

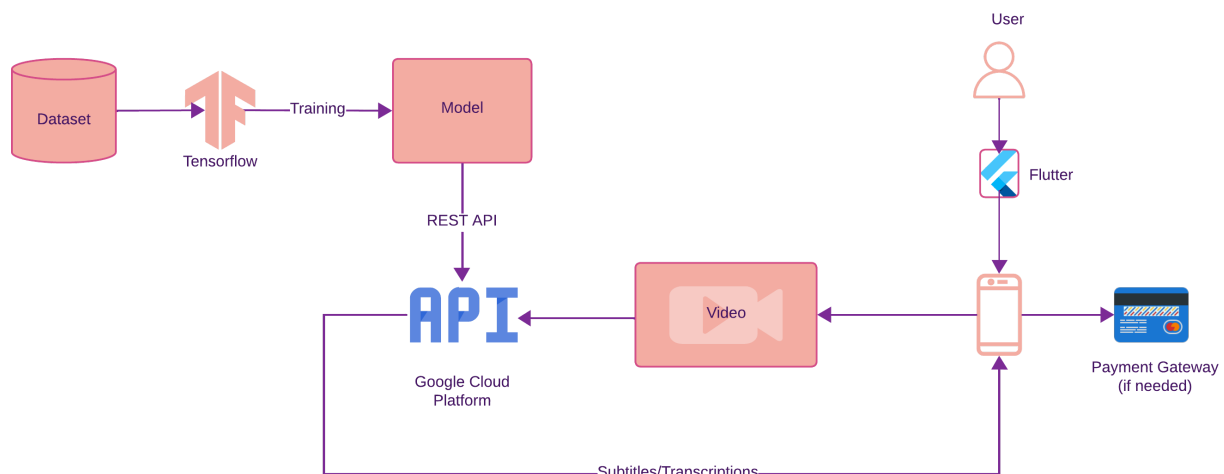
Lip Reading Using Deep Learning

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.
4. **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.

Technical Architecture



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 analyzing 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 send to the model on the Google Cloud using the API.

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

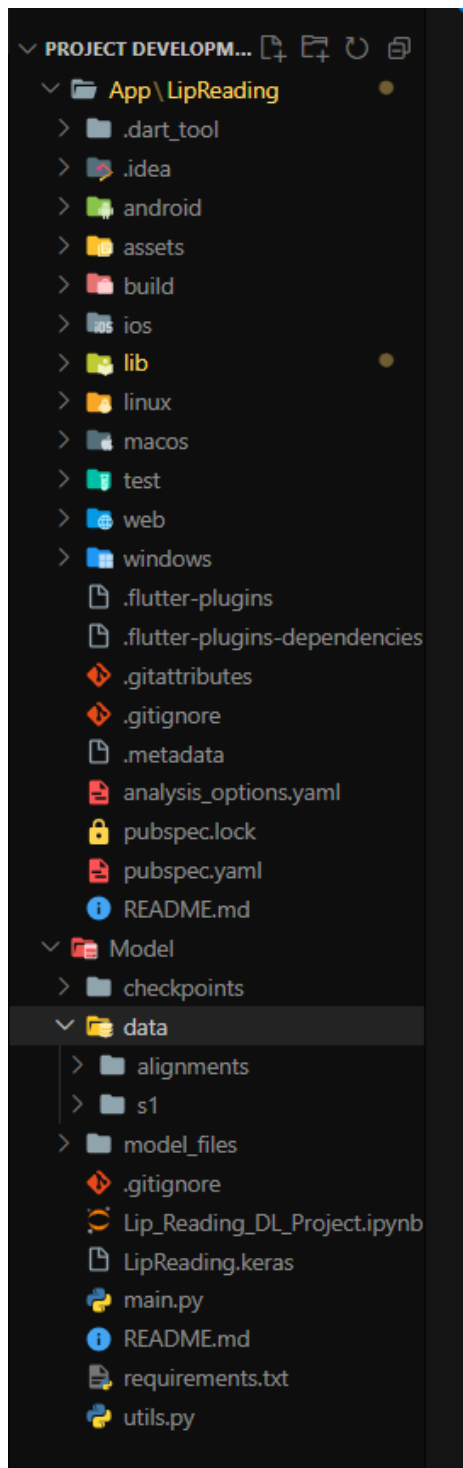
Once the model analyzes the input, the prediction is sent back to the Android App, which is then displayed to the user.

