

# Project Report Format

## Lip Reading Using Deep Learning

### 1. INTRODUCTION

Lip-reading Using Deep Learning is an innovative and accessible mobile application designed to assist individuals with hearing impairments by leveraging advanced deep learning techniques to interpret and transcribe spoken language through lip movements.

#### 1.1 Project Overview

The primary goal of this project is to develop a user-friendly mobile application that empowers individuals who are deaf or hard of hearing to better comprehend spoken communication by analysing and interpreting lip movements. The application utilizes state-of-the-art deep learning models to convert these visual cues into accurate and real-time transcriptions.

**Objective:** To develop an end-to-end Machine Learning solution for detecting speech from the video of a person speaking. The solution includes the use of Deep Learning technique such as LSTM and Neural Network to achieve the same and predict the accurate output.

The idea of lip reading can be used in a variety of situations. Some of them are: -

1. **Improved Speech Recognition:** Lip reading can complement audio-based speech recognition systems, especially in noisy environments or scenarios where the audio signal is unclear. Integrating lip reading with traditional speech recognition can enhance accuracy and robustness.
2. **No Need for Audio Data:** Traditional speech recognition models require large amounts of transcribed audio data for training. In contrast, an end-to-end lip-reading system can be trained solely on video data, eliminating the need for transcribed audio, which can be expensive and time-consuming to obtain.
3. **Multi-Modal Applications:** End-to-end lip reading can be combined with audio-based systems to create multi-modal applications. For example, in video conferencing, it can help improve real-time communication by providing more accurate transcriptions.

**Accessibility for Hearing-Impaired Individuals:** Lip reading can be an essential communication tool for individuals with hearing impairments. An accurate lip-reading system can enhance their ability to understand spoken language and participate in conversations.

#### 1.2 Purpose

##### **Enhancing Accessibility:**

The primary purpose of the project is to enhance accessibility for individuals with hearing impairments. By leveraging deep learning technology, the application facilitates a more inclusive communication experience, allowing users to understand spoken language through visual cues.

##### **Empowering the Deaf and Hard of Hearing:**

The project aims to empower individuals who are deaf or hard of hearing by providing them with a tool that improves their ability to participate in conversations, reducing communication barriers and fostering independence.

**Filling a Critical Communication Gap:**

The project addresses a critical gap in communication for the hearing-impaired community. Lip reading is an important skill, and the application seeks to make this skill more accessible and accurate through the integration of deep learning algorithms.

**Promoting Independence:**

By offering a real-time and reliable lip-reading solution, the project encourages independence among users. They can actively engage in conversations without relying solely on sign language or other forms of communication support.

**Advancing Assistive Technology:**

The project contributes to the advancement of assistive technology by harnessing the capabilities of deep learning. It showcases how cutting-edge technologies can be applied to address specific needs within the accessibility and disability community.

**Global Language Support:**

The purpose includes breaking language barriers for the hearing-impaired. With multi-lingual support, the application aims to cater to diverse linguistic communities, making lip reading accessible across different languages.

**Safety and Emergency Assistance:**

The inclusion of an emergency services integration feature serves the purpose of ensuring user safety. Users can quickly and easily access emergency assistance when needed, adding an extra layer of security to their daily lives.

## **2. LITERATURE SURVEY**

**2.1 Existing problem:**

The existing problem in the field of lip reading and speech recognition lies in the limitations of audio-based systems, especially in challenging environments where the audio signal may be unclear or contaminated by noise. Traditional speech recognition models heavily rely on transcribed audio data for training, which can be both expensive and time-consuming to obtain. Moreover, these models may struggle in scenarios with multiple speakers, accents, or non-standard pronunciation.

To address these challenges, there is a need for a more robust and accurate solution that can complement audio-based systems. Lip reading using machine learning presents itself as a promising approach. By leveraging deep learning algorithms like LSTM and Neural Networks, it becomes possible to create an end-to-end system capable of accurately detecting words from a video of a person speaking.

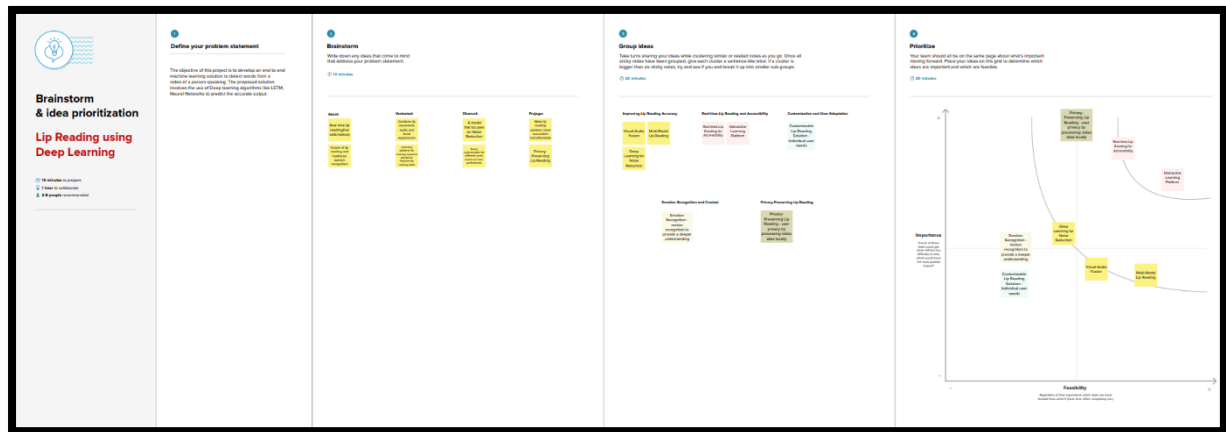
**2.2 References:**

The development of lip reading using deep learning is built upon existing research and methodologies. Key references in the literature include studies on deep neural networks, recurrent neural networks (RNNs), and Long Short-Term Memory (LSTM) networks. Researchers have explored the fusion of visual and audio information for improved speech recognition, and the integration of lip reading with traditional audio-based systems.

Some seminal works in this area may include research papers on deep learning applications in computer vision, speech recognition, and multi-modal learning. Additionally,



## 3.2 Ideation & Brainstorming



## 4. REQUIREMENT ANALYSIS

### 4.1 Functional requirement:

Functional requirements define the specific capabilities and functionalities that the lip-reading system must possess to meet its objectives. In the context of this project, the functional requirements include:

**Video Input Processing:** The system should be able to take video input containing a person speaking and extract relevant visual features for lip reading.

**Deep Learning Model:** Implement deep learning algorithms, such as LSTM and Neural Networks, to predict accurate word outputs based on the visual features extracted from the video.

**Integration with Audio-Based Systems:** Ensure seamless integration with traditional audio-based speech recognition systems to enhance accuracy and robustness in varied environments.

**Multi-Modal Integration:** Allow for the combination of lip reading with audio-based systems, providing a multi-modal approach that can be employed in real-time communication scenarios.

**Training Module:** Develop a training module that can efficiently train the deep learning model on a dataset of video samples, enabling the system to learn and improve its accuracy over time.

**User Interface (UI):** Provide a user-friendly interface for users to interact with the system, input video data, and receive transcriptions or outputs.

**Real-Time Processing:** If applicable, design the system to process video data in real-time, ensuring timely and efficient word predictions.

**Accessibility Features:** Implement features that cater to the needs of hearing-impaired individuals, ensuring that the system serves as an effective communication tool for this user group.

## 4.2 Non-Functional requirements

Non-functional requirements specify the characteristics and qualities that the system must exhibit but are not directly related to specific functionalities. In the context of this lip-reading system, the non-functional requirements include:

**Accuracy:** The system should achieve a high level of accuracy in predicting words from lip movements in videos, ensuring reliable performance in various scenarios.

