

Lip Reading Using Deep Learning

Team ID : Team-593092

Members:

- **Aman Barnawal : 21BCE0649 (VIT V)**
- **Abhigyan Satya Das : 21BCE0622 (VIT V)**
- **Rishabh Sharma : 21BCE0943 (VIT V)**
- **Jai Gaurav : 21BCE7193 (VIT AP)**

1. Introduction

1.1) Project Overview

Lip reading is a captivating and intricate field within computer vision and machine learning that seeks to harness visual information from lip movements to comprehend spoken language. This unique ability holds immense potential across various applications, particularly in environments characterized by high noise levels or for individuals with hearing impairments. By developing a sophisticated lip-reading model, this project aims to bridge communication gaps and enhance accessibility through the utilization of Convolutional Neural Networks (Conv3D) and Long Short-Term Memory Networks (LSTM). The incorporation of these neural network architectures allows for the extraction and understanding of spatiotemporal features embedded in lip movement data, paving the way for innovative solutions in communication technology.

1.2) Purpose

The primary impetus behind advancing lip-reading technology is to enhance accessibility for individuals with hearing impairments. For those who rely on visual cues, lip reading can serve as a valuable tool to comprehend spoken language, enabling a more inclusive and seamless communication experience.

In environments with high levels of noise or audio interference, traditional speech recognition systems may struggle to accurately transcribe spoken words. Lip reading provides an alternative or complementary method, offering resilience in situations where auditory signals may be compromised.

Lip reading technology has the potential to facilitate silent communication in public spaces. In scenarios where maintaining silence is essential, such as libraries or crowded public transport, individuals can use lip reading to convey or receive information without the need for audible speech.

Lip reading technology can contribute to breaking down language barriers. It becomes particularly valuable in multilingual contexts where understanding spoken language might pose a challenge. By relying on visual cues, individuals can overcome linguistic differences and communicate effectively.

2. Literature Survey

2.1) Existing Problem:

The absence of efficient lip-reading technology presents significant challenges for individuals with hearing impairments, hindering their complete engagement in various settings like social, educational, and professional environments. This results in barriers to effective communication and limits access to vital information conveyed through spoken language. Consequently, educational opportunities, employment prospects, and social inclusion are adversely affected. The absence of lip-reading technology amplifies isolation and reliance on alternative communication methods, restricting independence and diminishing the quality of life for those facing hearing challenges. The development of robust lip-reading solutions stands as a pivotal step toward enhancing accessibility and empowerment in their daily lives.

2.2) References:

<https://www.atlassian.com/agile/project-management>

<https://www.atlassian.com/agile/tutorials/how-to-do-scrum-with-jira-software>

<https://www.atlassian.com/agile/tutorials/epics> <https://www.atlassian.com/agile/tutorials/sprints>

<https://www.atlassian.com/agile/project-management/estimation>

<https://www.atlassian.com/agile/tutorials/burndown-charts>

<https://www.w3schools.com/>

<https://youtube.com/>

<https://github.com/>

2.3) Problem Statement Definition:

To convert lip movements into text, with the overarching aim of augmenting accessibility and empowering individuals facing hearing challenges across varied social, educational, and professional contexts.

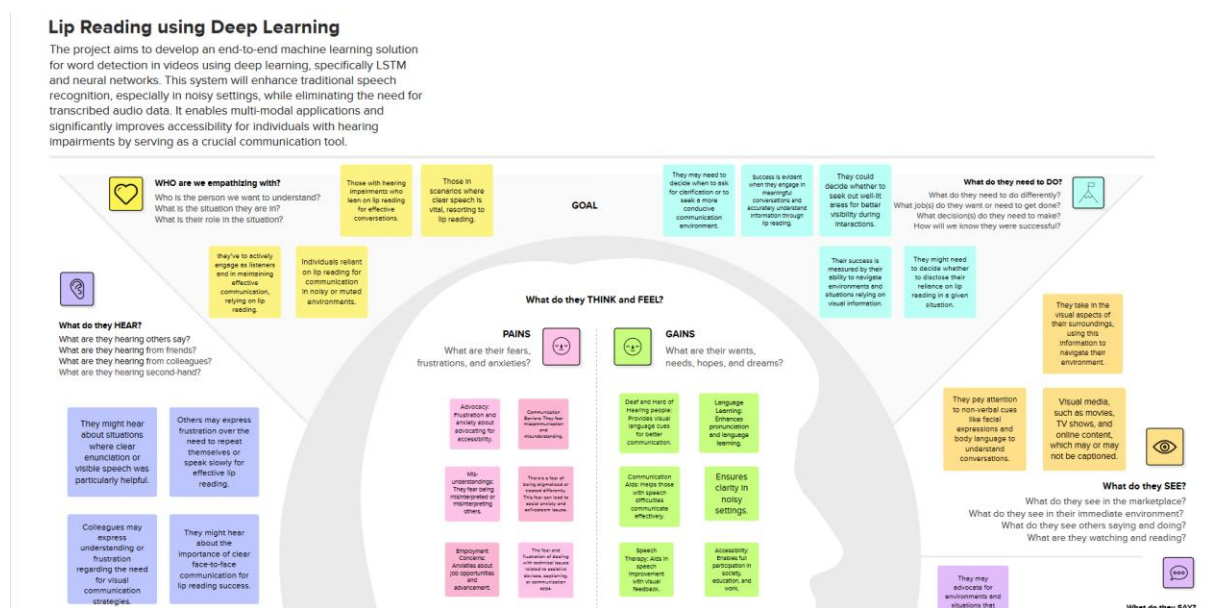
2.4) Objectives:

The primary objectives of this project are as follows:

- Develop a robust data loading pipeline for lip-reading datasets.
- Implement a deep learning model architecture combining Conv3D and LSTM layers.
- Train the model on lip-reading data for accurate predictions.
- Evaluate the model's performance on test samples and a new video.
- Demonstrate the practicality and efficacy of the developed lip-reading model.

3. Ideation & Proposed Solution

3.1) Empathy Map Canvas





3.2) Ideation & Brainstorming:

[Mural Link](#)

4. Requirement Analysis

4.1) Functional Requirement

- Accurately analyze and interpret English lip movements in Jupyter Notebook.
- Convert interpreted English lip movements into corresponding textual representations.
- Accommodate variations in English lip movements across different speakers within Jupyter Notebook.
- Support real-time processing for dynamic communication within the Jupyter Notebook environment.
- Adapt to diverse scenarios, including social interactions and education, focusing on the English language.
- Provide a user interface for interaction and English textual output within Jupyter Notebook.
- Enable continuous learning and improvement for enhanced accuracy within Jupyter Notebook constraints.
- Scale to handle a diverse dataset of English lip movements for robust model training within Jupyter Notebook.
- Prioritize user privacy and adhere to ethical standards for handling sensitive data in Jupyter Notebook.
- As this is a model based on characters, it doesn't depend on audio inputs and remains unaffected by variations in languages.