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 an Flutter Application folder for the development of the Android App
- We have our saved model as **LipReading.keras** and we have out 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/rishisrddy/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_BGR2RGB)
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

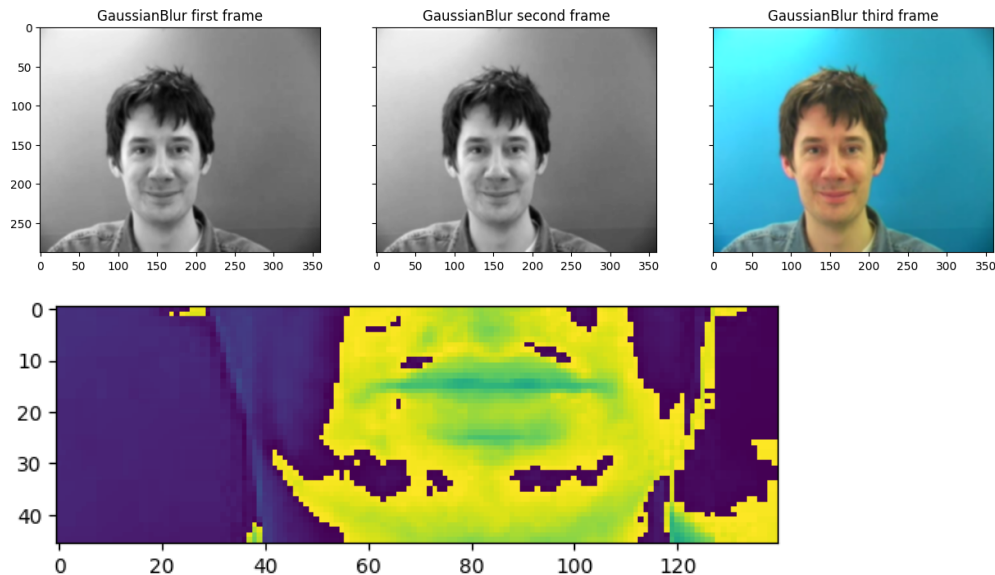
Activity 1: Descriptive statistical

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 preprocessed 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, Birdirectional 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 out 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
[mpeglvideo @ 0x7864d801f000] ac-tex damaged at 22 17
[mpeglvideo @ 0x7864d801f000] Warning MVs not available
450/450 [=====] - ETA: 0s - loss: 86.6525
[mpeglvideo @ 0x7863780635c0] ac-tex damaged at 22 17
[mpeglvideo @ 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
[mpeglvideo @ 0x58c00a8057c0] ac-tex damaged at 22 17
[mpeglvideo @ 0x58c00a8057c0] Warning MVs not available
450/450 [=====] - ETA: 0s - loss: 82.4802
[mpeglvideo @ 0x78633824df80] ac-tex damaged at 22 17
[mpeglvideo @ 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
[mpeglvideo @ 0x786378058300] ac-tex damaged at 22 17
[mpeglvideo @ 0x786378058300] Warning MVs not available
450/450 [=====] - ETA: 0s - loss: 82.3892
[mpeglvideo @ 0x7864e4078bc0] ac-tex damaged at 22 17
[mpeglvideo @ 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
[mpeglvideo @ 0x78631c0dff00] ac-tex damaged at 22 17
[mpeglvideo @ 0x78631c0dff00] Warning MVs not available
450/450 [=====] - ETA: 0s - loss: 81.8530
[mpeglvideo @ 0x786344082bc0] ac-tex damaged at 22 17
[mpeglvideo @ 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
[mpeglvideo @ 0x78630403a0c0] ac-tex damaged at 22 17
[mpeglvideo @ 0x78630403a0c0] Warning MVs not available
450/450 [=====] - ETA: 0s - loss: 83.4419
[mpeglvideo @ 0x78630c082040] ac-tex damaged at 22 17
[mpeglvideo @ 0x78630c082040] 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
[mpeglvideo @ 0x7863200b84c0] ac-tex damaged at 22 17
[mpeglvideo @ 0x7863200b84c0] Warning MVs not available