**Scalability:** Design the system to handle an increasing amount of data, users, or processing demands without compromising performance.

**Robustness:** Ensure that the system can perform effectively in noisy environments and handle variations in lip movements, accents, and pronunciations.

**Security:** Implement security measures to protect the privacy and integrity of the data, especially if the system is handling sensitive information.

**Usability:** The user interface should be intuitive and user-friendly, catering to users with varying levels of technical expertise.

**Training Efficiency:** The training module should be efficient in terms of time and computational resources, enabling quick iterations during the model development phase.

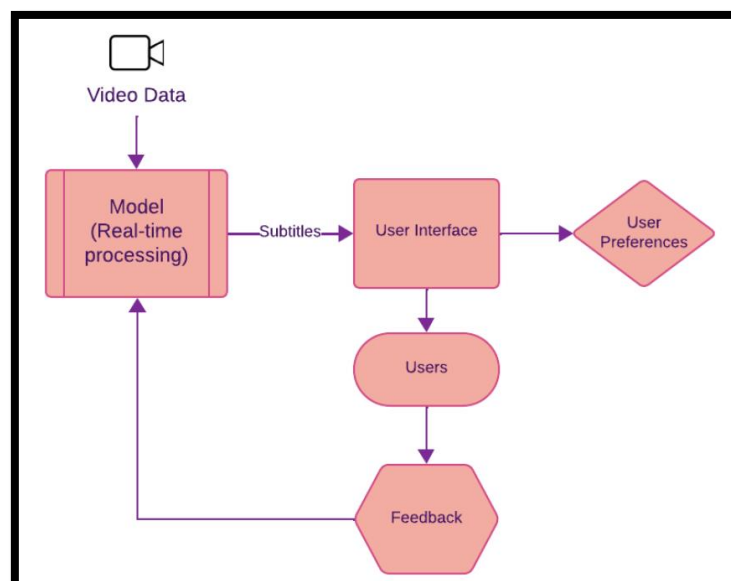
**Real-Time Performance:** If applicable, the system should exhibit real-time processing capabilities, providing timely predictions for practical applications.

**Compatibility:** Ensure compatibility with different video formats, platforms, and devices to enhance the system's versatility and usability.

**Maintainability:** Design the system with a clear and modular structure, making it easy to maintain and update as needed.

## 5. PROJECT DESIGN

### 5.1 Data Flow Diagrams & User Stories



User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Customer (Mobile user)	Video Input and Processing	USN-1	As a mobile user, I want to be able to use a lip reading app to understand what people are saying in noisy environments or when the audio signal is unclear.	The app should be able to accurately transcribe words from videos of people speaking in noisy environments or when the audio signal is unclear/The app should be easy to use and accessible to a wide range of users.	High	Sprint-1
hearing-impaired individual	Registration Confirmation	USN-1	As a user, I will receive confirmation email once I have registered for the applicationAs a hearing-impaired individual, I want to be able to use a lip reading app to improve my ability to understand spoken language and participate in conversations.	The app should be able to accurately transcribe words from videos of people speaking at a variety of speeds and with different accents/The app should be able to transcribe words in real time, so that the user can follow conversations as they happen.	High	Sprint-1
video conferencing participant	Real-time Integration	USN-1	As a video conferencing participant, I want to be able to use a lip reading app to improve the accuracy of transcriptions	The app should be able to accurately transcribe words from videos of	High	Sprint-2

## 5.2 Solution Architecture

### User Interface (UI) Layer:

- Component: Flutter Mobile App.
- Interaction: Users interact with the mobile app to upload videos and view real-time subtitles.

### Application Layer:

- Components: Flutter Mobile App, Real-Time Transcription Engine (Python with TensorFlow), REST API (Google Cloud Platform).
- Interaction: The Flutter mobile app communicates with both the local Real-Time Transcription Engine (built using TensorFlow) and the remote REST API hosted on Google Cloud Platform for real-time word detection and transcription.

### Cloud Infrastructure:

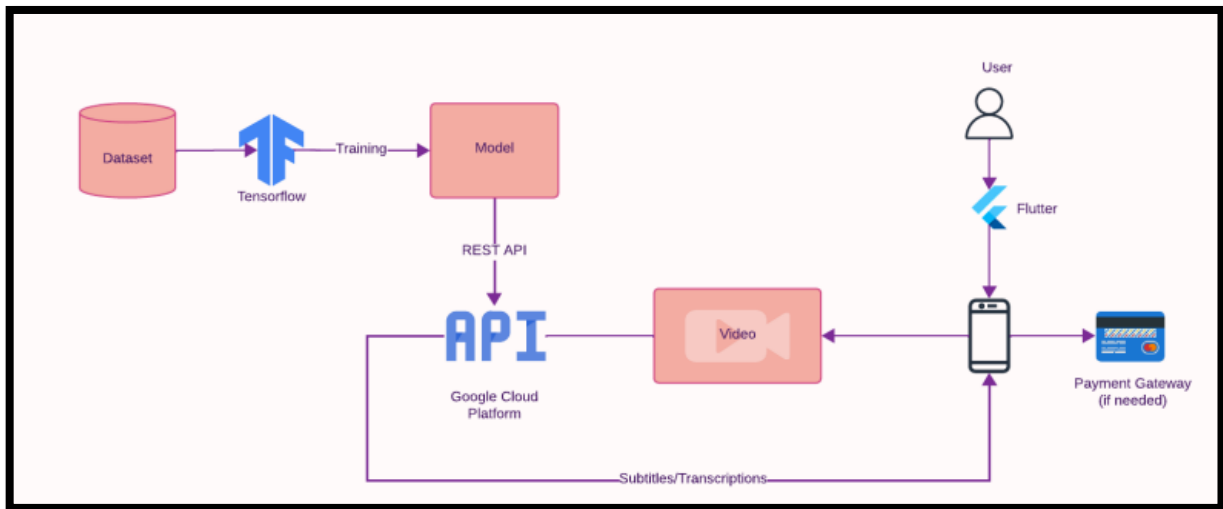
- Component: Google Cloud Platform (GCP - for REST API hosting).
- Interaction: The REST API is hosted on GCP for model access.

### Monetization and Business Logic:

- Component: Payment Gateway (Razorpay).
- Interaction: Payments are processed via the Razorpay payment gateway

## 6. PROJECT PLANNING & SCHEDULING

### 6.1 Technical Architecture



### 6.2 Sprint Planning & Estimation

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-1	Project Setup & Infrastructure	USN-1	Set up the development environment with the required tools and frameworks to start the Lip Reading Project	1	High	Akash P
Sprint-2	Data Collection	USN-2	Gather a diverse dataset of images and videos along with their movement data for training the LSTM model.	2	High	Whole Team
Sprint-2	Data Preprocessing	USN-3	Preprocess the collected dataset by normalizing pixel values, and splitting it into training and validation sets.	2	High	Dhanush, Prajagar

Sprint-3	Model Development	USN-4	Explore and evaluate LSTM models with different parameters to select the most suitable model for Lip Reading using Deep Learning.	5	High	Venkatesh, Akash
Sprint-3	Training	USN-5	Train the selected LSTM model using the preprocessed dataset and monitor its performance on the validation set.	5	High	Dhanush, Prajagar
Sprint-4	App Design	USN-6	Design a simple, intuitive, easy to navigate with clear icons and large text mobile application	2	Medium	Whole Team
Sprint-4	App Development	USN-7	Develop the mobile application following the design and test its functioning.	4	Medium	Akash, Venkatesh
Sprint-5	Model Deployment	USN-8	Deploy the trained LSTM model as an API on Google Cloud Platform.	3	Medium	Prajagar, Akash
Sprint-5	Model Integration	USN-9	Integrate the model's API into a user-friendly application for users to upload their images or videos and receive the results.	2	Medium	Venkatesh, Dhanush
Sprint-6	Testing and Quality Assurance	USN-10	Test the functioning capabilities of the application developed and make sure everything works as planned and improve upon feedback.	1	Low	Whole Team

