Lip Reading using Deep Learning

By:

Madhav

Sonu Kumar

Gyaneshwer Jha

Jodgudri Pratik Shivamurti

1. Introduction

1.1 Project Overview

This project endeavors to develop a lip reading system employing a CNN-LSTM architecture to transcribe spoken words accurately from visual lip movements in user-selected videos. Focused on enhancing communication accessibility, the system is designed to allow users to choose from a set of videos for transcription rather than real-time processing. The methodology involves collecting and preprocessing a diverse dataset, implementing a CNN-LSTM model architecture to capture both spatial and temporal features, and optimizing its performance through training and validation. The expected outcomes include achieving high accuracy in lip reading, offering a versatile tool for users to select videos for transcription, and ensuring the scalability of the solution. This project aims to contribute to the field of assistive technologies, providing a valuable resource for individuals with hearing impairments seeking transcription services for prerecorded content.

1.2 Purpose

The central goal of this project is to develop an accurate lip reading system utilizing a specialized CNN-LSTM architecture. The project focuses on transcribing spoken words from a selected set of videos, catering to scenarios where content is prerecorded. The primary purpose is to significantly enhance communication accessibility for individuals with hearing impairments. Leveraging deep learning techniques, particularly CNN-LSTM, the project aims to achieve a high level of accuracy in lip reading by effectively capturing both spatial and temporal features from visual lip movements. This endeavor contributes to the field of assistive technologies, providing a valuable tool for accurate transcription of spoken content from prerecorded videos and addressing specific communication needs within the hearing-impaired community.

2. LITERATURE SURVEY

2.1 Existing problem

- Variability in Lip Movements:
  - Lip movements exhibit inherent variability across individuals, accents, and languages, posing a challenge for lip reading systems to generalize effectively. Addressing this issue involves devising strategies to enhance model adaptability, ensuring accurate performance in diverse scenarios.
- Limited Datasets and Benchmarks:
  - The scarcity of comprehensive lip reading datasets hampers the development of robust models. Investigating existing datasets, and their limitations, and proposing methodologies for creating more extensive benchmarks is crucial for evaluating lip reading model performance accurately.
- Cross-Modal Challenges:
  - Integrating visual lip movements with other modalities, such as audio cues, is complex but critical for improving lip reading accuracy. Research in this area focuses on effective cross-modal fusion techniques to broaden the applicability of lip reading models.
- Real-world Conditions:
  - Lip reading models often struggle in real-world conditions with factors like background noise and varying lighting. Exploring strategies to enhance model robustness ensures reliable performance in practical and challenging scenarios, addressing a key limitation in current lip reading technology.