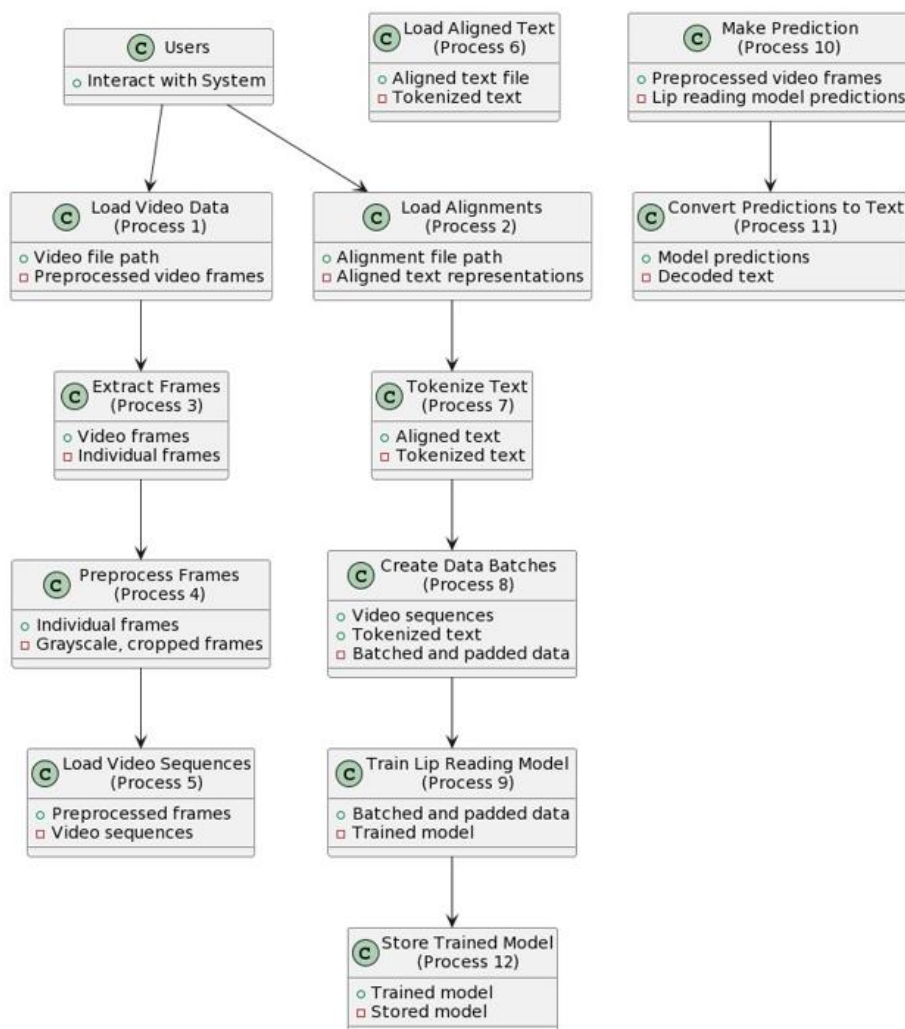
4.2) Non-Functional requirements

The system should achieve real-time processing with minimal latency. It must handle multiple concurrent users within the Jupyter Notebook environment efficiently. The system should scale seamlessly to accommodate a growing dataset and increasing user demands. It must maintain performance levels as the dataset expands.

The system should operate consistently without frequent disruptions. It must have mechanisms for error handling and recovery to ensure uninterrupted service. The user interface should be intuitive and user-friendly within the Jupyter Notebook interface. The codebase should be well-documented and organized for ease of maintenance. The system should allow for straightforward updates and enhancements.

5. Project Design

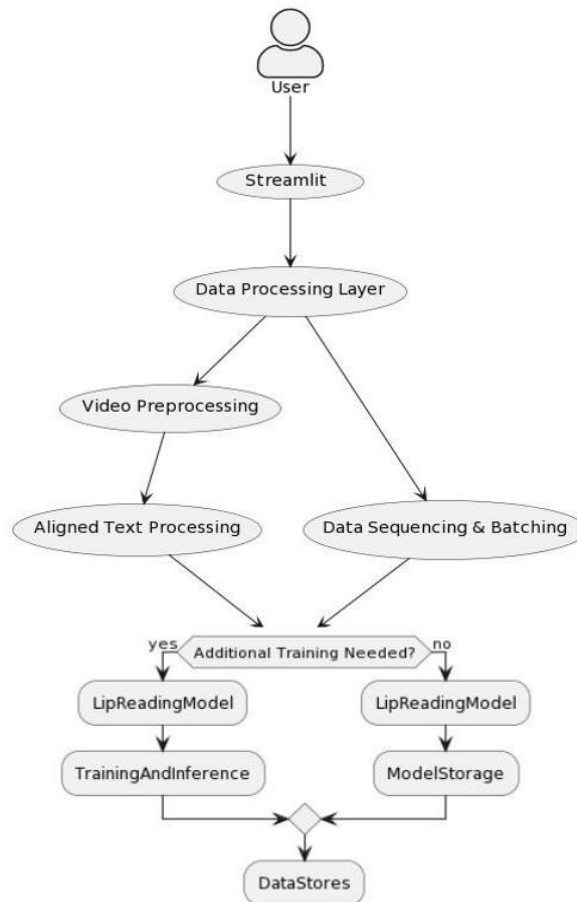
5.1) Data Flow Diagram & User Stories



5.2) Solution Architecture

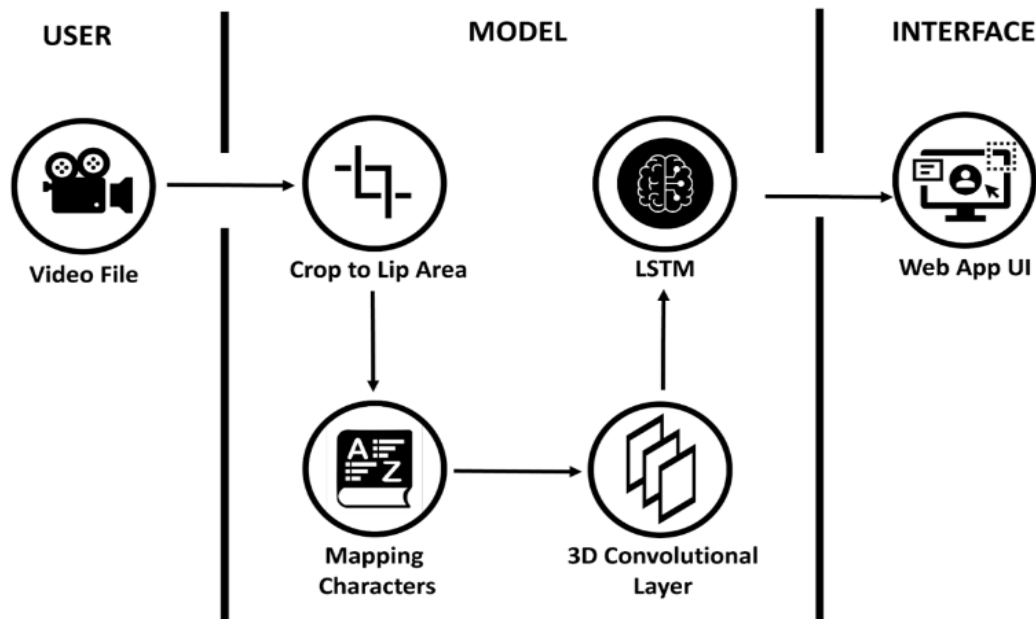
Project Planning Template:

Solution Architecture Diagram



6. Project Planning & Scheduling

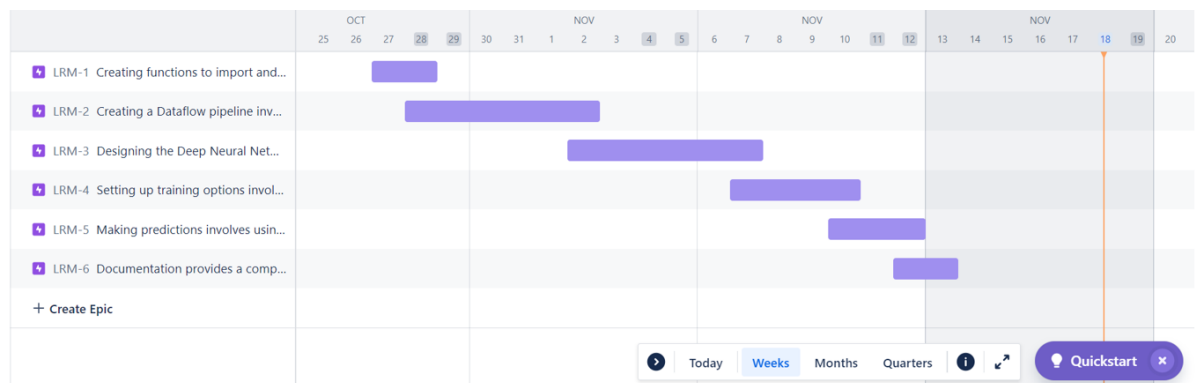
6.1) Technical Architecture



6.2) Sprint Planning & Estimation:

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-1	Build Data Loading Functions	USN-1	Creating functions to import and preprocess video data for training a lip-reading model, encompassing steps like downloading video files, extracting relevant frames, and aligning them with corresponding transcriptions.	2	High	Rishabh Sharma
Sprint-2		USN-2	Creating a Dataflow pipeline involves structuring a streamlined process for loading, preprocessing, and batching video and alignment data, optimizing it for efficient training of the lip-reading model.	5	High	Aman Barnawal
Sprint-3	Designing of the Deep Neural Network	USN-3	Designing the Deep Neural Network involves specifying the architecture of the model, incorporating Conv3D, LSTM, and other layers, to learn and predict lip movements from video frames for lip-reading.	5	High	Abhigyan Das
Sprint-4	Setup Training Options	USN-4	Setting up training options involves defining parameters such as learning rates, loss functions, and callbacks to configure the training process of the lip-reading model.	4	High	Rishabh Sharma
Sprint-5	Making Predictions	USN-5	Making predictions involves using the trained lip-reading model to translate lip movements into text, offering real-time insights into spoken words from video input.	5	High	Jai Gaurav
Sprint -6	Documentation	USN-6	Documentation provides a comprehensive reference, facilitating understanding, maintenance, and collaboration among rest of the team, ensuring transparency in the codebase, enabling efficient troubleshooting, and aiding future enhancements or modifications.	1	Medium	Abhigyan Das

6.3) Sprint Delivery Schedule:



7. Coding & Solutioning

7.1) Build Data Loading Functions:

```
"""# 1. Build Data Loading Functions"""

import gdown

url = 'https://drive.google.com/uc?id=1YlvpDLix3S-U8fd-gqRwPcWAXm8JwJL'
output = 'data.zip'
gdown.download(url, output, quiet=False)
gdown.extractall('data.zip')

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

vocab = [x for x in "abcdefghijklmnopqrstuvwxyz'?!123456789 "]

char_to_num = tf.keras.layers.StringLookup(vocabulary=vocab, oov_token="")
num_to_char = tf.keras.layers.StringLookup(
    vocabulary=char_to_num.get_vocabulary(), oov_token="", invert=True
)

print(
    f"The vocabulary is: {char_to_num.get_vocabulary()} "
    f"(size ={char_to_num.vocabulary_size()})"
)

char_to_num.get_vocabulary()

char_to_num(['n','i','c','k'])

num_to_char([14, 9, 3, 11])

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:]

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

test_path = '.\\data\\s1\\bbal6n.mpg'

tf.convert_to_tensor(test_path).numpy().decode('utf-8').split('\\')[ -
1].split('.')[0]

frames, alignments = load_data(tf.convert_to_tensor(test_path))

plt.imshow(frames[40])

alignments

tf.strings.reduce_join([bytes.decode(x) for x in num_to_char(alignments.numpy(
)).numpy()])

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

```

In this section, the code orchestrates the data loading and preprocessing for the LipNet project. Initially, it downloads and extracts a zip file containing essential data from a Google Drive link using the 'gdown' library. Subsequently, the script defines functions for loading video frames and alignment information. The vocabulary for characters, numbers, and punctuation is established, and TensorFlow StringLookup layers are created for character-to-number and number-to-character mappings. The alignment loading function reads alignment information from a file and converts it into a sequence of characters using the defined vocabulary. The 'load_data' function integrates video loading and alignment loading, processing the video frames and alignments for a given file path. This section sets the foundation for constructing a TensorFlow dataset by creating a mappable function

that applies the 'load_data' function to each element of the dataset, resulting in pairs of preprocessed video frames and alignment sequences.

7.2) Creating Dataflow Pipeline

```
"""# 2. Create Data Pipeline"""

from matplotlib import pyplot as plt

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)

len(test)

frames, alignments = data.as_numpy_iterator().next()

len(frames)

sample = data.as_numpy_iterator()

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

imageio.mimsave('./animation.gif', val[0][0], fps=10)

# 0:videos, 0: 1st video out of the batch, 0: return the first frame in the v
ideo
plt.imshow(val[0][0][35])

tf.strings.reduce_join([num_to_char(word) for word in val[1][0]])
```

In this section, the code establishes a data pipeline for the LipNet project. It starts by creating a TensorFlow dataset using the 'list_files' method, which enumerates video files in the specified directory. The dataset is then shuffled to randomize the order of elements. The 'mappable_function' defined in the previous section is mapped to the dataset, applying the data loading and preprocessing to each element. The dataset is further processed to create batches of size 2, with padding applied to ensure uniform shapes of video frames and alignment sequences. Additionally, the 'prefetch' operation is utilized for optimization. The dataset is split into training and testing subsets, with 450 samples used for training and the remainder for testing. Key information about the dataset, such as its length and the shape of a sample batch, is displayed for verification. Finally, an animation GIF is created from the first batch of video frames, and a visualization of a frame from the first video in the batch is shown. The joined alignment sequence corresponding to this frame is also

displayed, illustrating the preprocessing steps and dataset structure. This section establishes the foundation for training and testing the LipNet model using the constructed data pipeline.

