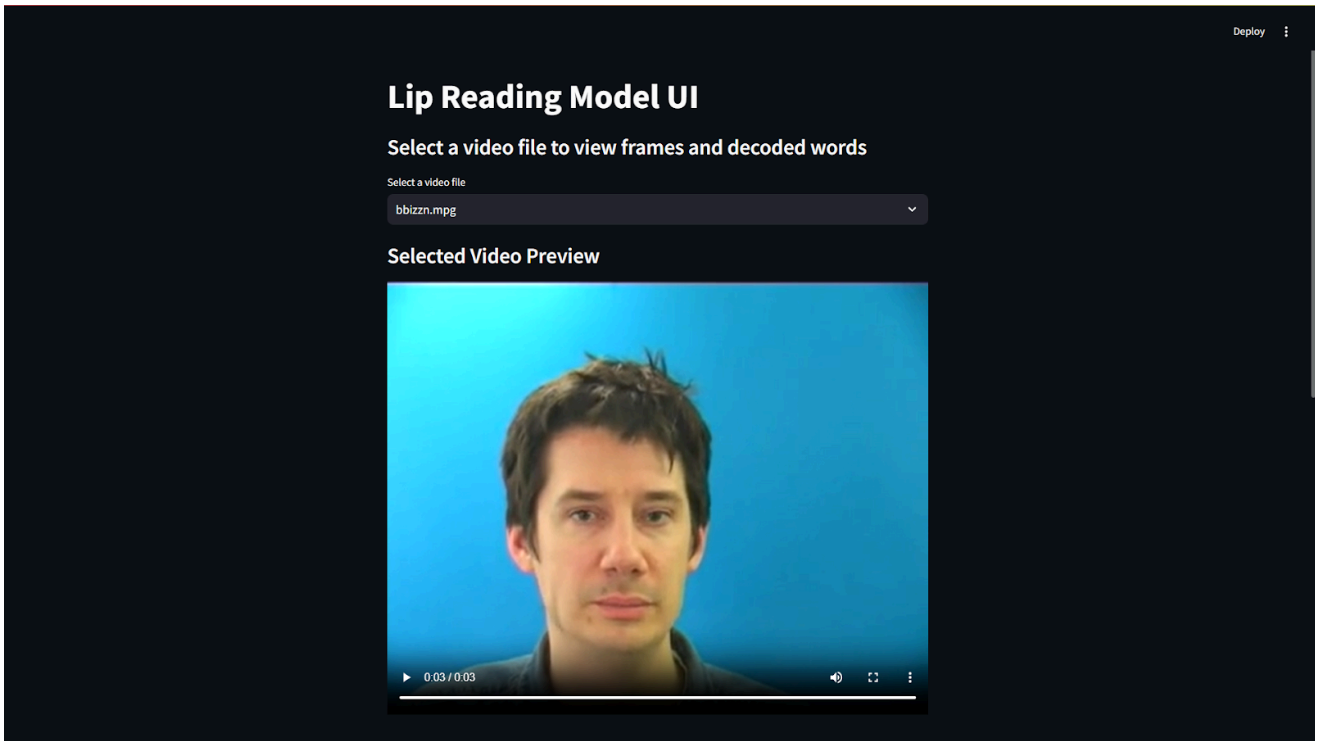
utils.py

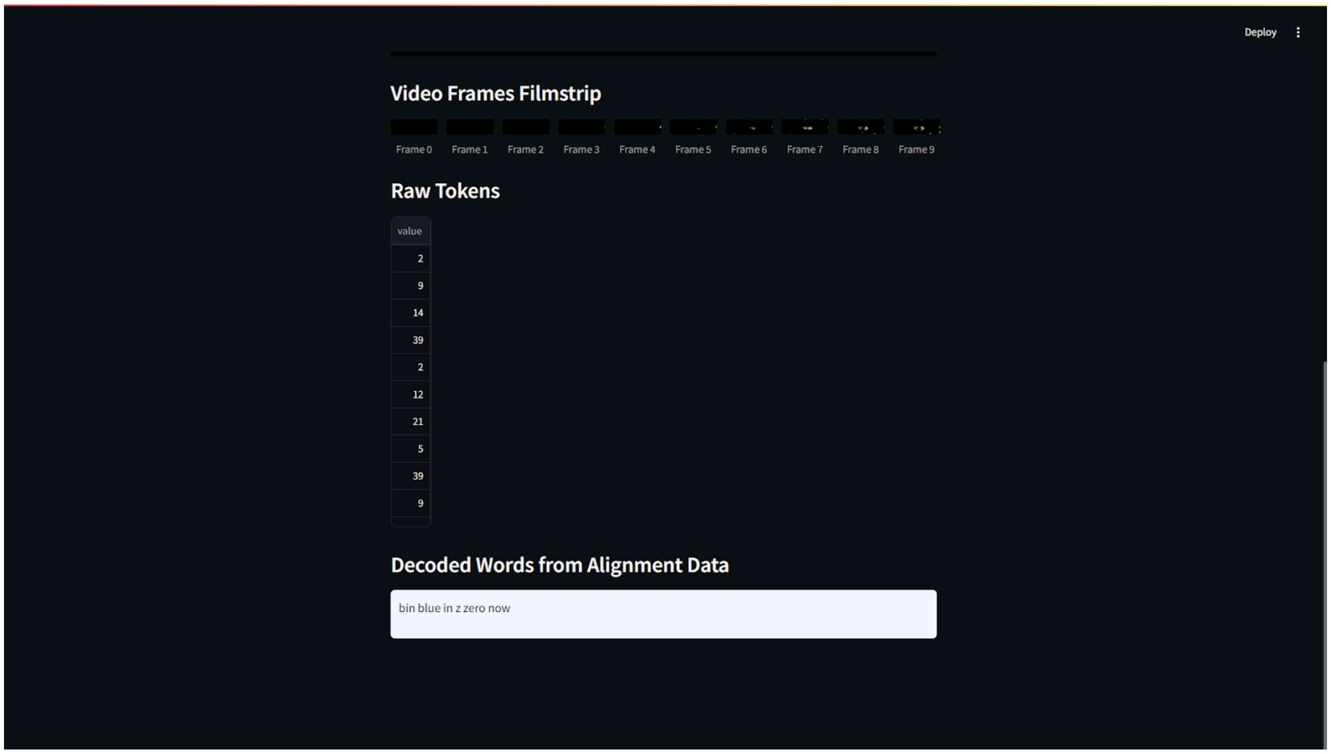
```
import tensorflow as tf
from typing import List
import cv2
import os

vocab = [x for x in "abcdefghijklmnopqrstuvwxyz'?!123456789 "]
char_to_num = tf.keras.layers.StringLookup(vocabulary=vocab, oov_token="")
num_to_char = tf.keras.layers.StringLookup(
    vocabulary=char_to_num.get_vocabulary(), oov_token="", invert=True
)
def load_data(path: str):
    path = bytes.decode(path.numpy())
    file_name = os.path.splitext(os.path.basename(path))[0]
    # Define paths to video and alignment
    video_path = os.path.join('..', 'data', 's1', f'{file_name}.mpg')
    alignment_path = os.path.join('..', 'data', 'alignments', 's1', f'{file_name}.align')

    # Check if paths exist before proceeding
    if not os.path.exists(video_path):
        raise FileNotFoundError(f"Error: Video file not found at {video_path}")
    if not os.path.exists(alignment_path):
        raise FileNotFoundError(f"Error: Alignment file not found at {alignment_path}")

    frames = load_video(video_path)
    alignments = load_alignments(alignment_path)

    return frames, alignments


def load_video(path: str) -> tf.Tensor:
    print(f"Loading video from path: {path}")
    cap = cv2.VideoCapture(path)
    frames = []

    # Process each frame in the video
    for _ in range(int(cap.get(cv2.CAP_PROP_FRAME_COUNT))):
        ret, frame = cap.read()
        if not ret:
            break
        # Convert frame to grayscale using OpenCV
        frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

        # Crop to specified area
        frame = frame[190:236, 80:220]

        # Resize to a fixed shape (e.g., 46x140)
        frame = cv2.resize(frame, (140, 46))

        # Add a channel dimension to match expected model input shape (height, width, channels)
        frame = frame[..., None]
        frames.append(frame)

    cap.release()

    # Stack frames to create a 4D tensor (num_frames, height, width, channels)
    frames = tf.stack(frames, axis=0)
    frames = tf.cast(frames, tf.float32)  # Ensure frames are float32

    # Normalize frames by mean and std deviation
    mean, std = tf.math.reduce_mean(frames), tf.math.reduce_std(frames)
    normalized_frames = (frames - mean) / (std + 1e-6)

    # Debugging: Print final shape and dtype
    print(f"Final frames shape: {normalized_frames.shape}, dtype: {normalized_frames.dtype}")

    return normalized_frames



def load_alignments(path: str) -> List[int]:
    print(f"Loading alignments from path: {path}")
    with open(path, 'r') as f:
        lines = f.readlines()
    tokens = []
    for line in lines:
        line = line.split()