2.2 References

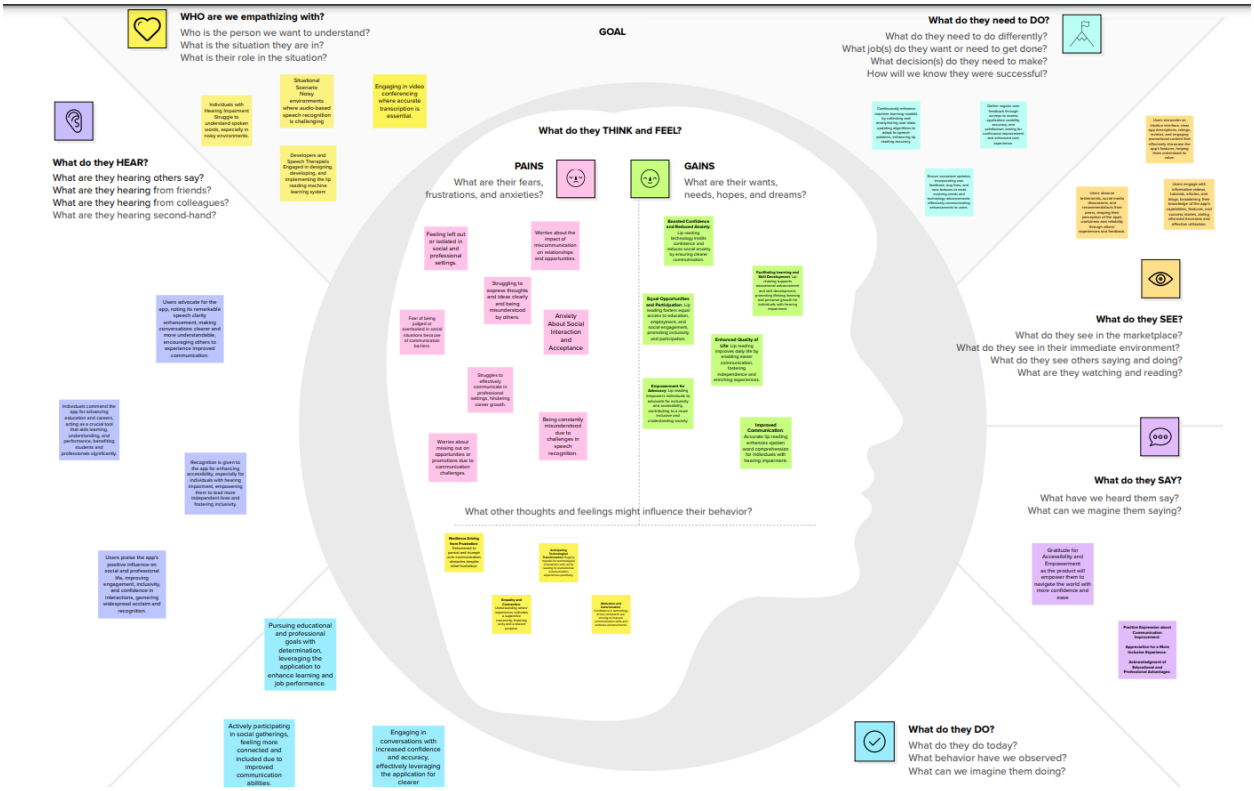
- LipNet: End-to-End Sentence-level Lipreading (https://arxiv.org/abs/1611.01599)
- https://youtu.be/uKyojQjbx4c?si=zhO5xOdnLjPIffuW
- https://youtu.be/FMPYShYLg3o?si=9FKOk-DlK1sHwmph