## 6.3 Sprint Delivery Schedule

Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date (Planned)	Story Points Completed (as on Planned End Date)	Sprint Release Date (Actual)
Sprint-1	1	1	24 Oct 2023	24 Oct 2023	1	24 Oct 2023
Sprint-2	4	1	25 Oct 2023	25 Oct 2023	4	25 Oct 2023
Sprint-3	10	5	26 Oct 2023	30 Oct 2023		
Sprint-4	6	4	31 Oct 2023	3 Nov 2023		
Sprint-5	5	1	4 Nov 2023	4 Nov 2023		
Sprint-6	1	2	5 Nov 2023	6 Nov 2023		

## 7. CODING & SOLUTIONING

### 7.1 Feature 1 (ML model)

#### Prerequisites:

To complete this project, you must require the following software's, concepts, and packages

*Anaconda Navigator - A tool for installing and using Python Environments*

To install Anaconda navigator and to know how to use Jupyter Notebook & Spyder using Anaconda watch the video

**Link:** [Click here](#) to watch the tutorial

**VSCode:** It is a text editor which is powerful enough to perform as an IDE for most of the languages.

To install VScode and setup python and Jupyter Notebook, see the below video

**Link:** [Click Here](#) to watch the tutorial

To build this Machine learning model you require the following packages

- **Numpy:** It is an open-source numerical Python library. It contains a multidimensional array and matrix data structures and can be used to perform mathematical operations
- **Scikit-learn:** It is a free machine learning library for Python. It features various algorithms like support vector machine, random forests, and k-neighbors, and it also supports Python numerical and scientific libraries like NumPy and SciPy
- **Flask:** Web framework used for building Web applications
- **OpenCV:** OpenCV is a library of programming functions mainly for real-time computer vision.

#### Python packages:

- Open anaconda prompt as administrator
- Type “pip install numpy pandas scikit-learn tensorflow keras Flask imageio opencv-python” and click enter.
- You would have all the packages used in this project installed



## **Deep Learning Concepts**

**CNN:** a Convolutional neural network is a class of deep neural networks, most commonly applied to analysing visual imagery.

**LSTM:** LSTM (Long Short-Term Memory) is a recurrent neural network (RNN) architecture widely used in Deep Learning. It excels at capturing long-term dependencies, making it ideal for sequence prediction tasks.

## **App Development Concepts**

**Flutter and Dart:** Flutter is an open-source UI software development kit created by Google. It is used to develop cross platform applications from a single codebase for any web browser, Fuchsia, Android, iOS, Linux, macOS, and Windows. Dart is the programming language used to code Flutter apps.

## **Project Objectives:**

By the end of this project, you will:

- Know fundamental concepts and techniques of Deep Learning and LSTM.
- Gain a broad understanding of video processing and analysis.
- Know how to pre-process/clean the data using different data preprocessing techniques.
- know how to build a FAST API end point using the Flask framework.
- know how to deploy the API in Google Cloud and Connect it to Flutter App

## **Project Flow**

The user interacts with the Android App to select the file (from camera or a saved file).

Selected input is sent to the model on the Google Cloud using the API.

The sent file is processed by the model that we have built and deployed

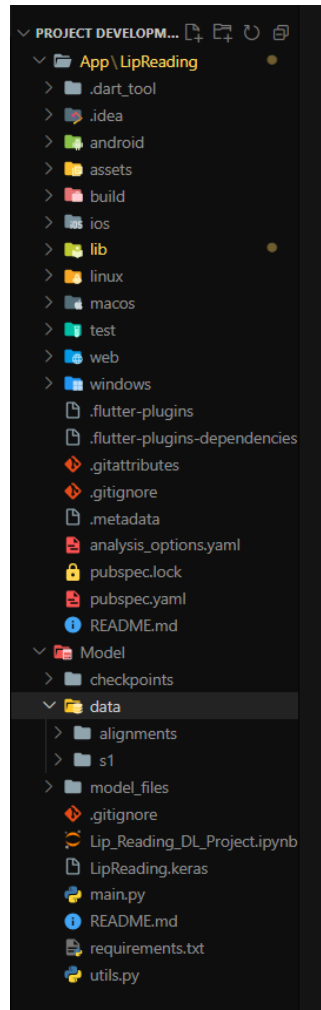
Once the model analyses the input, the prediction is sent back to the Android App, which is then displayed to the user.

To accomplish this, we have to complete all the activities listed below,

- Define Problem / Problem Understanding
  - Specify the business problem
  - Business requirements
  - Literature Survey.
  - Social or Business Impact.
- Data Collection & Preparation
  - Collect the dataset
  - Data Preparation
- Exploratory Data Analysis
  - Descriptive statistical
  - Visual Analysis
- Model Building
  - Building a model.
  - Training the model.

- Testing the model
- Model Deployment
  - Save the best model
- App Development
  - Create an appealing Flutter Application and integrate model with App
- Project Demonstration & Documentation
  - Record explanation Video for project end to end solution
  - Project Documentation-Step by step project development procedure

## Project Structure



- 
- The dataset contains folders **s1** which contains the videos for training and a folder **alignments** which has the words the person speaks in the video
  - We have a Flutter Application folder for the development of the Android App
  - We have our saved model as **LipReading.keras** and we have our FAST API Endpoint in the **main.py** file
  - The **model\_files** and **checkpoints** folders contain the model data such as weights etc.

## **Milestone 1: Define Problem / Problem Understanding**

### **Activity 1: Specify the business problem**

Refer Project Description

### **Activity 2: Business requirements**

Here are some potential business requirements for lip reading using deep learning:

1. **Accurate prediction:** The predictor must be able to accurately predict the words. The accuracy of the prediction is crucial for the client. The client could be a corporate or a business person or a deaf person or others with the need for it.
2. **User-friendly interface:** The predictor must have a user-friendly interface that is easy to navigate and understand. The interface should present the results of the predictor in a clear and concise manner.
3. **Scalability:** The predictor must be able to scale up based on the prediction from our product. The model should be able to handle any size of data without compromising on its accuracy or efficiency.

### **Activity 3: Literature Survey**

The literature review on lip reading utilizing deep learning reveals a burgeoning field at the intersection of computer vision and natural language processing. Researchers have explored various deep neural network architectures, such as Convolutional neural networks (CNNs) and recurrent neural networks (RNNs), to decipher the intricate visual cues present in lip movements. Studies highlight the challenges associated with diverse lighting conditions, speaker variations, and the need for robust feature extraction techniques. Additionally, recent advancements in attention mechanisms and transformer-based models have shown promise in enhancing lip reading accuracy. This review synthesizes key findings, methodological approaches, and benchmark datasets, shedding light on the evolving landscape of deep learning applications in lip reading and paving the way for future research directions in this compelling area.

### **Activity 4: Social or Business Impact**

1. **Privacy and Security:** Audio-based speech recognition systems may raise privacy concerns, as they capture and process audio data, potentially infringing on individuals' privacy. Lip reading systems, on the other hand, rely on visual information and might be considered less intrusive in this regard.
2. **Use in Noisy Environments:** In environments with high background noise, audio-based speech recognition can be challenging. Lip reading can help provide context and improve accuracy in these noisy scenarios.
3. **Cross-Lingual Applications:** Lip reading is language-agnostic, which means the same model can potentially be applied to lip reading in different languages without requiring language-specific training data.
4. **Hearing Impaired Individuals:** Lip reading can help those with hearing issues to understand a video and its contents without having to resort to more expensive alternatives. This is a quick fix solution for the same.

