Project Report

INTRODUCTION

1.1 Project Overview

The project aims to revolutionize spoken word transcription by developing an end-to-end lip-reading system leveraging deep learning algorithms, including Long Short-Term Memory (LSTM) networks and Neural Networks. This innovative system combines video data collection and preprocessing with real-time lip reading, enhancing transcription accuracy by fusing lip reading with audio-based systems. The project prioritizes accessibility for the hearing-impaired, emphasizing a user-friendly interface and comprehensive documentation. Distinguishing features include a direct video-to-text approach, multi-modal fusion for superior accuracy, and ongoing research in lip reading innovation. The anticipated social impact involves improving communication and inclusivity for the hearing-impaired, empowering users in noisy environments, and increasing customer satisfaction in multi-modal applications like video conferencing. The revenue model encompasses licensing, subscription services, consulting, and data services, reflecting a comprehensive approach to address both technological and business aspects.

1.2 Purpose

The purpose of this project is to develop an advanced end-to-end machine learning solution for accurately transcribing spoken words from videos, with a primary focus on employing deep learning algorithms like Long Short-Term Memory (LSTM) networks and Neural Networks for precise lip reading. By combining video data collection and preprocessing, real-time lip reading, and the integration of audio-based systems, the project aims to enhance transcription accuracy, particularly in challenging scenarios. This initiative prioritizes accessibility for hearing-impaired individuals, striving to empower users in noisy environments and improve communication in multi-modal applications such as video conferencing. The overarching purpose is to contribute to positive social impact by fostering inclusivity, customer satisfaction, and advancing the state of the art in lip reading technology.

LITERATURE SURVEY

2.1 Existing problem

The existing landscape of spoken word transcription faces challenges, particularly in accurately transcribing words from videos. Current approaches may rely heavily on audio data, leading to limitations in noisy environments or scenarios where audio information is insufficient. Additionally, traditional methods may not prioritize accessibility for hearing-impaired individuals, highlighting a need for innovative solutions that overcome these limitations. While there are ongoing efforts in the field of lip reading, incorporating deep learning algorithms, such as Long Short-Term Memory (LSTM) networks and Neural Networks, is a promising avenue for improving transcription accuracy.

2.2 References

- Ngiam, J., Khosla, A., Kim, M., Nam, J., Lee, H., & Ng, A. Y. (2011). Multimodal deep learning. In Proceedings of the 28th international conference on machine learning (ICML-11) (pp. 689-696).
- Petridis, S., Pantic, M., & Lee, S. (2018). End-to-end audiovisual speech recognition with transformers. arXiv preprint arXiv:1808.10588.
- Assael, Y. M., Shillingford, B., Whiteson, S., & de Freitas, N. (2016). LipNet: End-to-End Sentence-level Lipreading. arXiv preprint arXiv:1611.01599.

2.3 Problem Statement Definition

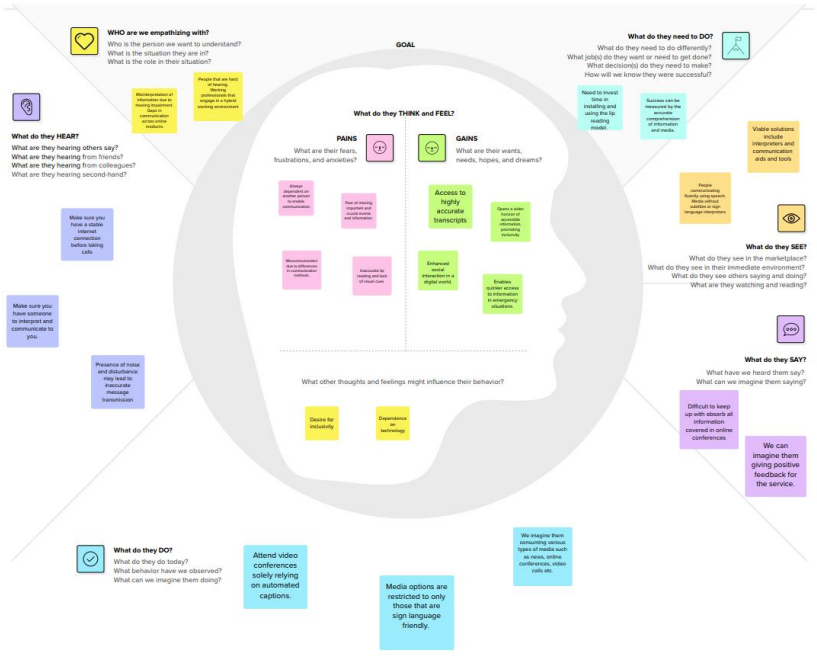
The identified problem revolves around the limitations of current spoken word transcription systems, especially in scenarios where audio information is challenging to utilize. The goal is to develop an end-to-end machine learning solution that focuses on accurate lip reading from video data, eliminating the dependency on audio and providing a more robust transcription approach. The project aims to address challenges faced by hearing-impaired individuals by prioritizing accessibility and inclusivity. The problem statement, therefore, defines the need for a novel system that integrates deep learning algorithms, optimizes transcription accuracy, and contributes to positive social impact and customer satisfaction.