2.3 Problem Statement Definition

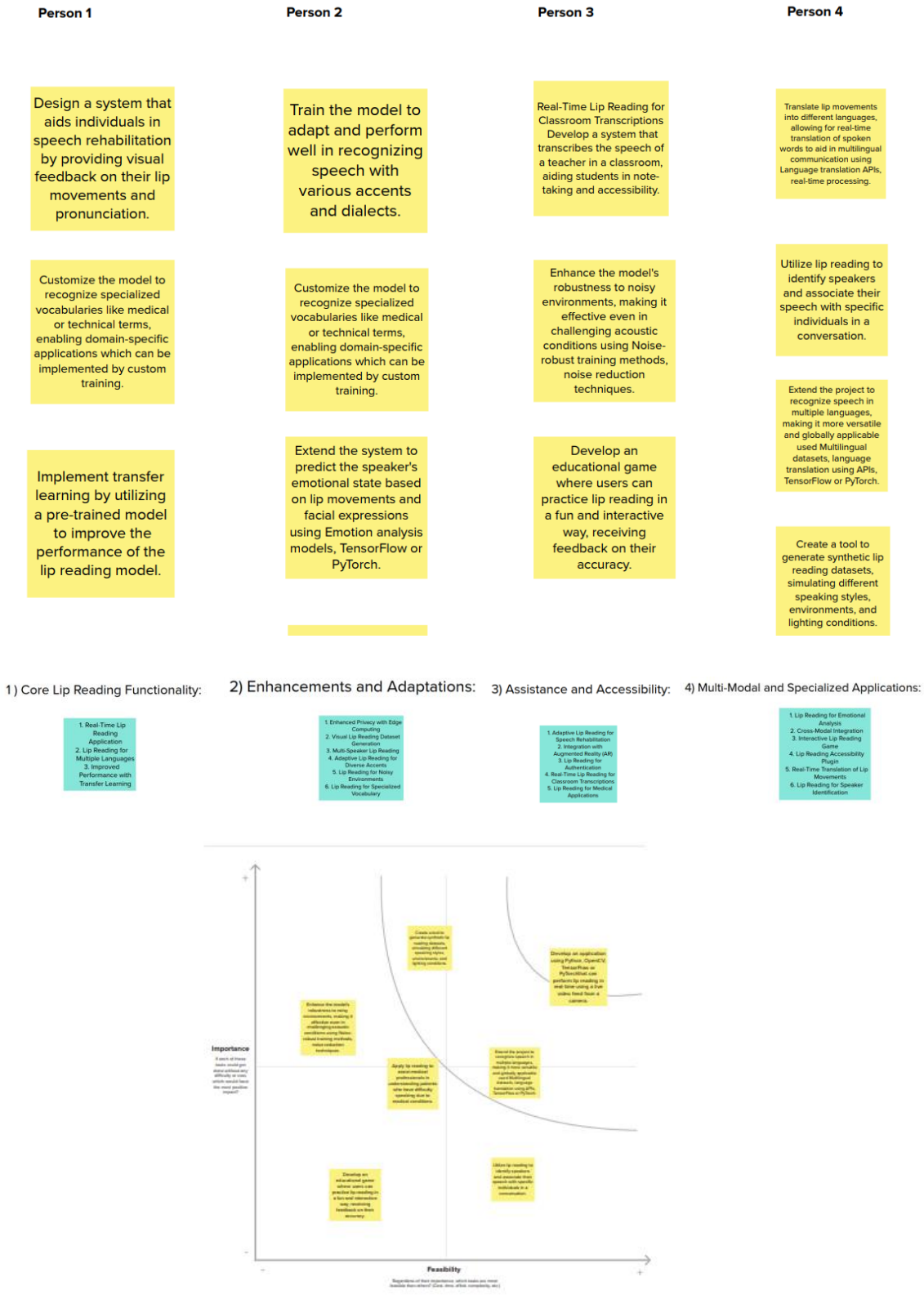
The objective of this project is to develop an end-to-end machine learning solution to detect words from a video of a person speaking. The proposed solution involves the use of Deep learning algorithms like LSTM, Neural Networks to predict the accurate output.

3. IDEATION & PROPOSED SOLUTION

3.1 Empathy Map Canvas



3.2 Ideation & Brainstorming



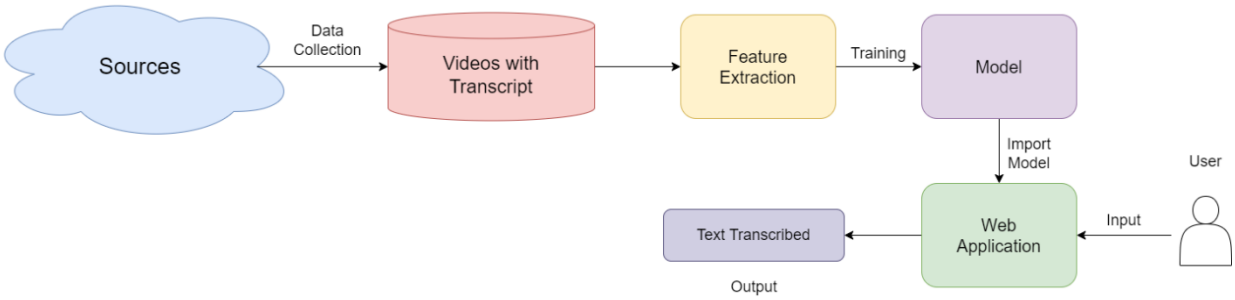
4. REQUIREMENT ANALYSIS

4.1 Functional requirement

'Google Colab' is a very good platform to run the model. It can be run on any computer with an internet connection.