## **Milestone 2: Data Collection & Preparation**

DL depends heavily on data. It is the most crucial aspect that makes algorithm training possible. So, this section deals with the preprocessing steps on the data and make it such that the model can use it to train.

### **Activity 1: Loading the data**

Link for the required dataset: <https://www.kaggle.com/datasets/rishisrdy/lipreading>

Download the provided data and load the data into the data folder

## Activity 2: Data Preparation


As we have understood how the data is, let's pre-process the collected data.

We need the following libraries so we need to import them before starting to code.

- OpenCV (cv2)
- tensorflow
- numpy
- matplotlib

We have 5 functions, to load the video, to load the alignments, to load both video and alignments correspondingly, to convert the input to numbers and to convert it back to characters in the end. They are displayed as follows.

► **load\_video:** The function basically uses OpenCV to load the video into frames and then crop the frame to the part of the lips and then convert all frames to grayscale for easy processing and then returns the normalized frames which is done using Z-Score Normalisation



```
1 def load_video(path:str) -> List[float]:
2
3     cap = cv2.VideoCapture(path)
4     frames = []
5     for _ in range(int(cap.get(cv2.CAP_PROP_FRAME_COUNT))):
6         print(int(cap.get(cv2.CAP_PROP_FRAME_COUNT)))
7         ret, frame = cap.read()
8         frame = tf.image.rgb_to_grayscale(frame)
9         frames.append(frame[190:236,80:220,:])
10    cap.release()
11
12    mean = tf.math.reduce_mean(frames)
13    std = tf.math.reduce_std(tf.cast(frames, tf.float32))
14    return tf.cast((frames - mean), tf.float32) / std
```

► **char\_to\_num:** This function sets up a vocabulary and a StringLookup layer using TensorFlow (tf.keras.layers) to map characters to numeric values.

► **num\_to\_char:** This function sets up a vocabulary and a StringLookup layer using TensorFlow (tf.keras.layers) to map numeric values back to their corresponding characters.

```

1 vocab = [x for x in "abcdefghijklmnopqrstuvwxyz?!123456789 "]
2 char_to_num = tf.keras.layers.StringLookup(vocabulary=vocab, oov_token="")
3 num_to_char = tf.keras.layers.StringLookup(
4     vocabulary=char_to_num.get_vocabulary(), oov_token="", invert=True
5 )

```

► **load\_alignments:** This function reads the align file which contains the words spoken and then converts it into the numeric tensor using the char\_to\_num and returns the list.

```

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

```

► **load\_data:** This function takes the path of the file as input and then loads the video using the load\_video and the corresponding alignment using the load\_alignment function and then returns the frames and alignments

```

1 def load_data(path: str):
2     path = bytes.decode(path.numpy())
3     file_name = path.split('\\')[-1].split('.')[0]
4     video_path = os.path.join('data', 's1', f'{file_name}.mpg')
5     alignment_path = os.path.join('data', 'alignments', 's1', f'{file_name}.align')
6     frames = load_video(video_path)
7     alignments = load_alignments(alignment_path)
8
9     return frames, alignments

```

The load\_data takes input in the form of a tensor string. We can use the tf.convert\_to\_tensor(path\_str) to give the input to the function.

### Activity 3: Data Preprocessing

We need to preprocess the image frames in order to make them more efficient while building the model. So, we use Smoothing to better make the frames clear.

```
1  SHOW_DEBUG_STEPS = True
2
3  # Reading video
4  cap = cv2.VideoCapture('/content/drive/MyDrive/data/s1/bbaf3s.mpg')
5
6  # if video is not present, show error
7  if not(cap.isOpened()):
8      print("Error reading file")
9
10 # Check if you are able to capture the video
11 ret, fFrame = cap.read()
12
13 # Capturing 2 consecutive frames and making a copy of those frame. Perform all operations on the copy frame.
14 ret, fFrame1 = cap.read()
15 ret, fFrame2 = cap.read()
16 ret, fFrame3 = cap.read()
17 img1 = fFrame1.copy()
18 img2 = fFrame2.copy()
19 img3 = fFrame3.copy()
20
21 if(SHOW_DEBUG_STEPS):
22     print ('img1 height = ' + str(img1.shape[0]))
23     print ('img1 width = ' + str(img1.shape[1]))
24     print ('img2 height = ' + str(img2.shape[0]))
25     print ('img2 width = ' + str(img2.shape[1]))
26     print ('img3 height = ' + str(img3.shape[0]))
27     print ('img3 width = ' + str(img3.shape[1]))
28
29 # Convert the colour images to greyscale in order to enable fast processing
30 img1 = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
31 img2 = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)
32 img3 = cv2.cvtColor(img3, cv2.COLOR_BGR2GRAY)
33
34 #plotting
35 plot_image([img1, img2, img3], cmap='gray', captions=["First frame", "Second frame", "Third frame"])
36
37 # Add some Gaussian Blur
38 img1 = cv2.GaussianBlur(img1,(5,5),0)
39 img2 = cv2.GaussianBlur(img2,(5,5),0)
40 img3 = cv2.GaussianBlur(img3,(5,5),0)
41 #plotting
42 plot_image([img1, img2, img3], cmap='gray', captions=["GaussianBlur first frame", "GaussianBlur second frame", "GaussianBlur third frame"])
```

### Milestone 3: Exploratory Data Analysis

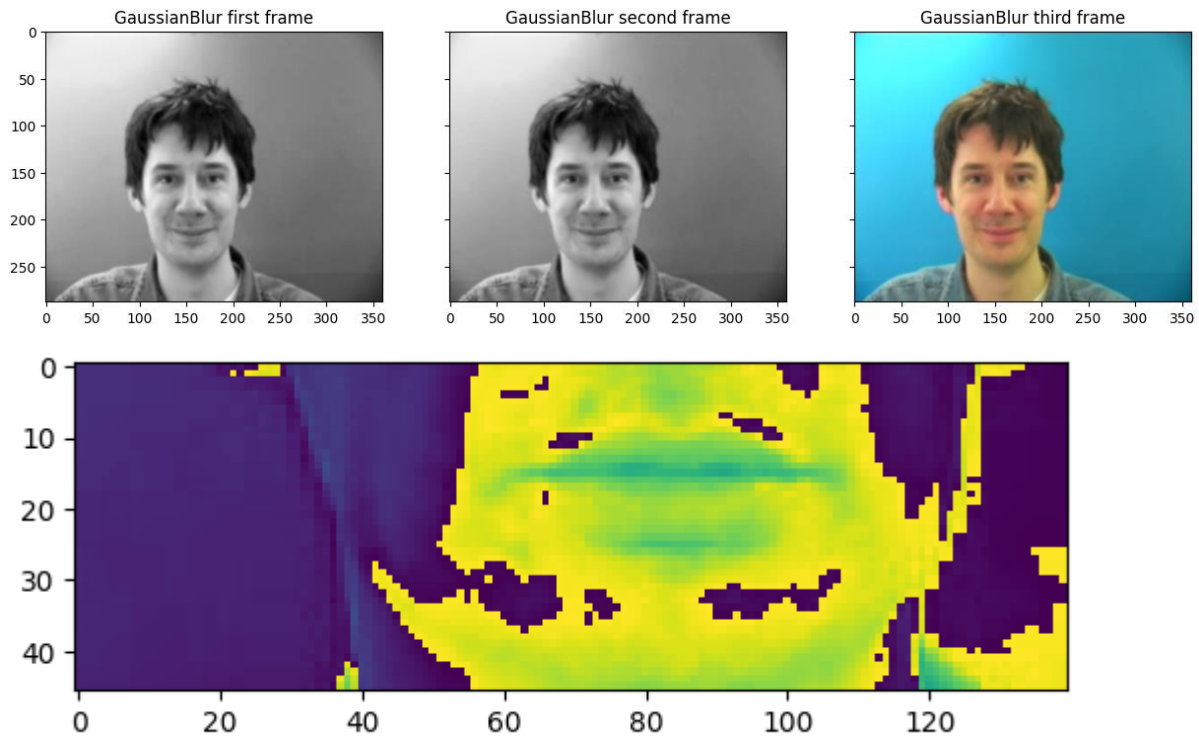
#### Activity 1: Descriptive statistical

Descriptive analysis is to study the basic features of data with the statistical process. With this describe function we can understand the unique, top and frequent values of categorical features. And we can find mean, std, min, max and percentile values of continuous features. which are not suitable for our dataset.