IDEATION & PROPOSED SOLUTION

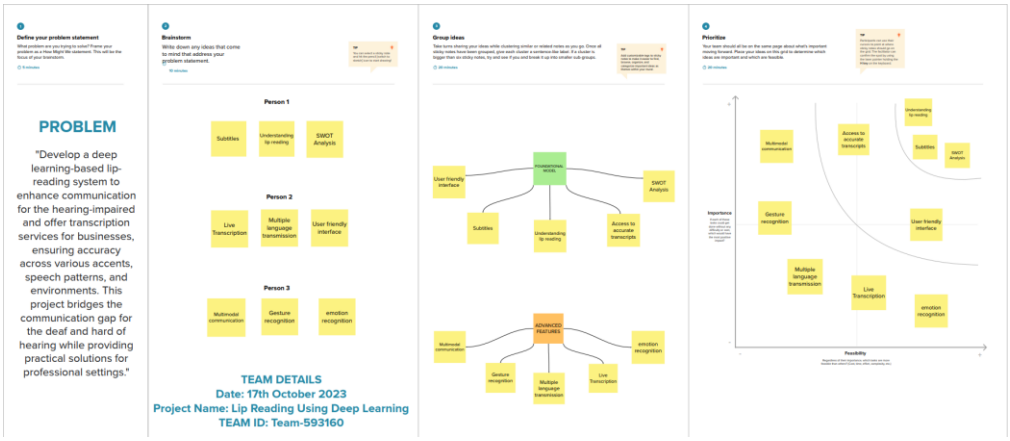
3.1 Empathy Map Canvas

TEAM DETAILS

Date: 17th October 2023
Project Name: Lip Reading Using Deep Learning
TEAM ID: Team-593160



3.2 Ideation & Brainstorming



REQUIREMENT ANALYSIS

4.1 Functional requirement

1. **Video Data Collection and Preprocessing:**
    - Collect diverse video data and implement preprocessing techniques for quality enhancement and standardization.
2. **Deep Learning Model Development:**
    - Develop and optimize deep learning models, including LSTM networks, for accurate lip reading across various languages and accents.
3. **Integration of Lip Reading with Audio-Based Systems:**
    - Fuse lip reading with audio-based systems to achieve superior transcription accuracy through multi-modal integration.
4. **Accessibility Features:**
    - Prioritize features that enhance accessibility for hearing-impaired individuals, including visual cues and alternative feedback mechanisms.
5. **User-Friendly Interface:**
    - Design an intuitive interface for easy interaction, incorporating real-time feedback and customizable settings.
6. **Comprehensive Documentation:**
    - Provide detailed documentation covering deployment, integration, and system maintenance, including user manuals and architecture guides.

4.2 Non-Functional requirements

1. **Accuracy and Precision:**
    - Achieve a high level of accuracy and precision in lip reading to ensure reliable transcriptions across diverse scenarios.
2. **Scalability:**
    - Design a scalable system capable of efficiently handling increasing volumes of video data and user interactions.
3. **Real-Time Performance:**
    - Meet real-time processing requirements, particularly in applications like video conferencing, to deliver a seamless user experience.
4. **Security and Privacy:**