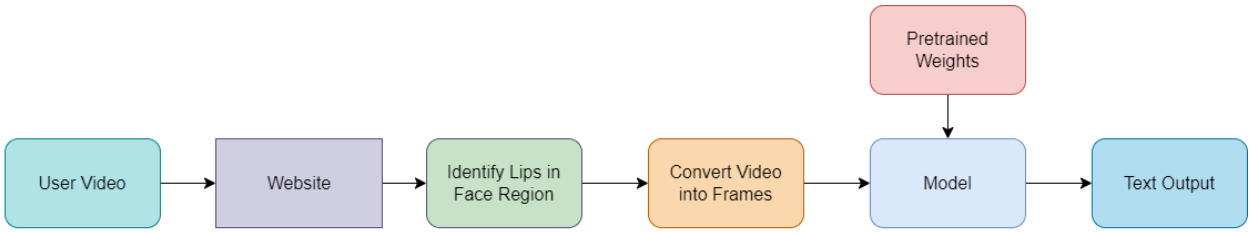
5. PROJECT DESIGN

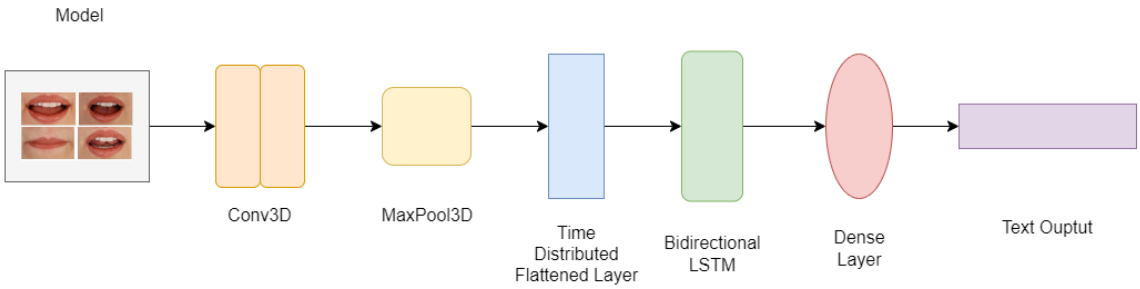
5.1 Data Flow Diagrams & User Stories



User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance Criteria	Priority
User with a hearing impairment	The system should allow users to upload video content for lip reading analysis	USN - 1	As a user with a hearing impairment, I want to upload a video containing spoken words to the system for accurate transcriptions.	The system must provide an option to upload video files. Uploaded videos should be processed for lip reading. The system must return a text transcription of the spoken words in the video.	High
User of the lip reading system	The system should provide a user-friendly website for interaction.	USN - 2	As a user of the lip reading system, I want to access and use the service through a user-friendly website, enabling easy and intuitive interactions.	The website should have a clear and intuitive user interface. Users should be able to navigate and use the website without difficulty. The website should provide clear instructions and controls for video input.	High
Parent or caregiver of a non-verbal child	The system should be accessible to children and provide accurate transcriptions.	USN – 3	As a parent or caregiver of a non-verbal child, I want the system to be accessible to children and provide accurate transcriptions for their benefit.	The system should have a child-friendly user interface and controls. Children should be able to use the system with minimal adult assistance. The system should accurately transcribe spoken words to assist non-verbal children in understanding and communicating.	Medium
User who may encounter various accents and speaking styles	The system should adapt to different accents and speech patterns based on user feedback.	USN - 4	As a user who may encounter various accents and speaking styles, I want the system to continuously improve its accuracy and adapt to different speech patterns based on user feedback.	The system should include a feedback mechanism for users to report inaccuracies or provide input. User feedback should be used to train the deep learning models and improve the system's accuracy. The system's transcriptions should become more accurate and adaptable over time.	Medium