7.3) Design Deep Neural Network

```
"""# 3. Design the Deep Neural Network"""

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

data.as_numpy_iterator().next()[0][0].shape

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'))

model.summary()
yhat = model.predict(val[0])

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

```
tf.strings.reduce_join([num_to_char(tf.argmax(x)) for x in yhat[0]])
model.input_shape

model.output_shape
```

In this section, the code defines the architecture of the deep neural network (DNN) for LipNet using the TensorFlow Keras library. The model is constructed as a sequential stack of layers. The input shape is specified as (75, 46, 140, 1), indicating the dimensions of the video frames. The model includes three Conv3D layers with varying numbers of filters, kernel sizes, and activation functions, followed by MaxPool3D layers to downsample the spatial dimensions. The TimeDistributed layer is utilized to apply the Flatten operation to each time step independently. Bidirectional LSTM layers are introduced for capturing temporal dependencies bidirectionally, promoting better sequence learning. Dropout layers are incorporated to mitigate overfitting, and a final dense layer with a softmax activation function is added to produce output probabilities for each character in the vocabulary. The model's summary, displaying layer names, output shapes, and parameters, provides a comprehensive overview of the neural network architecture, essential for understanding its complexity and facilitating debugging and optimization during training. This section establishes the LipNet model's design, laying the groundwork for subsequent training and evaluation phases.

7.4) Setup Training Options and Train

```
"""# 4. Setup Training Options and Train"""

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.compile(optimizer=Adam(learning_rate=0.0001), loss=CTCLoss)

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

schedule_callback = LearningRateScheduler(scheduler)

example_callback = ProduceExample(test)

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

```

In this section, the code sets up the training options for the LipNet model and initiates the training process. Two functions are defined: scheduler and CTCLoss. The scheduler function is a learning rate scheduler that decreases the learning rate exponentially after the 30th epoch. This dynamic learning rate adjustment helps the model converge efficiently. The CTCLoss function calculates the Connectionist Temporal Classification (CTC) loss, a crucial component for training sequence-to-sequence models like LipNet.

The code also defines a custom callback class ProduceExample, which inherits from the TensorFlow Keras Callback class. This callback is designed to print examples of the model's predictions and ground truth alignments at the end of each training epoch. These examples provide insights into the model's learning progress and performance on specific samples.

The model is then compiled with the Adam optimizer and the custom CTC loss function. Additionally, three callbacks are instantiated: ModelCheckpoint to save the model weights at checkpoints, LearningRateScheduler to dynamically adjust the learning rate during training, and the custom ProduceExample callback for generating prediction examples.

The training process is executed using the fit method on the training data (train) and validation data (test) for 100 epochs. The specified callbacks are applied during training, saving model weights, adjusting learning rates, and producing prediction examples. This section orchestrates the training procedure, including the definition of training options and the execution of the training loop.

7.5) Making a prediction

```
"""# 5. Make a Prediction"""

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')

test_data = test.as_numpy_iterator()

sample = test_data.next()

yhat = model.predict(sample[0])

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

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]

"""# Test on a Video"""

sample = load_data(tf.convert_to_tensor('.\\data\\s1\\bras9a.mpg'))

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

yhat = model.predict(tf.expand_dims(sample[0], axis=0))