450/450 [=====] - ETA: 0s - loss: 76.3918
[mpeglvideo @ 0x7864e00346c0] ac-tex damaged at 22 17
[mpeglvideo @ 0x7864e00346c0] 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
[mpeglvideo @ 0x786340425840] ac-tex damaged at 22 17
[mpeglvideo @ 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
[mpeglvideo @ 0x786418010b80] ac-tex damaged at 22 17
[mpeglvideo @ 0x786418010b80] Warning MVs not available
450/450 [=====] - ETA: 0s - loss: 76.1636
[mpeglvideo @ 0x58c00a8d18c0] ac-tex damaged at 22 17
[mpeglvideo @ 0x58c00a8d18c0] 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
[mpeglvideo @ 0x78618927b680] ac-tex damaged at 22 17
[mpeglvideo @ 0x78618927b680] Warning MVs not available
450/450 [=====] - ETA: 0s - loss: 75.9525
[mpeglvideo @ 0x78637c039f40] ac-tex damaged at 22 17
[mpeglvideo @ 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
[mpeglvideo @ 0x7864dc038340] ac-tex damaged at 22 17
[mpeglvideo @ 0x7864dc038340] Warning MVs not available
450/450 [=====] - ETA: 0s - loss: 76.2397
[mpeglvideo @ 0x786324080280] ac-tex damaged at 22 17
[mpeglvideo @ 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
[mpeglvideo @ 0x7863e805ad80] ac-tex damaged at 22 17
[mpeglvideo @ 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

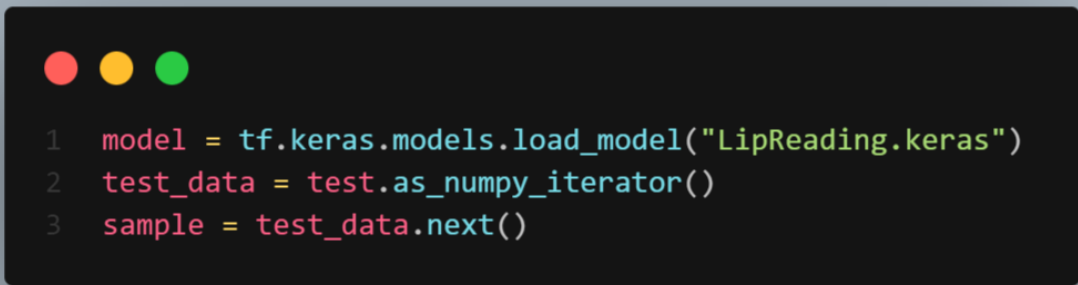


```
1 model.save("LipReading.keras")
```

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.

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.