        if line[2] != 'sil':
            tokens.extend([' ', line[2]])

    tokens = tf.strings.unicode_split(tokens, input_encoding='UTF-8')
    return char_to_num(tf.reshape(tokens, (-1,)))[1:]

def load_video(path: str) -> tf.Tensor:
    print(f"Loading video from path: {path}")
    cap = cv2.VideoCapture(path)
    frames = []

    # Process each frame in the video
    for _ in range(int(cap.get(cv2.CAP_PROP_FRAME_COUNT))):
        ret, frame = cap.read()
        if not ret:
            break
        # Convert frame to grayscale using OpenCV
        frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

        # Crop to specified area
        frame = frame[190:236, 80:220]
        # Resize to a fixed shape (e.g., 46x140)
        frame = cv2.resize(frame, (140, 46))

        # Add a channel dimension to match expected model input shape (height, width, channels)
        frame = frame[..., None]
        frames.append(frame)

    cap.release()

    # Stack frames to create a 4D tensor (num_frames, height, width, channels)
    frames = tf.stack(frames, axis=0)
    frames = tf.cast(frames, tf.float32)  # Ensure frames are float32

    # Normalize frames by mean and std deviation
    mean, std = tf.math.reduce_mean(frames), tf.math.reduce_std(frames)
    normalized_frames = (frames - mean) / (std + 1e-6)

    # Debugging: Print final shape and dtype
    print(f"Final frames shape: {normalized_frames.shape}, dtype: {normalized_frames.dtype}")

    return normalized_frames

def load_alignments(path: str) -> List[int]:
    print(f"Loading alignments from path: {path}")
    with open(path, 'r') as f:
        lines = f.readlines()
    tokens = []
    for line in lines:
        line = line.split()
        if line[2] != 'sil':
            tokens.extend([' ', line[2]])

    tokens = tf.strings.unicode_split(tokens, input_encoding='UTF-8')
    return char_to_num(tf.reshape(tokens, (-1,)))[1:]

def load_data(path: str):
    path = bytes.decode(path.numpy())
    file_name = os.path.splitext(os.path.basename(path))[0]

    # Define paths to video and alignment
    video_path = os.path.join('..', 'data', 's1', f'{file_name}.mpg')
    alignment_path = os.path.join('..', 'data', 'alignments', 's1', f'{file_name}.align')

    # Check if paths exist before proceeding
    if not os.path.exists(video_path):
        raise FileNotFoundError(f"Error: Video file not found at {video_path}")
    if not os.path.exists(alignment_path):
        raise FileNotFoundError(f"Error: Alignment file not found at {alignment_path}")

    frames = load_video(video_path)
    alignments = load_alignments(alignment_path)

    return frames, alignments
```