5.2 Solution Architecture





6. PROJECT PLANNING & SCHEDULING

6.1 Technical Architecture

S.No	Components	Description	Technology
1.	Data Collection	Gather a dataset of video clips featuring people speaking and simultaneously showing their lip movements.	Digital cameras, webcams, and video recording software for capturing training data.
2.	Data Preprocessing	Cleaning, normalizing, and transforming collected data into a suitable format for modeling.	Python (Pandas), data cleaning techniques
3.	Feature Engineering	Feature engineering for lip reading involves extracting and preprocessing visual lip movement cues from video frames .	Feature selection techniques, domain knowledge
4.	Exploratory Data Analysis (EDA)	Visualizing and analyzing data to uncover patterns and relationships among variables.	Data visualization tools (Matplotlib, Seaborn)
5.	Model Selection	Choose the appropriate deep learning architecture for lip reading accuracy and performance optimization.	Scikit-Learn, TensorFlow or PyTorch
6.	Model Architecture:	Design the neural network architecture to map video frames to corresponding text transcriptions. Implement model to capture temporal dependencies in the lip movements and predict the accurate output text and to to extract features from the video frames.	Recurrent Neural Network (RNN) , Long Short-Term Memory (LSTM) Convolutional Neural Networks (CNNs)
7.	Training and Testing	Splitting the dataset into training and testing sets to train and evaluate model performance.	Cross-validation, model evaluation metrics
8.	Hyperparameter	Optimizing model hyperparameters to improve predictive	Grid search, random search, hyperparameter

	Tuning	accuracy.	optimization tools
9.	Model Evaluation	Assessing model performance using metrics like Mean Absolute Error (MAE) and R-squared.	Scikit-Learn, custom evaluation scripts
10.	Deployment	Implementing the model for real-time predictions, potentially through APIs or within hotel management systems.	Flask, python, HTML, CSS, JS, Bootstrap
11.	Monitoring and Maintenance	Continuously monitoring the model's performance and making updates to maintain accuracy.	Logging, alerting systems, automated pipelines via AWS CloudWatch
12.	Visualization and Reporting	Creating dashboards and reports for interpreting model results and environmental impact.	Data visualization tools (AWS QuickSight, Power BI)

7. CODING & SOLUTIONING

Loading the data.

Download the dataset here: https://drive.google.com/file/d/1VNY2vZ9gtqfEb0UX80jSKto3dVRfSL0v/view?usp=sharing

Data Preparation

```
import os
import cv2
import tensorflow as tf
import numpy as np
from typing import List
from matplotlib import pyplot as plt
import imageio
```

We require 5 functions to process the data

- load_video()
- char_to_num
- num_to_char
- load_alignments()
- load_data()

```python
def load_video(path:str) -> List[float]:

    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[190:236,80:220,:])
    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
```

```python
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)
```

```python
def load_alignments(path:str) -> List[str]:
    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:]
```

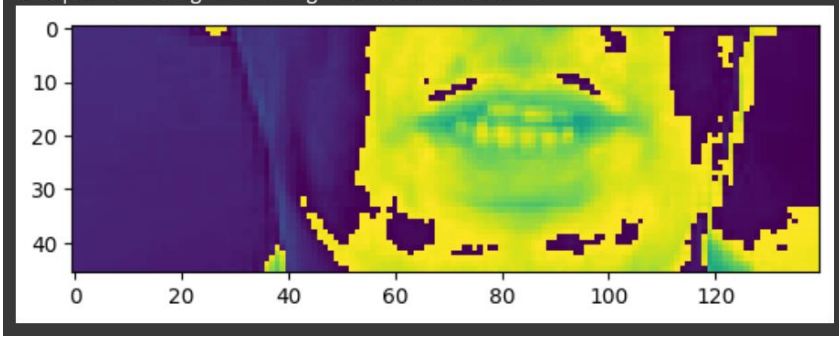
```python
def load_data(path: str):
    path = bytes.decode(path.numpy())
    file_name = path.split('/')[-1].split('.')[0]
    # File name splitting for windows
    # file_name = path.split('\\')[-1].split('.')[0]
    video_path = os.path.join('data','s1',f'{file_name}.mpg')
    alignment_path = os.path.join('data','alignments','s1',f'{file_name}.align')
    frames = load_video(video_path)
    alignments = load_alignments(alignment_path)

    return frames, alignments
```