As our data consists of videos and alignments, there is no unnecessary data which can be eliminated.

#### Activity 2: Visual analysis

Visual analysis is the process of using visual representations, such as charts, plots, and graphs, to explore and understand data. It is a way to quickly identify patterns, trends, and outliers in the data, which can help to gain insights and make informed decisions.



### Activity 3: Splitting data into train and test and validation sets

We now need to load the training data for preprocessing and feeding to our model. So, we use tensorflow to load data from the dir. Now we need to use load\_data function which loads videos and alignments so we need to make a mappable function which does for each file in our data. Hence, we have a mappable function and the code to load the dataset. We also use the take and skip function to split dataset into training and validation sets.

```

1  def mappable_function(path:str) -> List[str]:
2      result = tf.py_function(load_data, [path], (tf.float32, tf.int64))
3      return result
4
5  data = tf.data.Dataset.list_files('/content/drive/MyDrive/data/s1/*.mpg')
6  data = data.shuffle(500, reshuffle_each_iteration=False)
7  data = data.map(mappable_function)
8  data = data.padded_batch(2, padded_shapes=([75, None, None, None],[40]))
9  data = data.prefetch(tf.data.AUTOTUNE)
10
11  ##Added for split
12  train = data.take(450)
13  test = data.skip(450)

```

Printing the pre-processed data:



```
1 frames, alignments = data.as_numpy_iterator().next()
2 print(frames)
3 print(alignments)
```

We have 500 videos in total so we use 450 for training and 50 for testing. The train and tests are randomly shuffled and splitted.

## Milestone 4: Model Building

### Activity 1: Importing necessary libraries



```
1 from tensorflow.keras.models import Sequential
2 from tensorflow.keras.layers import Conv3D, LSTM, Dense, Dropout, Bidirectional, MaxPool3D, Activation, TimeDistributed, Flatten
3 from tensorflow.keras.optimizers import Adam
4 from tensorflow.keras.callbacks import ModelCheckpoint, LearningRateScheduler
```

### Activity 2: Defining callbacks, Loss function and building the model

Now we define certain callbacks which will help the model during the training and also a loss function which calculates the loss after each epoch



```
1 def scheduler(epoch, lr):
2     if epoch < 30:
3         return lr
4     else:
5         return lr * tf.math.exp(-0.1)
6
7 def CTCloss(y_true, y_pred):
8     batch_len = tf.cast(tf.shape(y_true)[0], dtype="int64")
9     input_length = tf.cast(tf.shape(y_pred)[1], dtype="int64")
10    label_length = tf.cast(tf.shape(y_true)[1], dtype="int64")
11
12    input_length = input_length * tf.ones(shape=(batch_len, 1), dtype="int64")
13    label_length = label_length * tf.ones(shape=(batch_len, 1), dtype="int64")
14
15    loss = tf.keras.backend.ctc_batch_cost(y_true, y_pred, input_length, label_length)
16    return loss
17
18 checkpoint_callback = ModelCheckpoint(os.path.join('models', 'checkpoint'), monitor='loss', save_weights_only=True)
19 schedule_callback = LearningRateScheduler(scheduler)
```

**Model Building:** We are using a Deep Learning Bidirectional LSTM model, the model has 3D Convolution layers, Bidirectional LSTM layers and Dense Layers along with Flatten and Max Pooling. The model is as follows



```

1 model = Sequential()
2 model.add(Conv3D(128, 3, input_shape=(75,46,140,1), padding='same'))
3 model.add(Activation('relu'))
4 model.add(MaxPool3D((1,2,2)))
5
6 model.add(Conv3D(256, 3, padding='same'))
7 model.add(Activation('relu'))
8 model.add(MaxPool3D((1,2,2)))
9
10 model.add(Conv3D(75, 3, padding='same'))
11 model.add(Activation('relu'))
12 model.add(MaxPool3D((1,2,2)))
13
14 model.add(TimeDistributed(Flatten()))
15
16 model.add(Bidirectional(LSTM(128, kernel_initializer='Orthogonal', return_sequences=True)))
17 model.add(Dropout(.5))
18
19 model.add(Bidirectional(LSTM(128, kernel_initializer='Orthogonal', return_sequences=True)))
20 model.add(Dropout(.5))
21
22 model.add(Dense(char_to_num.vocabulary_size()+1, kernel_initializer='he_normal', activation='softmax'))

```

The model summary is as follows

```

model.summary()
[33]
... 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
conv3d_2 (Conv3D)	(None, 75, 11, 35, 75)	518475
activation_2 (Activation)	(None, 75, 11, 35, 75)	0
max_pooling3d_2 (MaxPooling3D)	(None, 75, 5, 17, 75)	0
time_distributed (TimeDistributed)	(None, 75, 6375)	0
bidirectional (Bidirectional)	(None, 75, 256)	6660096
dropout (Dropout)	(None, 75, 256)	0
bidirectional_1 (Bidirectional)	(None, 75, 256)	394240
dropout_1 (Dropout)	(None, 75, 256)	0
dense (Dense)	(None, 75, 41)	10537

```

=====
Total params: 8,471,924
Trainable params: 8,471,924
Non-trainable params: 0

```

We now compile the model and then train it using our training data

```
1 model.compile(optimizer=Adam(learning_rate=0.0001), loss=CTCLoss)
2 model.fit(train, validation_data=test, epochs=100, callbacks=[checkpoint_callback, schedule_callback])
```

Some screenshots of the training are as follows

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