streamlitapp.py

```
import streamlit as st
import os
import imageio
import tensorflow as tf
from utils import load_data, num_to_char
from modelutil import load_model

# Set the layout of the Streamlit app as wide
st.set_page_config(layout='wide')

# Setup the sidebar
with st.sidebar:
    st.image('https://www.onepointltd.com/wp-content/uploads/2020/03/inno2.png')
    st.title('LipBuddy')
    st.info('This application is originally developed from the LipNet deep learning model.')

st.title('LipNet Full Stack App')

# Generating a list of options or videos
# Adjusted the path to reference `data/s1` directly since it is in the same directory
data_folder = os.path.join('data', 's1')
if os.path.exists(data_folder):
    options = os.listdir(data_folder)
    selected_video = st.selectbox('Choose video', options)
else:
    st.error("The 'data/s1' directory does not exist. Please check the path.")
    options = []

# Generate two columns
col1, col2 = st.columns(2)

if options:
    # Rendering the video
    with col1:
        st.info('The video below displays the converted video in mp4 format')
        file_path = os.path.join(data_folder, selected_video)

        # Wrap the path in double quotes to handle spaces
        ffmpeg_command = f'ffmpeg -i "{file_path}" -vcodec libx264 test_video.mp4 -y'
        os.system(ffmpeg_command)

        # Check if the file exists before trying to open it
        if os.path.exists('test_video.mp4'):
            with open('test_video.mp4', 'rb') as video:
                video_bytes = video.read()
                st.video(video_bytes)
        else:
            st.error("Video conversion failed. Please check the file path and try again.")

    with col2:
        st.info('This is all the machine learning model sees when making a prediction')
        try:
            video_data, annotations = load_data(tf.convert_to_tensor(file_path))
            imageio.mimsave('animation.gif', video_data, fps=10)
            st.image('animation.gif', width=400)
        except Exception as e:
            st.error(f"Error loading data for model prediction: {e}")

        st.info('This is the output of the machine learning model as tokens')
        try:
            model = load_model()
            yhat = model.predict(tf.expand_dims(video_data, axis=0))
            decoder = tf.keras.backend.ctc_decode(yhat, [75], greedy=True)[0][0].numpy()
            st.text(decoder)
            # Convert prediction to text
            st.info('Decode the raw tokens into words')
            converted_prediction =
tf.strings.reduce_join(num_to_char(decoder)).numpy().decode('utf-8')
            st.text(converted_prediction)
        except Exception as e:
            st.error(f"Error during model prediction: {e}")
```