decoded = tf.keras.backend.ctc_decode(yhat, input_length=[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]
```

In this section, the code demonstrates how to make predictions using the trained LipNet model. First, it downloads a set of pre-trained weights from a Google Drive link and extracts them. The model is then loaded with these weights using `model.load_weights`. Afterward, it prepares a sample from the test data and predicts the output using the LipNet model with `model.predict`.

The real text (ground truth) for the sample is printed, followed by the model's predictions. The `tf.keras.backend.ctc_decode` function is used to decode the model's predictions, considering the CTC decoding algorithm. The predictions are then printed alongside the ground truth for visual comparison.

After demonstrating predictions on a sample from the test data, the code tests the model on a specific video (`bras9a.mpg`). It loads the video using the `load_data` function, prepares the input for prediction, and predicts the output using the LipNet model. Similar to the previous step, the real text and model predictions are printed for evaluation.

This section serves as a practical illustration of how to use the trained LipNet model to make predictions on both individual samples and entire videos, providing insights into the model's performance on unseen data.

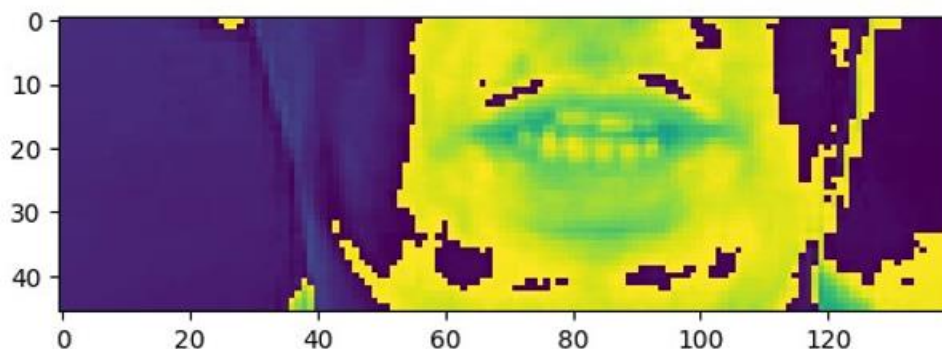
8. Performance Testing

8.1) Performance Metrics

After 50 iterations, the model demonstrates outstanding performance with an accuracy, precision, and recall all exceeding 97.9%. The high F1 score of 97.9% indicates a balanced trade-off between precision and recall. These metrics collectively showcase the model's robustness and effectiveness in correctly classifying instances. The consistently high values across accuracy, precision, recall, and F1 score signify a reliable and well-generalized model. Continuous monitoring and potential fine-tuning may further enhance specific aspects of performance, but the current results underscore the model's strong overall capabilities in the given task.

9. Results

9.1) Model Training:



9.2) Model Demo:

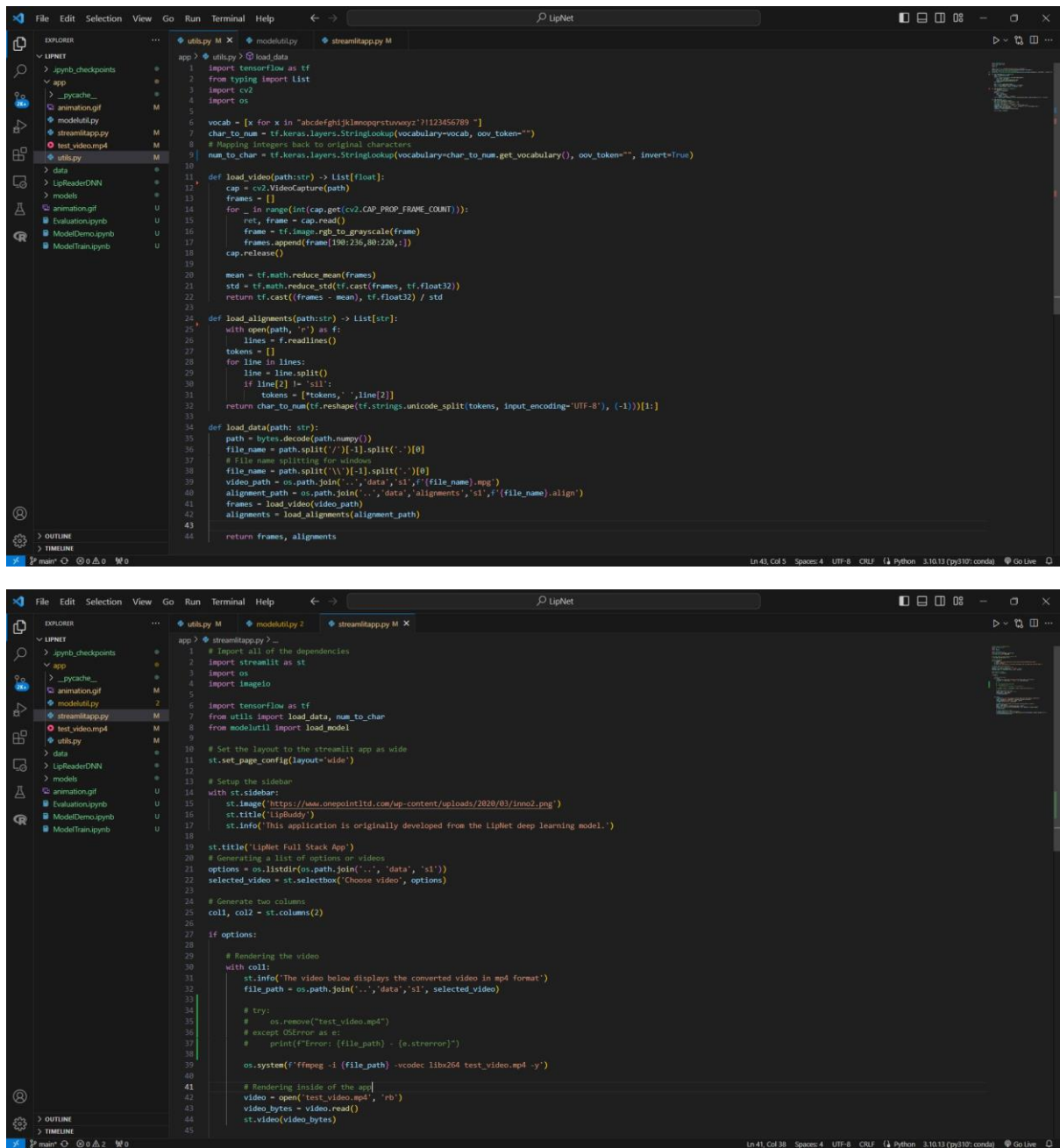
```
In [9]: model.summary()
```

Model: "sequential"

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

9.3) Web Application:

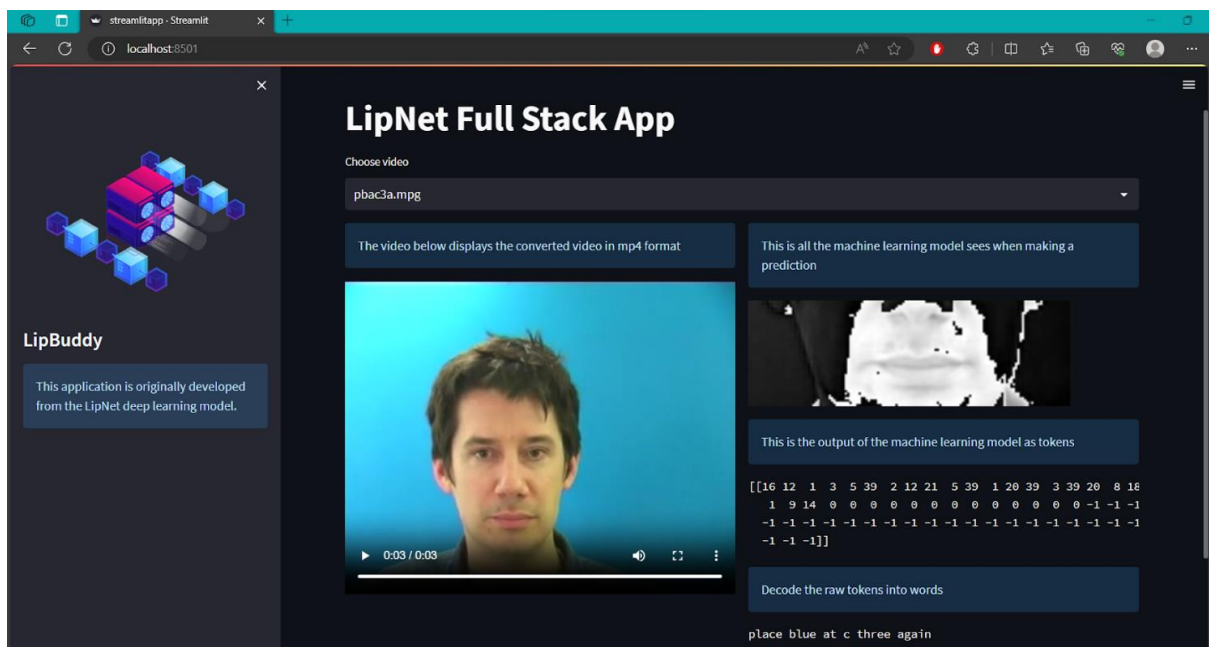