```
Epoch 1/100
129/450 [=====>.....] - ETA: 3:48 - loss: 91.6139
[mpeg1video @ 0x78640801fd00] ac-tex damaged at 22 17
[mpeg1video @ 0x78640801fd00] Warning MVs not available
450/450 [=====] - ETA: 0s - loss: 86.6525
[mpeg1video @ 0x7863780635c0] ac-tex damaged at 22 17
[mpeg1video @ 0x7863780635c0] Warning MVs not available
450/450 [=====] - 562s 1s/step - loss: 86.6525 - val_loss: 126.5831 - lr: 0.0200
Epoch 2/100
52/450 [=>.....] - ETA: 4:38 - loss: 81.9282
[mpeg1video @ 0x58c00a0857c0] ac-tex damaged at 22 17
[mpeg1video @ 0x58c00a0857c0] Warning MVs not available
450/450 [=====] - ETA: 0s - loss: 82.4802
[mpeg1video @ 0x78633824df80] ac-tex damaged at 22 17
[mpeg1video @ 0x78633824df80] Warning MVs not available
450/450 [=====] - 534s 1s/step - loss: 82.4802 - val_loss: 117.2443 - lr: 0.0200
Epoch 3/100
120/450 [=====>.....] - ETA: 3:50 - loss: 81.1733
[mpeg1video @ 0x786378058300] ac-tex damaged at 22 17
[mpeg1video @ 0x786378058300] Warning MVs not available
450/450 [=====] - ETA: 0s - loss: 82.3892
[mpeg1video @ 0x7864e4078bc0] ac-tex damaged at 22 17
[mpeg1video @ 0x7864e4078bc0] Warning MVs not available
450/450 [=====] - 535s 1s/step - loss: 82.3892 - val_loss: 143.3543 - lr: 0.0200
Epoch 4/100
31/450 [=>.....] - ETA: 4:56 - loss: 86.6192
[mpeg1video @ 0x78631c0dffcc0] ac-tex damaged at 22 17
[mpeg1video @ 0x78631c0dffcc0] Warning MVs not available
450/450 [=====] - ETA: 0s - loss: 81.8530
[mpeg1video @ 0x786344082bc0] ac-tex damaged at 22 17
[mpeg1video @ 0x786344082bc0] Warning MVs not available
450/450 [=====] - 538s 1s/step - loss: 81.8530 - val_loss: 140.7193 - lr: 0.0200
Epoch 5/100
433/450 [=====>.....] - ETA: 11s - loss: 83.5178
[mpeg1video @ 0x78638403a0c0] ac-tex damaged at 22 17
[mpeg1video @ 0x78638403a0c0] Warning MVs not available
450/450 [=====] - ETA: 0s - loss: 83.4419
[mpeg1video @ 0x78630c082d40] ac-tex damaged at 22 17
[mpeg1video @ 0x78630c082d40] Warning MVs not available
450/450 [=====] - 536s 1s/step - loss: 83.4419 - val_loss: 135.2464 - lr: 0.0200
Epoch 6/100
242/450 [=====>.....] - ETA: 2:25 - loss: 81.1992
[mpeg1video @ 0x7863200b84c0] ac-tex damaged at 22 17
[mpeg1video @ 0x7863200b84c0] Warning MVs not available
450/450 [=====] - ETA: 0s - loss: 76.3918
[mpeg1video @ 0x7864e08346c0] ac-tex damaged at 22 17
[mpeg1video @ 0x7864e08346c0] Warning MVs not available
450/450 [=====] - 532s 1s/step - loss: 76.3918 - val_loss: 131.5368 - lr: 2.4555e-04
Epoch 75/100
450/450 [=====] - ETA: 0s - loss: 75.8247
[mpeg1video @ 0x786340425840] ac-tex damaged at 22 17
[mpeg1video @ 0x786340425840] Warning MVs not available
450/450 [=====] - 532s 1s/step - loss: 75.8247 - val_loss: 130.2387 - lr: 2.2218e-04
Epoch 76/100
324/450 [=====>.....] - ETA: 1:27 - loss: 76.2352
[mpeg1video @ 0x786418010b80] ac-tex damaged at 22 17
[mpeg1video @ 0x786418010b80] Warning MVs not available
450/450 [=====] - ETA: 0s - loss: 76.1636
[mpeg1video @ 0x58c00a08d18c0] ac-tex damaged at 22 17
[mpeg1video @ 0x58c00a08d18c0] Warning MVs not available
450/450 [=====] - 532s 1s/step - loss: 76.1636 - val_loss: 128.9099 - lr: 2.0104e-04
Epoch 77/100
332/450 [=====>.....] - ETA: 1:22 - loss: 76.1289
[mpeg1video @ 0x78618927b680] ac-tex damaged at 22 17
[mpeg1video @ 0x78618927b680] Warning MVs not available
450/450 [=====] - ETA: 0s - loss: 75.9525
[mpeg1video @ 0x78637c039f40] ac-tex damaged at 22 17
[mpeg1video @ 0x78637c039f40] Warning MVs not available
450/450 [=====] - 532s 1s/step - loss: 75.9525 - val_loss: 129.5591 - lr: 1.8191e-04
Epoch 78/100
166/450 [=====>.....] - ETA: 3:17 - loss: 76.4204
[mpeg1video @ 0x7864dc038340] ac-tex damaged at 22 17
[mpeg1video @ 0x7864dc038340] Warning MVs not available
450/450 [=====] - ETA: 0s - loss: 76.2397
[mpeg1video @ 0x786324080280] ac-tex damaged at 22 17
[mpeg1video @ 0x786324080280] Warning MVs not available
450/450 [=====] - 532s 1s/step - loss: 76.2397 - val_loss: 130.4434 - lr: 1.6459e-04
Epoch 79/100
450/450 [=====] - ETA: 0s - loss: 76.4566
[mpeg1video @ 0x7863e805ad80] ac-tex damaged at 22 17
[mpeg1video @ 0x7863e805ad80] Warning MVs not available
450/450 [=====] - 534s 1s/step - loss: 76.4566 - val_loss: 131.6600 - lr: 1.4893e-04
Epoch 80/100
```

## Milestone 5: Model Deployment

### Activity 1: Save the model

After the model has been completed training, the weights are already saved using the checkpoint callback, now save the model using **model.save()** to save it to a keras format to use while prediction in future

```
1 model.save("LipReading.keras")
```

## Activity 2: Make a Prediction

Now load the model and test using the loaded model.

Loading test data

```
1 model = tf.keras.models.load_model("LipReading.keras")
2 test_data = test.as_numpy_iterator()
3 sample = test_data.next()
```

Testing using the data

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

1/1 [=====] - 2s 2s/step

print('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'bin white with b zero please'>,
 <tf.Tensor: shape=(), dtype=string, numpy=b'place blue by v nine again'>]

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

print('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'bin white with b zero please'>,
 <tf.Tensor: shape=(), dtype=string, numpy=b'place blue by v nine again'>]
```

If your prediction text is matching real text then move forward for the next step. The model works well for the data.

If not, train your model again changing the layers a bit or the hyperparameters.

## 7.2 Feature 2

### Milestone 6: App Development

#### Activity 1: Design and Develop an Android App as the front end to get the video

The android App has been coded using Flutter and the code is in the project repo.

#### Activity 2: Make an API End point using Flask and deploy it to Google Cloud Run

The API End point was created using Flask and the program was containerized and deployed to google cloud