Visual Analysis

```python
plt.imshow(frames[40])
```

```
<matplotlib.image.AxesImage at 0x7a9f0f005de0>
```



Splitting data into train and test and validation sets

```
data = tf.data.Dataset.list_files('./data/s1/*.mpg')
data = data.shuffle(500, 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)
# Added for split
train = data.take(450)
test = data.skip(450)
```

Importing necessary libraries

```
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.optimizers import Adam
from tensorflow.keras.callbacks import ModelCheckpoint, LearningRateScheduler
```

Defining callbacks, Loss function and building the model

```
def scheduler(epoch, lr):
    if epoch < 30:
        return lr
    else:
        return lr * tf.math.exp(-0.1)
```

```
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
```

```
class ProduceExample(tf.keras.callbacks.Callback):
    def __init__(self, dataset) -> None:
        self.dataset = dataset.as_numpy_iterator()

    def on_epoch_end(self, epoch, logs=None) -> None:
        data = self.dataset.next()
        yhat = self.model.predict(data[0])
        decoded = tf.keras.backend.ctc_decode(yhat, [75,75], greedy=False)[0][0].numpy()
        for x in range(len(yhat)):
            print('Original:', tf.strings.reduce_join(num_to_char(data[1][x])).numpy().decode('utf-8'))
            print('Prediction:', tf.strings.reduce_join(num_to_char(decoded[x])).numpy().decode('utf-8'))
            print('~'*100)
```

```
model = Sequential()
model.add(Conv3D(128, 3, input_shape=(75,46,140,1), padding='same'))
model.add(Activation('relu'))
model.add(MaxPool3D((1,2,2)))

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

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

model.add(TimeDistributed(Flatten()))

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

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

model.add(Dense(char_to_num.vocabulary_size()+1, kernel_initializer='he_normal', activation='softmax'))
```

Train the models

```
checkpoint_callback = ModelCheckpoint(os.path.join('models','checkpoint'), monitor='loss', save_weights_only=True)
```

```
schedule_callback = LearningRateScheduler(scheduler)
```

```
example_callback = ProduceExample(test)
```

```
model.fit(train, validation_data=test, epochs=100, callbacks=[checkpoint_callback, schedule_callback, example_callback])
```

Make a Prediction

```
import gdown
url = 'https://drive.google.com/uc?id=1vWscXs4Vt0a_1IH1-ct2TCgXAZT-N3_Y'
output = 'checkpoints.zip'
gdown.download(url, output, quiet=False)
gdown.extractall('checkpoints.zip', 'models')
```

```
model.load_weights('models/checkpoint')
```

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

```
sample = data.next()
```

```
yhat = model.predict(sample[0])
```

```
1/1 [==============================] - 14s 14s/step
```

```
print('~'*100, 'REAL TEXT')
[tf.strings.reduce_join([num_to_char(word) for word in sentence]) for sentence in sample[1]]
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ REAL TEXT
[<tf.Tensor: shape=(), dtype=string, numpy=b'set blue in t three soon'>,
 <tf.Tensor: shape=(), dtype=string, numpy=b'set green by p four now'>]
```

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

```
print('~'*100, 'PREDICTIONS')
[tf.strings.reduce_join([num_to_char(word) for word in sentence]) for sentence in decoded]
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ PREDICTIONS
[<tf.Tensor: shape=(), dtype=string, numpy=b'set blue in t three soon'>,
 <tf.Tensor: shape=(), dtype=string, numpy=b'set green by four now'>]
```

Integrate with Web Framework

We will use streamlit to deploy our product.