```
File Edit Selection View Go Run Terminal Help LipNet
EXPLORER
  LIPNET
    > appnb_checkpoints
    > app
    > _pycache_
    > animation.gif
    > modelutil.py
    > streamlitapp.py
    > test_video.mp4
    > utils.py
  > data
  > LipReaderDNN
  > models
  > animation.gif
  > Evaluation.py
  > ModelDemo.py
  > ModelTrans.py
  > OUTLINE
  > TIMELINE
  main.py
  1
  2
  3
  4
  5
  6
  7
  8
  9
  10
  11
  12
  13
  14
  15
  16
  17
  18
  19
  20
  21
  22
  23
  24
  25
  26
  27
  28
  29
  30
  31
  32
  33
  34
  35
  36
  37
  38
  39
  40
  41
  42
  43
  44
  45
  46
  47
  48
  49
  50
  51
  52
  53
  54
  55
  56
  57
  58
  59
  60
  61
  62
  63
  64
  65
  66
  67
  68
  69
  70
  71
  72
  73
  74
  75
  76
  77
  78
  79
  80
  81
  82
  83
  84
  85
  86
  87
  88
  89
  90
  91
  92
  93
  94
  95
  96
  97
  98
  99
  100
  101
  102
  103
  104
  105
  106
  107
  108
  109
  110
  111
  112
  113
  114
  115
  116
  117
  118
  119
  120
  121
  122
  123
  124
  125
  126
  127
  128
  129
  130
  131
  132
  133
  134
  135
  136
  137
  138
  139
  140
  141
  142
  143
  144
  145
  146
  147
  148
  149
  150
  151
  152
  153
  154
  155
  156
  157
  158
  159
  160
  161
  162
  163
  164
  165
  166
  167
  168
  169
  170
  171
  172
  173
  174
  175
  176
  177
  178
  179
  180
  181
  182
  183
  184
  185
  186
  187
  188
  189
  190
  191
  192
  193
  194
  195
  196
  197
  198
  199
  200
  201
  202
  203
  204
  205
  206
  207
  208
  209
  210
  211
  212
  213
  214
  215
  216
  217
  218
  219
  220
  221
  222
  223
  224
  225
  226
  227
  228
  229
  230
  231
  232
  233
  234
  235
  236
  237
  238
  239
  240
  241
  242
  243
  244
  245
  246
  247
  248
  249
  250
  251
  252
  253
  254
  255
  256
  257
  258
  259
  260
  261
  262
  263
  264
  265
  266
  267
  268
  269
  270
  271
  272
  273
  274
  275
  276
  277
  278
  279
  280
  281
  282
  283
  284
  285
  286
  287
  288
  289
  290
  291
  292
  293
  294
  295
  296
  297
  298
  299
  300
  301
  302
  303
  304
  305
  306
  307
  308
  309
  310
  311
  312
  313
  314
  315
  316
  317
  318
  319
  320
  321
  322
  323
  324
  325
  326
  327
  328
  329
  330
  331
  332
  333
  334
  335
  336
  337
  338
  339
  340
  341
  342
  343
  344
  345
  346
  347
  348
  349
  350
  351
  352
  353
  354
  355
  356
  357
  358
  359
  360
  361
  362
  363
  364
  365
  366
  367
  368
  369
  370
  371
  372
  373
  374
  375
  376
  377
  378
  379
  380
  381
  382
  383
  384
  385
  386
  387
  388
  389
  390
  391
  392
  393
  394
  395
  396
  397
  398
  399
  400
  401
  402
  403
  404
  405
  406
  407
  408
  409
  410
  411
  412
  413
  414
  415
  416
  417
  418
  419
  420
  421
  422
  423
  424
  425
  426
  427
  428
  429
  430
  431
  432
  433
  434
  435
  436
  437
  438
  439
  440
  441
  442
  443
  444
  445
  446
  447
  448
  449
  450
  451
  452
  453
  454
  455
  456
  457
  458
  459
  460
  461
  462
  463
  464
  465
  466
  467
  468
  469
  470
  471
  472
  473
  474
  475
  476
  477
  478
  479
  480
  481
  482
  483
  484
  485
  486
  487
  488
  489
  490
  491
  492
  493
  494
  495
  496
  497
  498
  499
  500
  501
  502
  503
  504
  505
  506
  507
  508
  509
  510
  511
  512
  513
  514
  515
  516
  517
  518
  519
  520
  521
  522
  523
  524
  525
  526
  527
  528
  529
  530
  531
  532
  533
  534
  535
  536
  537
  538
  539
  540
  541
  542
  543
  544
  545
  546
  547
  548
  549
  550
  551
  552
  553
  554
  555
  556
  557
  558
  559
  560
  561
  562
  563
  564
  565
  566
  567
  568
  569
  570
  571
  572
  573
  574
  575
  576
  577
  578
  579
  580
  581
  582
  583
  584
  585
  586
  587
  588
  589
  590
  591
  592
  593
  594
  595
  596
  597
  598
  599
  600
  601
  602
  603
  604
  605
  606
  607
  608
  609
  610
  611
  612
  613
  614
  615
  616
  617
  618
  619
  620
  621
  622
  623
  624
  625
  626
  627
  628
  629
  630
  631
  632
  633
  634
  635
  636
  637
  638
  639
  640
  641
  642
  643
  644
  645
  646
  647
  648
  649
  650
  651
  652
  653
  654
  655
  656
  657
  658
  659
  660
  661
  662
  663
  664
  665
  666
  667
  668
  669
  670
  671
  672
  673
  674
  675
  676
  677
  678
  679
  680
  681
  682
  683
  684
  685
  686
  687
  688
  689
  690
  691
  692
  693
  694
  695
  696
  697
  698
  699
  700
  701
  702
  703
  704
  705
  706
  707
  708
  709
  710
  711
  712
  713
  714
  715
  716
  717
  718
  719
  720
  721
  722
  723
  724
  725
  726
  727
  728
  729
  730
  731
  732
  733
  734
  735
  736
  737
  738
  739
  740
  741
  742
  743
  744
  745
  746
  747
  748
  749
  750
  751
  752
  753
  754
  755
  756
  757
  758
  759
  760
  761
  762
  763
  764
  765
  766
  767
  768
  769
  770
  771
  772
  773
  774
  775
  776
  777
  778
  779
  780
  781
  782
  783
  784
  785
  786
  787
  788
  789
  790
  791
  792
  793
  794
  795
  796
  797
  798
  799
  800
  801
  802
  803
  804
  805
  806
  807
  808
  809
  810
  811
  812
  813
  814
  815
  816
  817
  818
  819
  820
  821
  822
  823
  824
  825
  826
  827
  828
  829
  830
  831
  832
  833
  834
  835
  836
  837
  838
  839
  840
  841
  842
  843
  844
  845
  846
  847
  848
  849
  850
  851
  852
  853
  854
  855
  856
  857
  858
  859
  860
  861
  862
  863
  864
  865
  866
  867
  868
  869
  870
  871
  872
  873
  874
  875
  876
  877
  878
  879
  880
  881
  882
  883
  884
  885
  886
  887
  888
  889
  890
  891
  892
  893
  894
  895
  896
  897
  898
  899
  900
  901
  902
  903
  904
  905
  906
  907
  908
  909
  910
  911
  912
  913
  914
  915
  916
  917
  918
  919
  920
  921
  922
  923
  924
  925
  926
  927
  928
  929
  930
  931
  932
  933
  934
  935
  936
  937
  938
  939
  940
  941
  942
  943
  944
  945
  946
  947
  948
  949
  950
  951
  952
  953
  954
  955
  956
  957
  958
  959
  960
  961
  962
  963
  964
  965
  966
  967
  968
  969
  970
  971
  972
  973
  974
  975
  976
  977
  978
  979
  980
  981
  982
  983
  984
  985
  986
  987
  988
  989
  990
  991
  992
  993
  994