main.py containing the endpoint

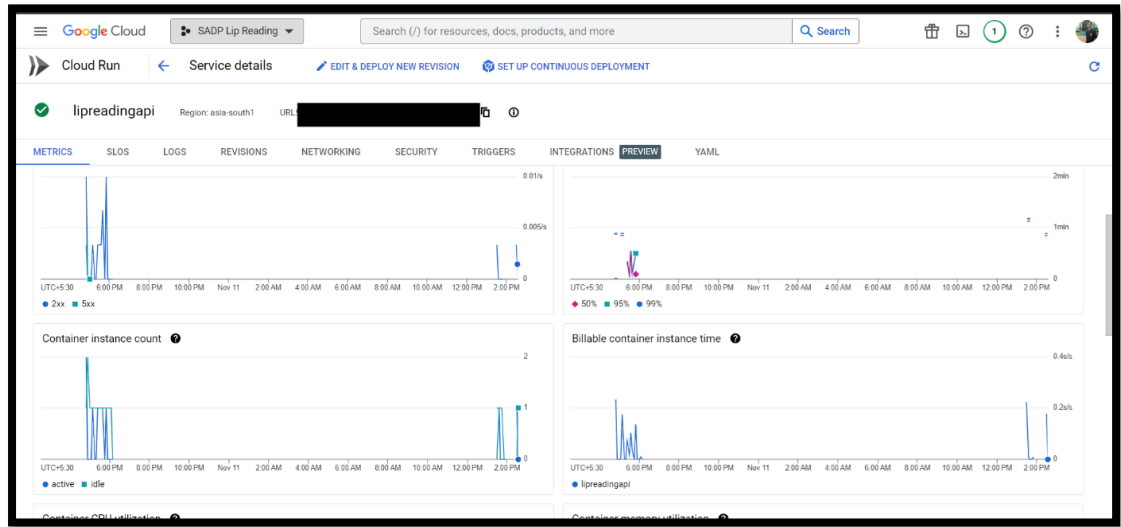
```
1  from flask import Flask, jsonify, request
2  import tensorflow as tf
3  import os,shutil
4  from utils import load_data,load_video,add_empty_frames,num_to_char,CTCLoss,split_video
5
6  model = tf.keras.models.load_model("./LipReading.keras",custom_objects={"CTCLoss":CTCLoss})
7  app = Flask(__name__)
8  def predict(video_path):
9      test_path = video_path
10     print(test_path)
11     sample = load_video(test_path)
12
13     if(len(sample)<75):
14         print("Frames < 75")
15         add_empty_frames(test_path,"output.mp4",75)
16         test_path = "output.mp4"
17         sample = load_video(test_path)
18         y_pred = model.predict(tf.expand_dims(sample, axis=0))
19         decoded = tf.keras.backend.ctc_decode(y_pred, input_length=[75], greedy=True)[0][0].numpy()
20         final_pred = tf.strings.reduce_join(num_to_char(decoded)).numpy().decode('utf-8')
21     elif(len(sample)>75):
22         print("Frames >75")
23         split_video(test_path,"output_75")
24         vals=[]
25         for i in os.listdir("output_75"):
26             # print(i)
27             sample = load_video(f"output_75/{i}")
28             vals.append(sample)
29         final_pred=""
30         for i in vals:
31             yhat = model.predict(tf.expand_dims(i, axis=0))
32             decoded = tf.keras.backend.ctc_decode(yhat, input_length=[75], greedy=True)[0][0].numpy()
33             preds = tf.strings.reduce_join(num_to_char(decoded)).numpy().decode('utf-8')
34             # print(final_pred)
35             final_pred+=f"{preds} "
36         shutil.rmtree("output_75")
37     else:
38         y_pred = model.predict(tf.expand_dims(sample, axis=0))
39         decoded = tf.keras.backend.ctc_decode(y_pred, input_length=[75], greedy=True)[0][0].numpy()
40         final_pred = tf.strings.reduce_join(num_to_char(decoded)).numpy().decode('utf-8')
41
42     print(final_pred)
43     return final_pred
44
45 @app.route('/predict', methods=['POST','GET'])
46 def prediction_endpoint():
47     if 'file' not in request.files:
48         return jsonify({'error': 'No file part'})
49
50     file = request.files['file']
51
52     if file.filename == '':
53         return jsonify({'error': 'No selected file'})
54
55     try:
56         video_path = "uploaded_video.mp4"
57         file.save(video_path)
58         result = predict(video_path)
59         os.remove(video_path) # Remove the uploaded video after processing
60         return jsonify({'prediction': result})
61
62     except Exception as e:
63         return jsonify({'error': str(e)})
64
65
66 if __name__ == '__main__':
67     app.run(debug=True)
68
```

Now after this use the following commands on the google cloud shell and deploy it.

```
1 gcloud builds submit --tag gcr.io/meta-gateway-*****/prediction_endpoint
2 gcloud run deploy --image gcr.io/meta-gateway-*****/prediction_endpoint --platform managed
```

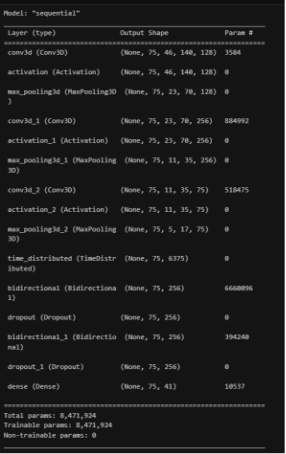
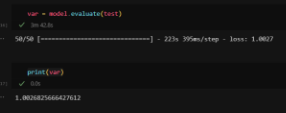
where **meta-gateway-\*\*\*\*\*** is the project id of the cloud project and **prediction\_endpoint** is the function name which predicts the output

The cloud run is as follows



## 8. PERFORMANCE TESTING

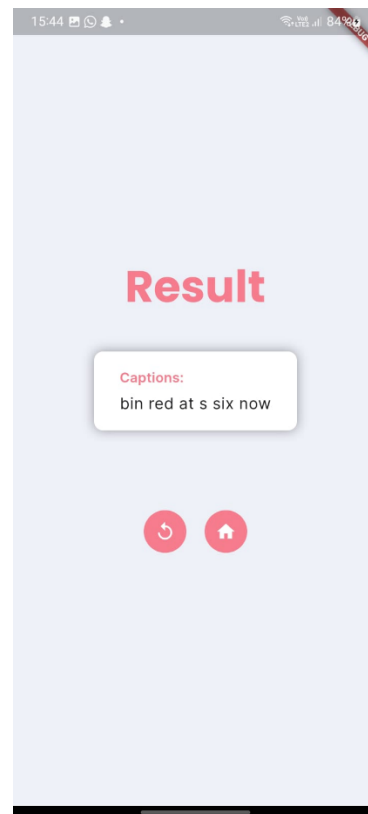
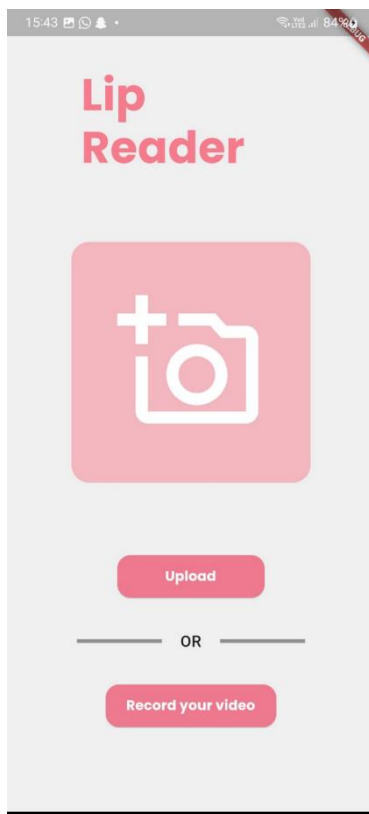
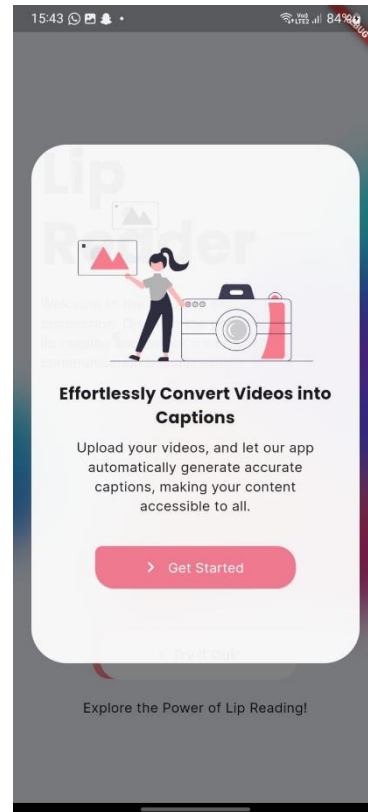
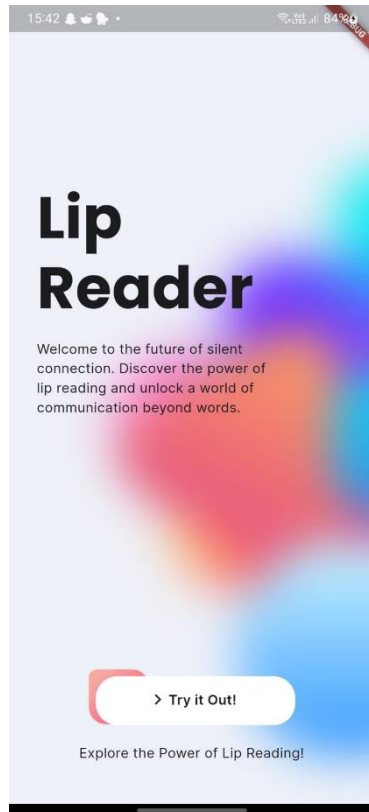
### 8.1 Performance Metrics

S.No.	Parameter	Values	Screenshot
1.	Model Summary	Total params: 8,471,924 Trainable params: 8,471,924 Non-trainable params: 0	
2.	Accuracy	Training Loss - 2.085 Acc(approx) - 97.913  Validation Loss - 3.0027 Acc (approx) - 96.8	
3.	Confidence Score (Only Yolo Projects)	Class Detected - NA  Confidence Score - NA	Not Applicable

## 9. RESULTS

### 9.1 Output Screenshots

#### Working of the App



## 10. ADVANTAGES & DISADVANTAGES

### Advantages of Lip Reading Using Deep Learning:

**Improved Speech Recognition:** Lip reading can enhance traditional audio-based speech recognition systems, especially in noisy environments or situations where the audio signal is unclear.