Streamlit is a free and open-source framework to rapidly build and share beautiful machine learning and data science web apps.

Create an app folder and create following .py files in it:

- modelutil.py
- streamlitapp.py
- utils.py

Write the code for 'utils.py'

```python
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
)

def load_video(path:str) -> List[float]:
    #print(path)
    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[190:236,80:220,:])
    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
```

```python
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:]

def load_data(path: str):
    path = bytes.decode(path.numpy())
    # file_name = path.split('/')[-1].split('.')[0]
    # File name splitting for windows
    file_name = path.split('\\')[-1].split('.')[0]
    video_path = os.path.join('..','data','s1',f'{file_name}.mpg')
    alignment_path = os.path.join('..','data','alignments','s1',f'{file_name}.align')
    frames = load_video(video_path)
    alignments = load_alignments(alignment_path)

    return frames, alignments
```

Write code for ‘modelutil.py’

```python
import os
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv3D, LSTM, Dense, Dropout, Bidirectional, MaxPool3D, Activation, TimeDistributed, Flatten

def load_model() -> Sequential:
    model = Sequential()

    model.add(Conv3D(128, 3, input_shape=(75,46,140,1), padding='same'))
    model.add(Activation('relu'))
    model.add(MaxPool3D((1,2,2)))

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

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

    model.add(TimeDistributed(Flatten()))

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

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

    model.add(Dense(41, kernel_initializer='he_normal', activation='softmax'))

    model.load_weights(os.path.join('..','models','checkpoint'))

    return model
```

Write code for ‘streamlitapp.py’

```python
# Import all of the dependencies
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 to the streamlit app as wide
st.set_page_config(layout='wide')

# Setup the sidebar
with st.sidebar:
    st.image('https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcRe7q7g-97c4CPUhH4U8RHvX7BAylxkEif4hg&usqp=CAU')
    st.title('Lip Reading using Deep Learning')

st.title('Lip Reading App')
# Generating a list of options or videos
options = os.listdir(os.path.join('data', 's1'))
selected_video = st.selectbox('Choose video', options)
```

```python
# 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','s1', selected_video)
        os.system(f'ffmpeg -i {file_path} -vcodec libx264 test_video.mp4 -y')

        # Rendering inside of the app
        video = open('test_video.mp4', 'rb')
        video_bytes = video.read()
        st.video(video_bytes)


    with col2:
        video, annotations = load_data(tf.convert_to_tensor(file_path))

        st.info('This is the output of the machine learning model as tokens')
        model = load_model()
        yhat = model.predict(tf.expand_dims(video, axis=0))
        decoder = tf.keras.backend.ctc_decode(yhat, [75], greedy=True)[0][0].numpy()
        st.text(decoder)

        # Convert prediction to text
        st.info('Words spoken in the video are:')
        converted_prediction = tf.strings.reduce_join(num_to_char(decoder)).numpy().decode('utf-8')
        st.markdown(f'## {converted_prediction}')
```