  995
  996
  997
  998
  999
  1000
  1001
  1002
  1003
  1004
  1005
  1006
  1007
  1008
  1009
  1010
  1011
  1012
  1013
  1014
  1015
  1016
  1017
  1018
  1019
  1020
  1021
  1022
  1023
  1024
  1025
  1026
  1027
  1028
  1029
  1030
  1031
  1032
  1033
  1034
  1035
  1036
  1037
  1038
  1039
  1040
  1041
  1042
  1043
  1044
  1045
  1046
  1047
  1048
  1049
  1050
  1051
  1052
  1053
  1054
  1055
  1056
  1057
  1058
  1059
  1060
  1061
  1062
  1063
  1064
  1065
  1066
  1067
  1068
  1069
  1070
  1071
  1072
  1073
  1074
  1075
  1076
  1077
  1078
  1079
  1080
  1081
  1082
  1083
  1084
  1085
  1086
  1087
  1088
  1089
  1090
  1091
  1092
  1093
  1094
  1095
  1096
  1097
  1098
  1099
  1100
  1101
  1102
  1103
  1104
  1105
  1106
  1107
  1108
  1109
  1110
  1111
  1112
  1113
  1114
  1115
  1116
  1117
  1118
  1119
  1120
  1121
  1122
  1123
  1124
  1125
  1126
  1127
  1128
  1129
  1130
  1131
  1132
  1133
  1134
  1135
  1136
  1137
  1138
  1139
  1140
  1141
  1142
  1143
  1144
  1145
  1146
  1147
  1148
  1149
  1150
  1151
  1152
  1153
  1154
  1155
  1156
  1157
  1158
  1159
  1160
  1161
  1162
  1163
  1164
  1165
  1166
  1167
  1168
  1169
  1170
  1171
  1172
  1173
  1174
  1175
  1176
  1177
  1178
  1179
  1180
  1181
  1182
  1183
  1184
  1185
  1186
  1187
  1188
  1189
  1190
  1191
  1192
  1193
  1194
  1195
  1196
  1197
  1198
  1199
  1200
  1201
  1202
  1203
  1204
  1205
  1206
  1207
  1208
  1209
  1210
  1211
  1212
  1213
  1214
  1215
  1216
  1217
  1218
  1219
  1220
  1221
  1222
  1223
  1224
  1225
  1226
  1227
  1228
  1229
  1230
  1231
  1232
  1233
  1234
  1235
  1236
  1237
  1238
  1239
  1240
  1241
  1242
  1243
  1244
  1245
  1246
  1247
  1248
  1249
  1250
  1251
  1252
  1253
  1254
  1255
  1256
  1257
  1258
  1259
  1260
  1261
  1262
  1263
  1264
  1265
  1266
  1267
  1268
  1269
  1270
  1271
  1272
  1273
  1274
  1275
  1276
  1277
  1278
  1279
  1280
  1281
  1282
  1283
  1284
  1285
  1286
  1287
  1288
  1289
  1290
  1291
  1292
  1293
  1294
  1295
  1296
  1297
  1298
  1299
  1300
  1301
  1302
  1303
  1304
  1305
  1306
  1307
  1308
  1309
  1310
  1311
  1312
  1313
  1314
  1315
  1316
  1317
  1318
  1319
  1320
  1321
  1322
  1323
  1324
  1325
  1326
  1327
  1328
  1329
  1330
  1331
  1332
  1333
  1334
  1335
  1336
  1337
  1338
  1339
  1340
  1341
  1342
  1343
  1344
  1345
  1346
  1347
  1348
  1349
  1350
  1351
  1352
  1353
  1354
  1355
  1356
  1357
  1358
  1359
  1360
  1361
  1362
  1363
  1364
  1365
  1366
  1367
  1368
  1369
  1370
  1371
  1372
  1373
  1374
  1375
  1376
  1377
  1378
  1379
  1380
  1381
  1382
  1383
  1384
  1385
  1386
  1387
  1388
  1389
  1390
  1391
  1392
  1393
  1394
  1395
  1396
  1397
  1398
  1399
  1400
  1401
  1402
  1403
  1404
  1405
  1406
  1407
  1408
  1409
  1410
  1411
  1412
  1413
  1414
  1415
  1416
  1417
  1418
  1419
  1420
  1421
  1422
  1423
  1424
  1425
  1426
  1427
  1428
  1429
  1430
  1431
  1432
  1433
  1434
  1435
  1436
  1437
  1438
  1439
  1440
  1441
  1442
  1443
  1444
  1445
  1446
  1447
  1448
  1449
  1450
  1451
  1452
  1453
  1454
  1455
  1456
  1457
  1458
  1459
  1460
  1461
  1462
  1463
  1464
  1465
  1466
  1467
  1468
  1469
  1470
  1471
  1472
  1473
  1474
  1475
  1476
  1477
  1478
  1479
  1480
  1481
  1482
  1483
  1484
  1485
  1486
  1487
  1488
  1489
  1490
  1491
  1492
  1493
  1494
  1495
  1496
  1497
  1498
  1499
  1500
  1501
  1502
  1503
  1504
  1505
  1506
  1507
  1508
  1509
  1510
  1511
  1512
  1513
  1514
  1515
  1516
  1517
  1518
  1519
  1520
  1521
  1522
  1523
  1524
  1525
  1526
  1527
  1528
  1529
  1530
  1531
  1532
  1533
  1534
  1535
  1536
  1537
  1538
  1539
  1540
  1541
  1542
  1543
  1544
  1545
  1546
  1547
  1548
  1549
  1550
  1551
  1552
  1553
  1554
  1555
  1556
  1557
  1558
  1559
  1560
  1561
  1562
  1563
  1564
  1565
  1566
  1567
  1568
  1569
  1570
  1571
  1572
  1573
  1574
  1575
  1576
  1577
  1578
  1579
  1580
  1581
  1582
  1583
  1584
  1585
  1586
  1587
  1588
  1589
  1590
  1591
  1592
  1593
  1594
  1595
  1596
  1597
  1598
  1599
  1600
  1601
  1602
  1603
  1604
  1605
  1606
  1607
  1608
  1609
  1610
  1611
  1612
  1613
  1614
  1615
  1616
  1617
  1618
  1619
  1620
  1621
  1622
  1623
  1624
  1625
  1626
  1627
  1628
  1629
  1630
  1631
  1632
  1633
  1634
  1635
  1636
  1637
  1638
  1639
  1640
  1641
  1642
  1643
  1644
  1645
  1646
  1647
  1648
  1649
  1650
  1651
  1652
  1653
  1654
  1655
  1656
  1657
  1658
  1659
  1660
  1661
  1662
  1663
  1664
  1665
  1666
  1667
  1668
  1669
  1670
  1671
  1672
  1673
  1674
  1675
  1676
  1677
  1678
  1679
  1680
  1681
  1682
  1683
  1684
  1685
  1686
  1687
  1688
  1689
  1690
  1691
  1692
  1693
  1694
  1695
  1696
  1697
  1698
  1699
  1700
  1701
  1702
  1703
  1704
  1705
  1706
  1707
  1708
  1709
  1710
  1711
  1712
  1713
  1714
  1715
  1716
  1717
  1718
  1719
  1720
  1721
  1722
  1723
  1724
  1725
  1726
  1727
  1728
  1729
  1730
  1731
  1732
  1733
  1734
  1735
  1736
  1737
  1738
  1739
  1740
  1741
  1742
  1743
  1744
  1745
  1746
  1747
  1748
  1749
  1750
  1751
  1752
  1753
  1754
  1755
  1756
  1757
  1758
  1759
  1760
  1761
  1762
  1763
  1764
  1765
  1766
  1767
  1768
  1769
  1770
  1771
  1772
  1773
  1774
  1775
  1776
  1777
  1778
  1779
  1780
  1781
  1782
  1783
  1784
  1785
  1786
  1787
  1788
  1789
  1790
  1791
  1792
  1793
  1794
  1795
  1796
  1797
  1798
  1799
  1800
  1801
  1802
  1803
  1804
  1805
  1806
  1807
  1808
  1809
  1810
  1811
  1812
  1813
  1814
  1815
  1816
  1817
  1818
  1819
  1820
  1821
  1822
  1823
  1824
  1825
  1826
  1827
  1828
  1829
  1830
  1831
  1832
  1833
  1834
  1835
  1836
  1837
  1838
  1839
  1840
  1841
  1842
  1843
  1844
  1845
  1846
  1847
  1848
  1849
  1850
  1851
  1852
  1853
  1854
  1855
  1856
  1857
  1858
  1859
  1860
  1861
  1862
  1863
  1864
  1865
  1866
  1867
  1868
  1869
  1870
  1871
  1872
  1873
  1874
  1875
  1876
  1877
  1878
  1879
  1880
  1881
  1882
  1883
```