**No Dependency on Audio Data:** Unlike traditional speech recognition models, lip reading systems can be trained solely on video data, eliminating the need for transcribed audio, which can be expensive and time-consuming to obtain.

**Multi-Modal Applications:** The integration of lip reading with audio-based systems enables the development of multi-modal applications, improving real-time communication accuracy and transcriptions in scenarios like video conferencing.

**Accessibility for Hearing-Impaired Individuals:** Accurate lip-reading systems serve as an essential communication tool for individuals with hearing impairments, enhancing their ability to understand spoken language and participate in conversations.

**Robustness in Diverse Environments:** Lip reading models, especially when coupled with deep learning algorithms, can exhibit robust performance across various environments, handling challenges such as multiple speakers, accents, and background noise.

**Privacy Preservation:** Since lip reading relies on visual information, it can be less intrusive in terms of privacy compared to audio-based systems, addressing concerns related to audio data collection.

### Disadvantages of Lip Reading Using Deep Learning:

**Challenges in Variability:** Lip movements can vary significantly among individuals, making it challenging to create a universally accurate model that accommodates diverse lip shapes, sizes, and movements.

**Limited Vocabulary:** Training a lip-reading model for a broad vocabulary may pose challenges, and the system might be more effective with a specific set of words rather than an extensive lexicon.

**Data Quality and Quantity:** The success of deep learning models depends on the availability of high-quality and diverse training data. Acquiring a large dataset of labelled video samples can be resource-intensive and may require substantial manual effort.

**Computational Intensity:** Deep learning models, particularly those involving recurrent neural networks like LSTMs, can be computationally intensive, requiring powerful hardware for training and inference.

**Dependency on Lighting and Camera Quality:** The accuracy of lip-reading systems can be influenced by lighting conditions and the quality of the video input. Poor lighting or low-resolution cameras may negatively impact performance.

**Real-Time Processing Challenges:** Achieving real-time processing capabilities may be challenging, especially in applications where low latency is crucial, as deep learning models may require significant computational time.

**Ethical Concerns:** The deployment of lip-reading technology raises ethical concerns related to privacy, as the visual nature of the data may lead to unintended intrusions into personal spaces.

**Potential Bias:** If not carefully curated, training datasets may introduce biases, affecting the model's performance, especially concerning diverse demographics and accents.

## 11. CONCLUSION

In conclusion, the development of a lip-reading system using deep learning algorithms, such as LSTM and Neural Networks, presents a promising avenue for enhancing speech recognition and communication technologies. The integration of visual cues from lip movements with traditional audio-based systems offers several advantages, including improved accuracy, accessibility for hearing-impaired individuals, and the potential for multi-modal applications.

The project aims to address existing challenges in audio-based systems, such as their dependency on transcribed audio data and susceptibility to noisy environments. By training the system solely on video data, the proposed solution eliminates the need for transcribed audio, making the training process more efficient and cost-effective.

The functional requirements encompass critical aspects, including video input processing, deep learning model implementation, integration with audio-based systems, and user interface development. These requirements ensure that the system is not only accurate but also user-friendly and capable of real-time processing.

On the non-functional front, considerations such as accuracy, scalability, robustness, and security are paramount. Achieving a high level of accuracy in diverse environments, ensuring the system can scale to handle increased demands, and maintaining robust performance are crucial for the success of the lip-reading system.

While there are notable advantages, it's essential to acknowledge potential challenges, including variability in lip movements, limitations in vocabulary, and computational intensity. The ethical considerations related to privacy and the potential for bias in training datasets also warrant careful attention.

## 12. FUTURE SCOPE

The future scope of a lip-reading system based on deep learning is promising and holds the potential for further advancements and applications. Some key areas of future development include:

### **Enhanced Accuracy and Vocabulary Expansion:**

Continuous research and development can lead to improvements in the accuracy of lip-reading models, particularly in handling diverse lip movements and expanding the vocabulary they can effectively recognize.



**Integration with Emerging Technologies:**

Integration with other emerging technologies, such as augmented reality (AR) and virtual reality (VR), can enhance the user experience. Lip reading systems could be integrated into AR/VR communication platforms, providing more immersive and inclusive interactions.

**Real-Time Processing Optimization:**

Ongoing efforts can be directed towards optimizing the real-time processing capabilities of lip-reading systems. This is particularly important for applications where low latency is critical, such as in live captioning for broadcasting or video conferencing.

**Cross-Language Lip Reading:**

Future developments may focus on creating lip reading models capable of understanding and transcribing speech in multiple languages. This would significantly broaden the applicability of the technology in diverse linguistic environments.

**Mobile and Edge Device Integration:**

As computational capabilities of mobile and edge devices continue to advance, there is potential for deploying lightweight versions of lip-reading models on these devices. This could enable on-device lip reading applications, enhancing privacy and reducing dependence on centralized processing.

**Adaptive Learning and Personalization:**

Future systems may incorporate adaptive learning mechanisms to personalize the lip-reading model for individual users. This could improve recognition accuracy by adapting to specific lip movements and speech patterns unique to each user.

**Gesture and Facial Expression Recognition:**

Expansion into recognizing not only lip movements but also facial expressions and gestures can provide a more comprehensive understanding of communication. This could be especially beneficial in applications where non-verbal cues are crucial, such as human-computer interaction or virtual assistant technologies.

**Ethical and Privacy Considerations:**

Continued research is essential to address ethical concerns related to privacy, consent, and potential biases in lip reading systems. Developing transparent and accountable practices in deploying these technologies is crucial for gaining public trust.

**Collaboration with Healthcare:**

Lip reading systems could find applications in the healthcare sector, assisting healthcare professionals in understanding patients with speech disorders or providing valuable tools for speech therapy.

**Multimodal Fusion for Redundancy:**

Integration with other modalities, such as gesture recognition or head movements, can provide redundancy and improve overall system reliability, especially in challenging scenarios.

**Global Collaboration and Dataset Diversity:**

Collaboration across research communities globally and the creation of diverse datasets that encompass various languages, accents, and cultural contexts will contribute to the robustness and inclusivity of lip-reading systems.

The future scope of lip reading using deep learning is dynamic and expansive, with the potential to revolutionize communication technologies, accessibility, and human-computer interaction. Continued research, technological innovation, and interdisciplinary collaboration will play pivotal roles in shaping the evolution of these systems.

### 13. APPENDIX

#### Source Code:

- **GitHub Repo Link:** <https://github.com/smartinternz02/SI-GuidedProject-602925-1697547982/tree/main>
- **Project Demo Link:** <https://drive.google.com/file/d/18LV5tCCH42HYs5Ien6NhcTUSiE-Ved9d/view>