App Screenshots at the end of the file

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

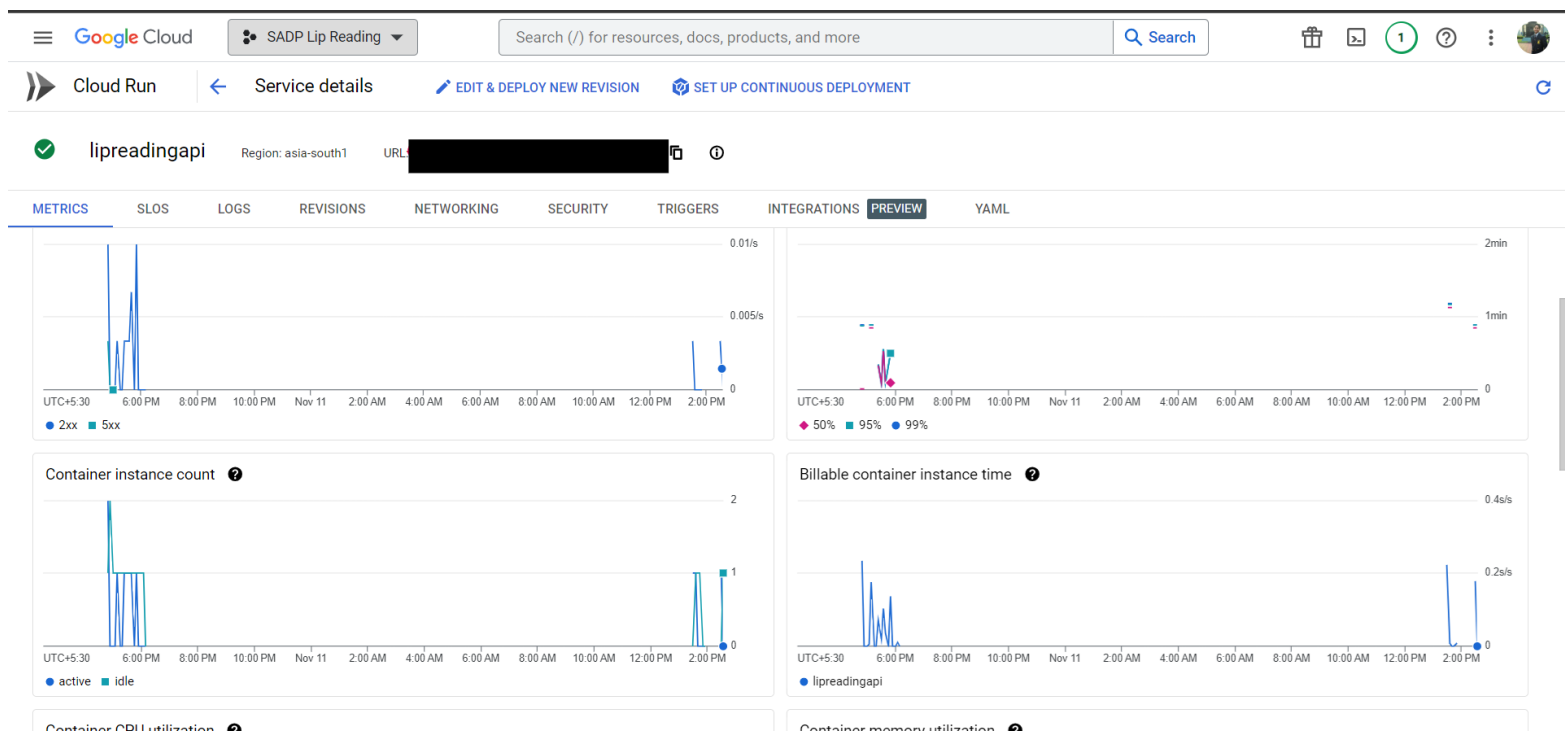
```
1 from flask import Flask, jsonify, request
2 import tensorflow as tf
3 import os,shutil
4 from utils import load_data,load_video,add_empty_frames,num_to_char,CTCLoss,split_video
5
6 model = tf.keras.models.load_model("./LipReading.keras",custom_objects={"CTCLoss":CTCLoss})
7 app = Flask(__name__)
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+="{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.