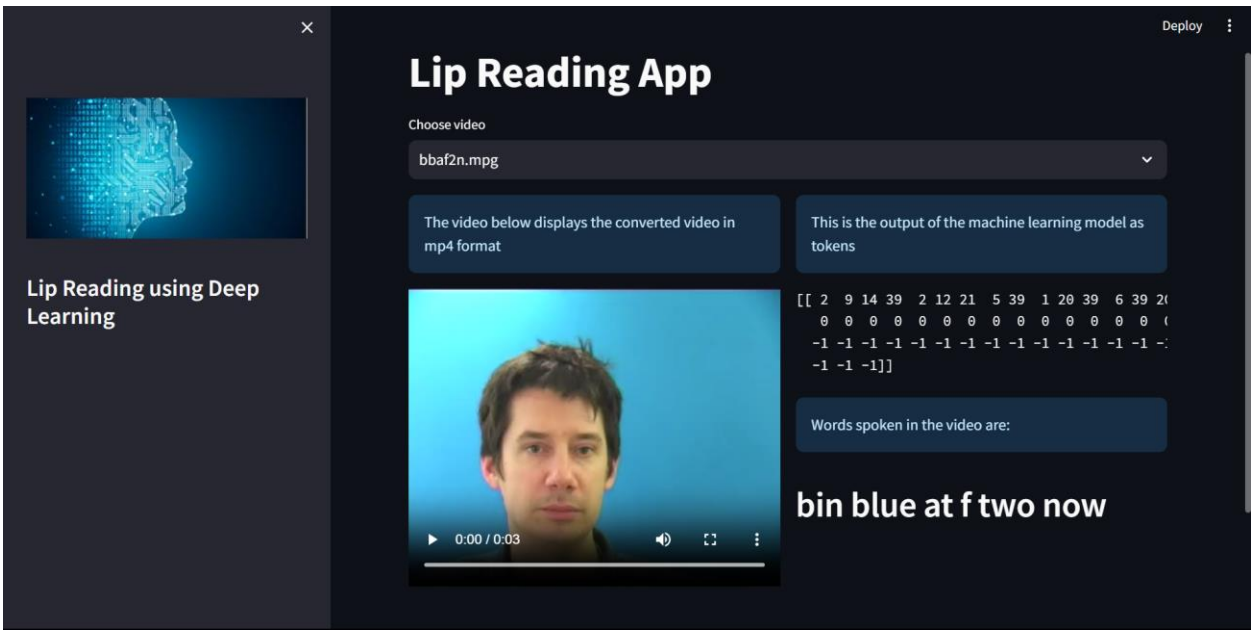
To run the website, go to the terminal and run the command:

- `streamlit run streamlitapp.py`

Make sure you have streamlit installed in your environment. If not install it using the command:

- `pip install streamlit`

This is what the website looks like:



8. PERFORMANCE TESTING

8.1 Performance Metrics

Training Accuracy :- 95.47%

Validation Accuracy :- 90.61%

9. RESULTS

9.1 Output Screenshots

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

```
sample = data.next()
```

```
yhat = model.predict(sample[0])
```

```
1/1 [==============================] - 14s 14s/step
```

```
print('~'*100, 'REAL TEXT')
[tf.strings.reduce_join([num_to_char(word) for word in sentence]) for sentence in sample[1]]
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ REAL TEXT
[<tf.Tensor: shape=(), dtype=string, numpy=b'set blue in t three soon'>,
 <tf.Tensor: shape=(), dtype=string, numpy=b'set green by p four now'>]
```

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

```
print('~'*100, 'PREDICTIONS')
[tf.strings.reduce_join([num_to_char(word) for word in sentence]) for sentence in decoded]
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ PREDICTIONS
[<tf.Tensor: shape=(), dtype=string, numpy=b'set blue in t three soon'>,
 <tf.Tensor: shape=(), dtype=string, numpy=b'set green by four now'>]
```

10. CONCLUSION

In conclusion, this project aims to develop an end-to-end machine learning solution for word detection in spoken language videos. Leveraging advanced deep learning algorithms like LSTM and Neural Networks, the project seeks to achieve accurate predictions. The focus on real-time processing and scalability reflects the project's commitment to practical applicability. As the system undergoes development, considerations for performance, reliability, security, and usability will be integral. The anticipated outcomes include a robust word detection system capable of handling diverse linguistic nuances and user scenarios. This project aspires to contribute to advancements in accurate transcription services and language processing, addressing a crucial need in the domain of spoken language recognition.

11. FUTURE SCOPE

- Webcam based feature can be added where the video can be transcribed live on the webcam feed.
- The project can be improved by using a more diversified dataset with data collected from different people.
- It can also be adapted for different languages.