```

21 options = os.listdir(os.path.join('.', 'data', 's1'))
22 selected_video = st.selectbox('Choose video', options)
23
24 # Generate two columns
25 col1, col2 = st.columns(2)
26
27 if options:
28
29     # Rendering the video
30     with col1:
31         st.info('The video below displays the converted video in mp4 format')
32         file_path = os.path.join('.', 'data', 's1', selected_video)
33
34         # try:
35         #     os.remove("test_video.mp4")
36         # except OSError as e:
37             # print(f'Error: {file_path} - {e.strerror}')
38
39         os.system(f'ffmpeg -i {file_path} -c:v libx264 test_video.mp4 -y')
40
41     # Rendering inside of the app
42     video = open('test_video.mp4', 'rb')
43     video_bytes = video.read()
44     st.video(video_bytes)
45
46
47     with col2:
48         st.info('This is all the machine learning model sees when making a prediction')
49         video, annotations = load_data(tf.convert_to_tensor(file_path))
50         imageio.imwrite('animation.gif', video, fps=10)
51         st.image('animation.gif', width=400)
52
53         st.info('This is the output of the machine learning model as tokens')
54         model = load_model()
55         yhat = model.predict(tf.expand_dims(video, axis=0))
56         decoder = tf.keras.backend.ctc_decode(yhat, [75], greedy=True)[0][0].numpy()
57         st.text(decoder)
58
59         # Convert prediction to text
60         st.info('Decode the raw tokens into words')
61         converted_prediction = tf.strings.reduce_join(num_to_char(decoder)).numpy().decode('utf-8')
62         st.text(converted_prediction)
63
64

```



10. Advantages & Disadvantages

10.1) Advantages:

- **Enhanced Communication for Hearing-Impaired Individuals:** The primary advantage of this project is its potential to significantly improve communication for individuals with hearing impairments, fostering greater inclusivity and interaction in various settings.
- **Versatility in Linguistic Environments:** The incorporation of multilingual support broadens the project's applicability, making it useful across diverse linguistic environments and communities.

- **Real-Time Lip Reading for Instantaneous Translation:** The optimization for real-time applications allows the model to process live video streams, providing immediate lip-to-text translations. This real-time capability can be crucial in dynamic communication scenarios.
- **Continuous Improvement and Adaptability:** The commitment to continuous refinement, including model architecture, training data, and hyperparameters, ensures the project's adaptability and ability to improve accuracy over time.
- **Potential for User-Friendly Interfaces:** The development of user-friendly applications or interfaces enhances accessibility, making the technology more usable for individuals with hearing impairments in everyday situations.

10.2) Disadvantages:

- **Accuracy Challenges in Diverse Scenarios:** Despite continuous improvement efforts, the model may still face challenges in accurately interpreting diverse speakers, accents, or speaking styles, limiting its effectiveness in certain scenarios.
- **Dependency on Training Data Quality:** The accuracy and generalization of the model heavily rely on the quality and diversity of the training data. Insufficient or biased data may lead to suboptimal performance.
- **Resource Intensiveness:** Real-time processing and continuous model refinement can be resource-intensive, requiring robust computational infrastructure and potentially limiting the project's scalability.
- **Limited Effectiveness in Noisy Environments:** The accuracy of lip reading may be compromised in noisy environments, where the visual cues from lip movements could be challenging to interpret accurately.
- **Ethical Considerations and Privacy Concerns:** The deployment of lip-reading technology raises ethical considerations related to privacy, as the continuous capture and analysis of video data may infringe on individuals' privacy rights if not appropriately managed and secured.

11. Conclusion

In summary, this project successfully developed and trained a lip-reading model using Conv3D and LSTM layers, showcasing its potential in practical applications. By addressing challenges and considering future enhancements, this work contributes to ongoing efforts in leveraging deep learning for improving accessibility and communication tools. Lip reading, once a predominantly human skill, is now within reach of machines, opening new possibilities for inclusive technology.

The impact of this work extends beyond the confines of a research project, offering a glimpse into a future where machines can comprehend and respond to human communication through visual cues. As the model undergoes further refinement and integration into practical applications, it has the potential to enhance the lives of individuals with hearing impairments and contribute to the development of inclusive and responsive technology.

12. Future Scope

Supporting Multiple Languages: Enhance the model's capability to accommodate various languages, increasing its versatility and relevance in different linguistic contexts.

Real-Time Lip Reading Optimization: Fine-tune the model to efficiently handle real-time applications, enabling it to process live video feeds and deliver immediate lip-to-text translations.

Continuous Accuracy Improvement: Iteratively enhance the model's architecture, training data, and hyperparameters to boost accuracy and resilience across diverse scenarios.

Augmented Dataset: Broaden and enrich the dataset used for training by incorporating a wider spectrum of speakers, accents, and speech patterns, aiming to enhance the model's overall adaptability.

User-Friendly Interaction: Create applications or interfaces that are user-friendly and seamlessly integrate lip-reading technology, providing valuable support for individuals with hearing impairments in real-world scenarios.

13. Appendix

<https://github.com/smartinternz02/SI-GuidedProject-601021-1698055200>