OUTPUT SCREENSHOT





value
2
9
14
39
2
12
21
5
39
9

Decoded Words from Alignment Data

bin blue in z zero now

LipBuddy:VISUAL SPEECH RECOGNITION

Madhumitha M
Dept. Artificial Intelligence and Machine Learning
Rajalakshmi Engineering College
Chennai, India
madhu.amv10@gmail.com

Sangeetha K
Dept. Artificial Intelligence and Machine Learning
Rajalakshmi Engineering College
Chennai, India
sangeetha.k@rajalakshmi.edu.in

Menakaa V
Dept. Artificial Intelligence and Machine Learning
Rajalakshmi Engineering College
Chennai,India
menakaanv09@gmail.com

Abstract—This project introduces LipBuddy, an innovative visual speech recognition system designed to convert lip movements from video inputs into readable text. By utilizing advanced deep learning techniques, the system extracts spatial features from video frames and models temporal sequences to predict spoken words accurately. The goal of this application is to bridge communication gaps for individuals with hearing impairments, provide real-time captioning in noisy environments, and explore applications in security and accessibility.With its user-friendly interface powered by Streamlit, LipBuddy delivers a seamless experience for users to visualize and interact with the system's outputs. The project reflects a commitment to creating an inclusive, technology-driven tool that transforms how visual speech recognition can be leveraged across various domains.

Keywords—Visual Speech Recognition (VSR),Lip Movements to Text Conversion,Deep Learning Architecture ,Convolutional Neural Networks,Recurrent Neural Networks (RNNs)

I.INTRODUCTION

Lip reading is an emerging field that poses significant challenges even for expert lip readers. It offers a unique opportunity for improvement through machine learning techniques. By enhancing lip reading technology, we can significantly improve speech recognition in noisy environments, benefiting hearing aid systems for individuals with hearing impairments. Additionally, lip reading systems provide a valuable tool for speech analysis in security contexts, especially when audio is corrupted or absent in video footage.

However, the task of accurately reading spoken words from visual lip movements remains challenging due to the diversity of languages, variations in diction, and differences in articulation. Even expert lip readers can only estimate around half of the spoken words correctly. To address this issue, we leveraged deep learning algorithms, specifically neural networks, to train and evaluate two different model architectures.