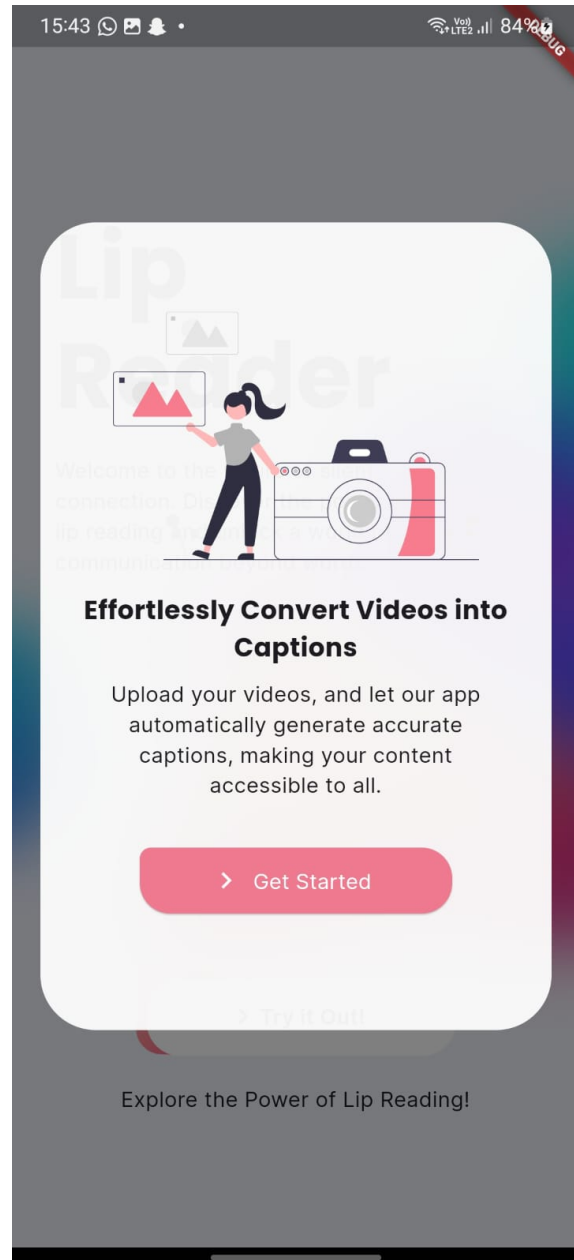
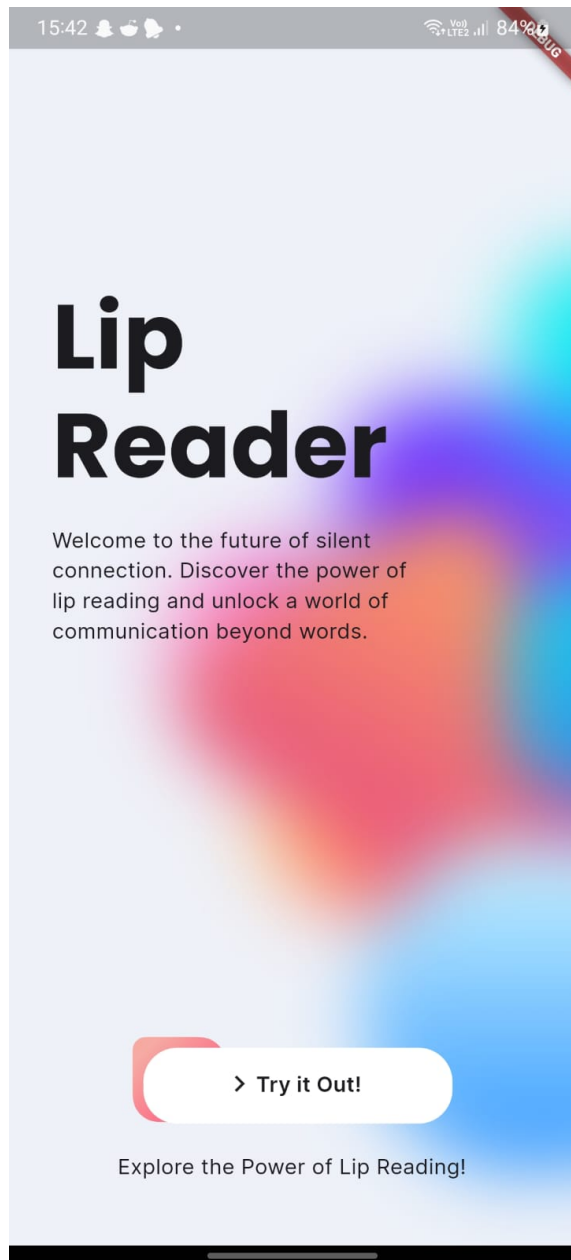
```
1 gcloud builds submit --tag gcr.io/meta-gateway-[redacted]/prediction_endpoint
2 gcloud run deploy --image gcr.io/meta-gateway-[redacted]/prediction_endpoint --platform managed
```

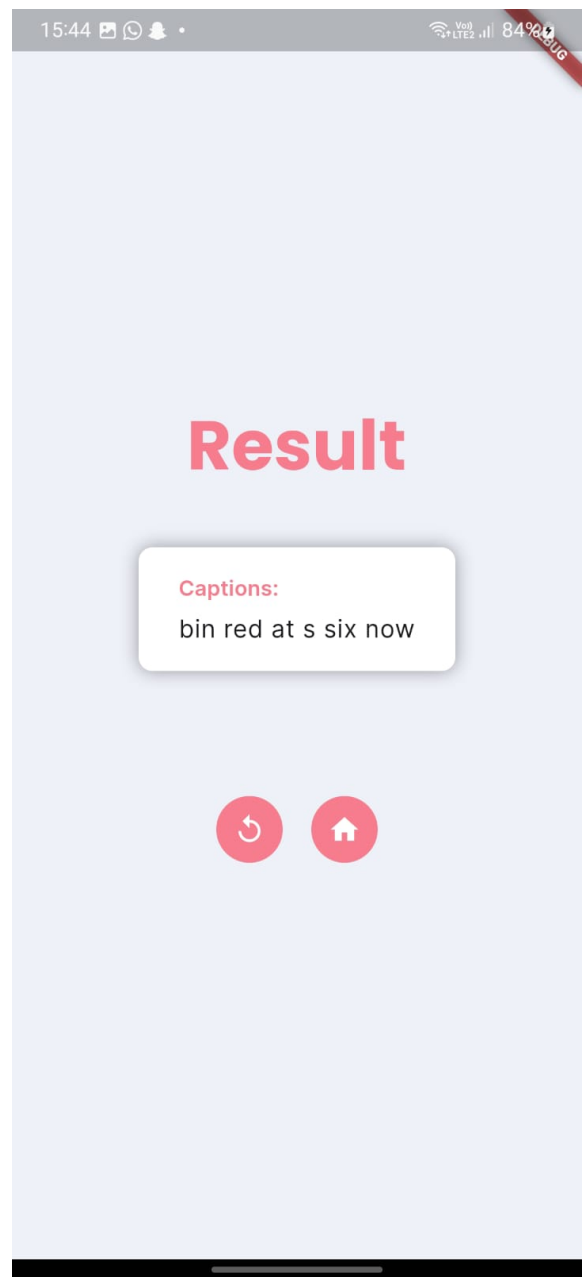
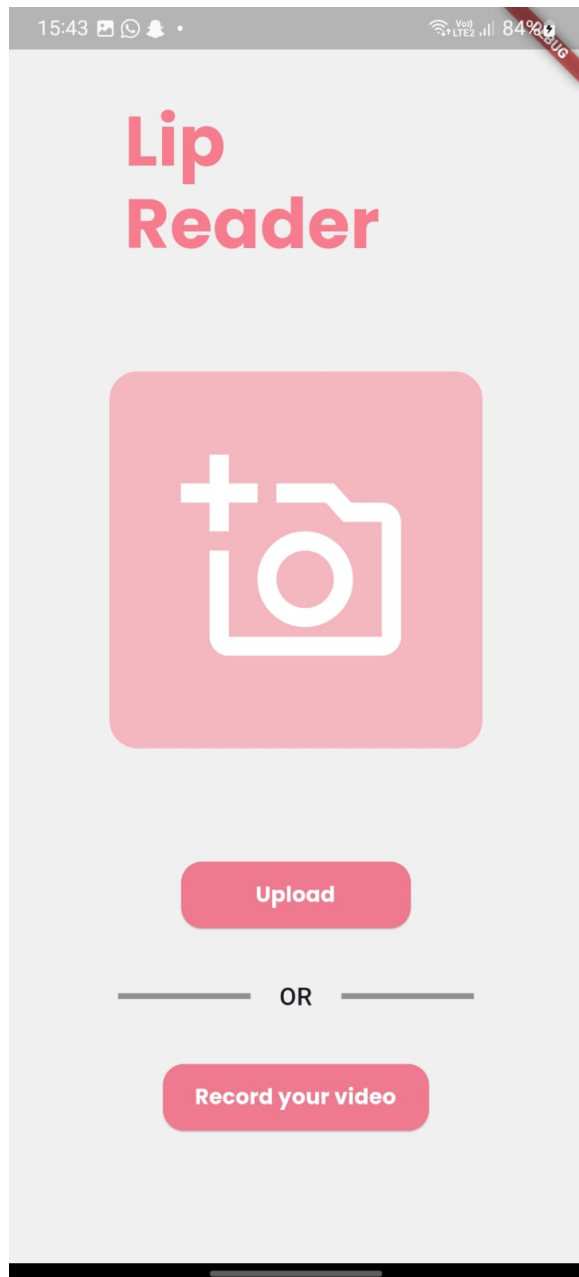
where **meta-gateway-******* is the project id of the cloud project and **prediction_endpoint** is the function name which predicts the output

The cloud run is as follows



Working of the App





Milestone 7: Project Demonstration & Documentation

Below mentioned deliverables are submitted along with other deliverables.

Activity 1: - Record explanation Video for project end to end solution.

Submitted in the link provided

Activity 2: - Project Documentation-Step by step project development procedure.

Create a document as per the template provided.

Submitted in the link provided