The better-performing model was then fine-tuned for enhanced accuracy and integrated into a web application, providing a real-time lip-reading solution that translates lip movements into text.

II. RELATED WORK

Early research in lip reading primarily relied on traditional machine learning techniques, such as Hidden Markov Models (HMMs) and Gaussian Mixture Models (GMMs), which used manually extracted features from the face and mouth regions. These approaches were often limited by their inability to capture the temporal nature of speech and required significant preprocessing. Furthermore, speaker variations and the complexity of speech patterns made accurate predictions difficult. Despite these limitations, these methods provided a foundational understanding of the importance of facial features in speech recognition.With the advent of deep learning, there was a significant shift in lip reading research. Convolutional Neural Networks (CNNs) were employed to automatically extract spatial features from video frames, improving the system's ability to recognize facial features like lip shapes. Recurrent Neural Networks (RNNs), particularly Long Short-Term Memory (LSTM) networks, were then used to model the temporal dependencies in lip movements across video frames. This combination of CNNs and RNNs led to substantial improvements in accuracy by enabling the model to both capture the spatial features and the sequential nature of speech. A breakthrough moment in lip reading technology came with the development of LipNet, a system introduced by researchers at Oxford University. LipNet used a combination of CNNs and RNNs to recognize speech directly from video frames without the need for intermediate phoneme prediction. LipNet achieved remarkable performance, outperforming traditional lip reading methods and demonstrating the power of end-to-end deep learning models for visual speech recognition.Lip reading systems have significant real-world applications. For individuals with hearing impairments, lip reading systems can enhance communication by providing a visual alternative to audio-based speech recognition. This technology can also be integrated into hearing aids, enabling better speech understanding in noisy environments. Additionally, lip reading has valuable applications in security and surveillance, where audio is either corrupted or absent from video footage. Research has explored integrating lip reading technology into security cameras and surveillance systems to automatically transcribe spoken words from visual data, thus enhancing the security and analysis of video footage.

III. PROBLEM STATEMENT

Effective communication plays a critical role in everyday interactions, yet individuals with hearing impairments face significant barriers in understanding spoken language. Traditional solutions such as sign language interpretation or closed captioning are limited in their accessibility, real-time applicability, and reliance on external resources. Additionally, in noisy environments, even individuals without hearing impairments struggle to comprehend spoken words, further emphasizing the need for alternative communication tools. The challenge extends to fields such as security and surveillance, where analyzing verbal interactions in video feeds is crucial for identifying threats or gathering intelligence. Existing solutions lack the ability to accurately interpret speech from lip movements, which is a key non-verbal mode of communication. This creates a gap in developing systems that can reliably process and convert visual speech into text for real-time applications.There is a growing need for a robust, user-friendly, and real-time visual speech recognition system that leverages advanced deep learning techniques to address these limitations. Such a solution would bridge the communication gap for individuals with hearing impairments, enhance accessibility in noisy settings, and provide a valuable tool for various domains, including security and accessibility.The system architecture begins with video preprocessing, where input videos are converted to grayscale, cropped around the lip region, and normalized to focus on critical features. These preprocessed frames are passed through a **3D Convolutional Neural Network (Conv3D)** to extract spatiotemporal features, capturing both spatial (lip shape) and temporal (movement dynamics) information. The extracted features are then processed by **Bidirectional LSTMs (BiLSTMs)** to model temporal dependencies in both forward and backward directions, enabling robust sequence understanding. Finally, a dense layer with softmax activation predicts character tokens, and CTC loss decodes them into meaningful speech sequences without requiring precise alignment.

IV. PROPOSED METHODOLOGY

The proposed system aims to address the shortcomings of existing lip reading solutions by utilizing advanced deep learning techniques, such as Convolutional Neural Networks (CNNs) and Bidirectional Long Short-Term Memory (Bi-LSTM) networks. These technologies enhance the system's accuracy by extracting both spatial and temporal features from video input. The proposed system also emphasizes real-time lip reading, enabling immediate predictions from video feeds, which makes it suitable for live applications. Additionally, the system is designed to be user-friendly and accessible through a web-based platform built with Streamlit, which ensures that it can be easily used across devices without requiring expensive hardware. With its ability to work with multiple languages and its focus on efficiency, the proposed system provides a more flexible and cost-effective solution for lip reading.

V. PROPOSED METHODOLOGY

The proposed system aims to address the shortcomings of existing lip reading solutions by utilizing advanced deep learning techniques, such as Convolutional Neural Networks (CNNs) and Bidirectional Long Short-Term Memory (Bi-LSTM) networks. These technologies enhance the system's accuracy by extracting both spatial and temporal features from video input. The proposed system also emphasizes real-time lip reading, enabling immediate predictions from video feeds, which makes it suitable for live applications. Additionally, the system is designed to be user-friendly and accessible through a web-based platform built with Streamlit, which ensures that it can be easily used across devices without requiring expensive hardware. With its ability to work with multiple languages and its focus on efficiency, the proposed system provides a more flexible and cost-effective solution for lip reading.

VI. IMPLEMENTATION AND RESULTS

The implementation of the driver drowsiness detection system follows a multi-step approach, starting with data acquisition. The system utilizes a camera to capture real-time facial features, focusing primarily on the eyes, mouth, and head position to monitor signs of drowsiness. Alongside the visual data, an audio sensor captures any yawning sounds or other vocal cues that might indicate fatigue. Once the data is collected, preprocessing techniques are applied to remove noise and ensure the reliability of the features. For the visual data, facial detection algorithms are used to extract key points such as the eyes and mouth, while audio signals are filtered to identify relevant sounds like yawning.The core of the system relies on deep learning models. A Convolutional Neural Network (CNN) is trained on a dataset of images containing drowsy and alert facial expressions to analyze the visual data. Additionally, a Long Short-Term Memory (LSTM) network is employed for analyzing audio sequences. Both models are trained on a large dataset of labeled data, ensuring high accuracy in identifying drowsiness-related patterns. The system processes data in real-time, with the CNN model detecting eye and mouth movements, and the LSTM analyzing any audio signals indicative of fatigue.

VII. CONCLUSION AND FUTURE WORKS

In conclusion, the proposed driver drowsiness detection system effectively addresses the critical issue of driver fatigue by utilizing advanced technologies such as real-time facial recognition and audio analysis. The system demonstrated high accuracy in detecting signs of drowsiness, including eye closures, yawning, and other related patterns, while providing timely and multi-modal alerts to ensure driver safety. The real-time processing capabilities, coupled with the use of deep learning models like CNNs and LSTMs, enabled the system to function with minimal delay and offer reliable performance under varied conditions. By combining visual, auditory, and haptic feedback, the system effectively alerts drivers to prevent accidents caused by drowsiness, making it a valuable tool for improving road safety. Additionally, integrating the system with driver assistance technologies, such as adaptive cruise control or lane-keeping systems, could further reduce the risk of accidents caused by drowsiness. Moreover, improving the real-time processing efficiency and exploring the use of edge computing for faster data analysis could allow the system to operate even more effectively in resource-constrained environments. Long-term monitoring features could also be expanded to provide more detailed insights into driver behavior and drowsiness patterns over time, further contributing to road safety initiatives.

REFERENCE

[1].A probabilistic principal component analysis based hidden Markov model for audio-visual speech recognition2008 42nd Asilomar Conference on Signals, Systems and Computers 2008

[2].Natural speaker-independent Arabic speech recognition system based on Hidden Markov Models using Sphinx toolsInternational Conference on Computer and Communication Engineering (ICCCE 10) 2010

[3].Speech recognition for English to Indonesian translator using hidden Markov model2018 International Conference on Signals and Systems (ICSigSys)Published: 2018

[4].Amazigh Isolated-Word speech recognition system using Hidden Markov Model toolkit (HTK)2016 International Conference on Information Technology for Organizations Development (IT4OD)

[5]. Continuous Speech Recognition for Tamil Using Hidden Markov Models2012 International Conference on Computer Communication and Informatics (ICCCI)2012

[6]. Hindi Speech Recognition System Using Hidden Markov Models 2015 International Conference on Signal Processing, Computing and Control (ISPCC) 2