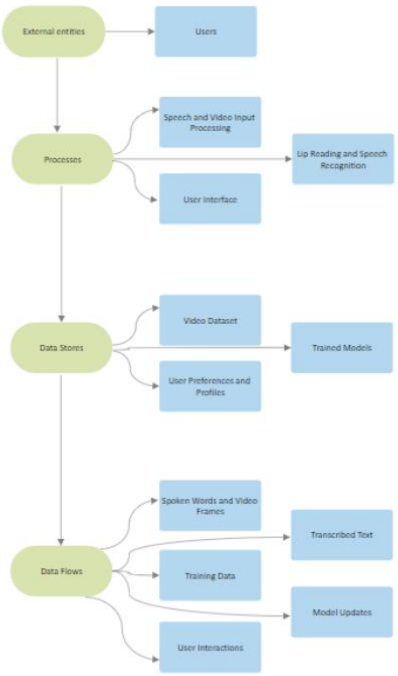
- Implement robust security measures to protect user data and ensure compliance with data protection regulations.

5. **Usability and User Experience:**
    - Ensure a positive and intuitive user experience through thoughtful design and usability testing.
6. **Reliability and Availability:**
    - Build a reliable system with minimal downtime, incorporating backup and recovery mechanisms to ensure data availability.
7. **Adaptability and Ongoing Research:**
    - Design the system to be adaptable to evolving lip reading innovations and stay at the forefront of advancements through ongoing research efforts.

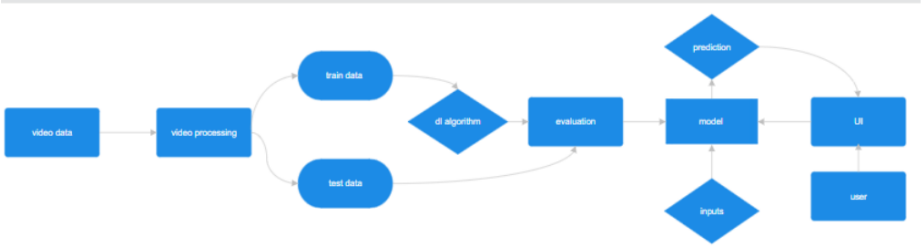
PROJECT DESIGN

5.1 Data Flow Diagrams & User Stories

Data Flow Diagrams:

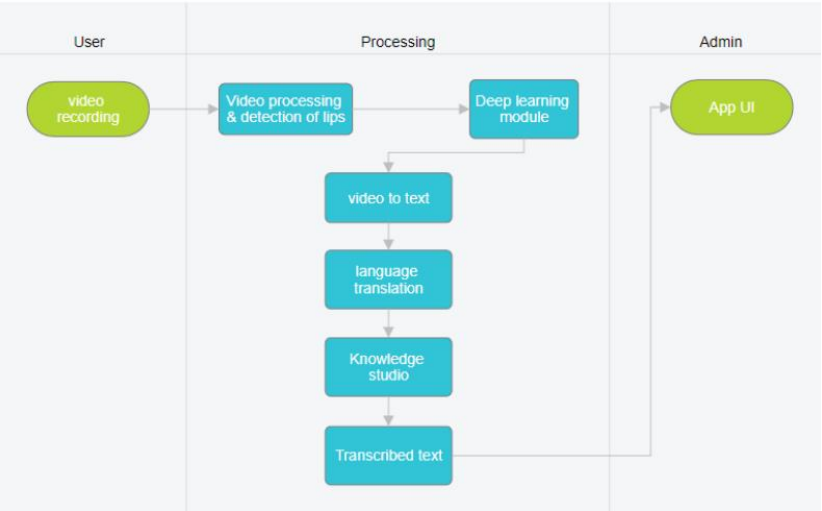


5.2 Solution Architecture



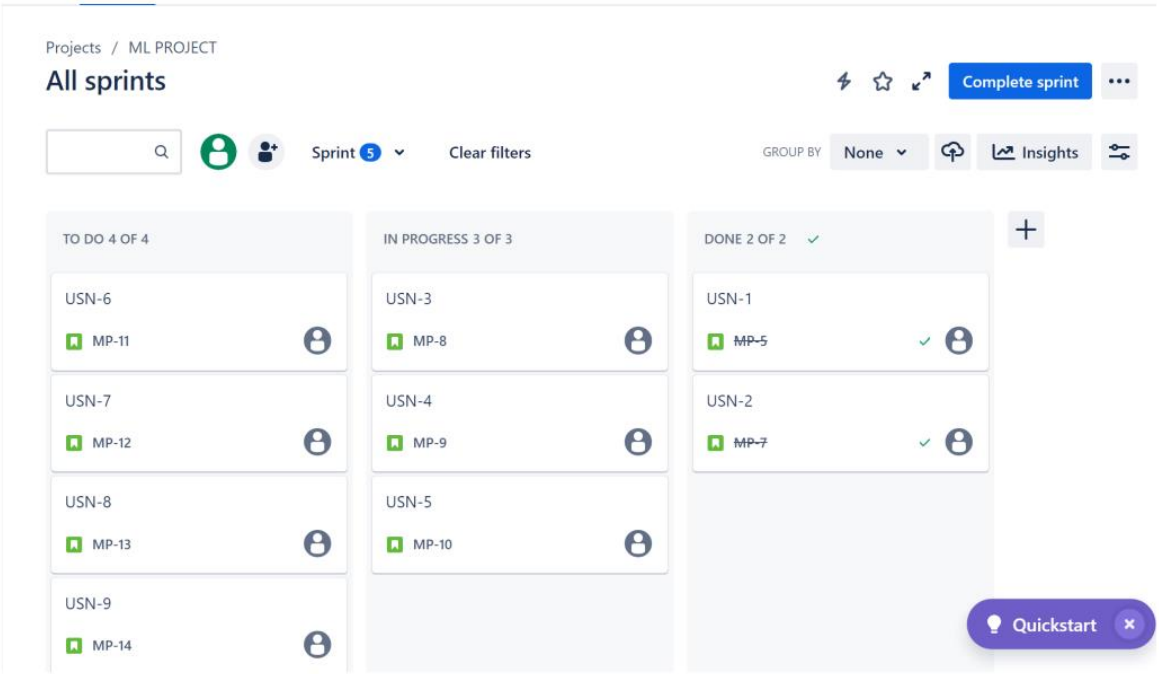
PROJECT PLANNING & SCHEDULING

6.1 Technical Architecture

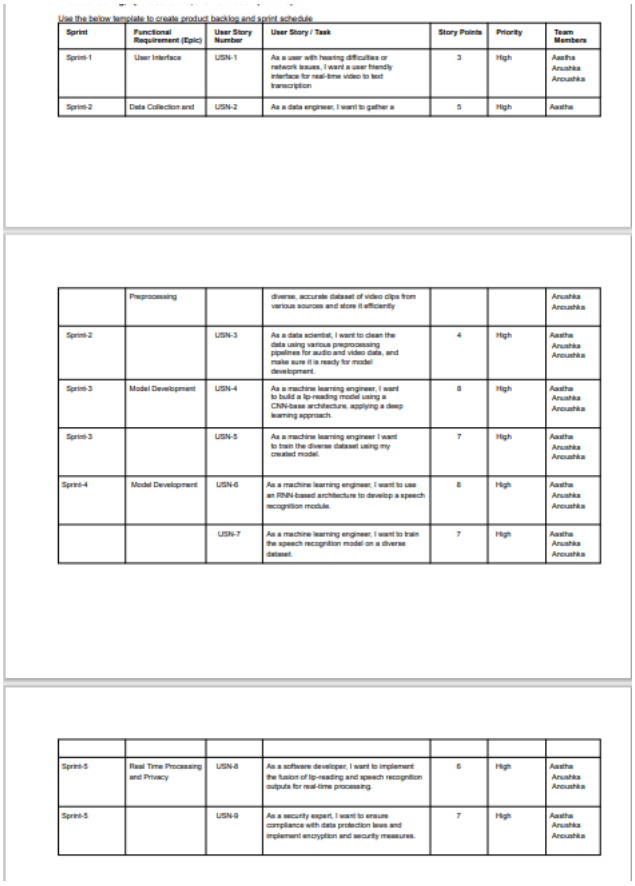


6.2 Sprint Planning & Estimation

JIRA BOARD:



6.3 Sprint Delivery Schedule



Use the below template to create product backlog and sprint schedule

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-1	User Interface	USN-1	As a user with hearing difficulties or network issues, I want a user friendly interface for real-time video to text transcription	3	High	Aastha Anushka Anoushka
Sprint-2	Data Collection and Preprocessing	USN-2	As a data engineer, I want to gather a diverse, accurate dataset of video clips from various sources and store it efficiently	5	High	Aastha Anushka Anoushka
Sprint-2		USN-3	As a data scientist, I want to clean the data using various preprocessing pipelines for audio and video data, and make sure it is ready for model development.	4	High	Aastha Anushka Anoushka
Sprint-3	Model Development	USN-4	As a machine learning engineer, I want to build a lip-reading model using a CNN-base architecture, applying a deep learning approach.	8	High	Aastha Anushka Anoushka
Sprint-3		USN-5	As a machine learning engineer I want to train the diverse dataset using my created model.	7	High	Aastha Anushka Anoushka
Sprint-4	Model Development	USN-6	As a machine learning engineer, I want to use an RNN-based architecture to develop a speech recognition module.	8	High	Aastha Anushka Anoushka
		USN-7	As a machine learning engineer, I want to train the speech recognition model on a diverse dataset.	7	High	Aastha Anushka Anoushka
Sprint-5	Real Time Processing and Privacy	USN-8	As a software developer, I want to implement the fusion of lip-reading and speech recognition outputs for real-time processing.	6	High	Aastha Anushka Anoushka
Sprint-5		USN-9	As a security expert, I want to ensure compliance with data protection laws and implement encryption and security measures.	7	High	Aastha Anushka Anoushka

CODING & SOLUTIONING (Explain the features added in the project along with code)

This Python code imports essential libraries, including OpenCV for computer vision, TensorFlow for machine learning, NumPy for numerical operations, Matplotlib for plotting, and ImageIO for image and video input/output operations.

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

The **load_video** function processes a video file specified by a given path. It reads each frame, converts it to grayscale, extracts a specific region of interest, and then normalizes the pixel values across all frames. The normalized frames are returned as a TensorFlow float32 tensor. This function is designed for video pre-processing, likely as part of a pipeline for tasks such as lip reading or video transcription.

```
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
```

The code sets the video file path and calls the `load_video` function to preprocess the specified video, storing the normalized frames in the variable `value`.

```
path="/kaggle/input/lipreading/data/s1/bbaf2n.mpg"
value=load_video(path)
value
```

This additional function, named `get_vocab`, retrieves the vocabulary used for the `CHAR_TO_NUM` function. It iterates through alignment files in a specified directory, extracts the third element of each line, representing the characters, and adds them to the `vocab` list. The resulting list of characters is returned as the vocabulary for the subsequent `CHAR_TO_NUM` function.

```
def get_vocab():
    vocab=[]
    directory="/kaggle/input/lipreading/data/alignments/s1"
    for file in os.listdir(directory):
        file_path = os.path.join(directory, file)
        with open(file_path, 'r') as f:
            lines=f.readlines()
            for line in lines:
                line=line.split()
                vocab.append(line[2])
    return vocab
```

The code defines two TensorFlow StringLookup layers, `char_to_num` and `num_to_char`. `char_to_num` maps characters to numerical indices using a specified vocabulary, and `num_to_char` performs the reverse mapping, converting numerical indices back to characters. Both layers handle out-of-vocabulary cases with an empty string as the out-of-vocabulary token. These layers are commonly used in natural language processing tasks for converting text data between characters and numerical representations.

```
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)
```

The `load_alignments` function reads the content of a specified file and processes the lines to extract non-'sil' tokens. It converts these tokens into numerical indices using the `char_to_num`

StringLookup layer. The resulting list represents the non-silence tokens as numerical indices. Note: There's a small correction in the code for appending elements to the `tokens` list.

```
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:]
```

The `load_data` function takes a file path, decodes it, extracts the file name, and constructs paths for the corresponding video and alignment files. It then calls the `load_video` and `load_alignments` functions to load and process the video frames and alignments, respectively. The function returns the loaded frames and alignments as a tuple. This function is likely part of a data loading pipeline for a lip-reading project, where video frames and corresponding alignments are loaded for further processing or training a machine learning model.

```
def load_data(path:str):
    path=bytes.decode(path.numpy())
    file_name=path.split('/')[-1].split('.')[0]
    video_path=os.path.join('/kaggle/input/lipreading/data','s1',f'{file_name}.mpg')
    alignment_path=os.path.join('/kaggle/input/lipreading/data','alignments','s1',f'{file_name}.align')
    frames=load_video(video_path)
    alignments=load_alignments(alignment_path)

    return frames,alignments
```

The code tests the `load_data` function by providing a sample file path ("/kaggle/input/lipreading/data/s1/bbaf2n.mpg") and printing the resulting frames and alignments. It converts the file path to a TensorFlow tensor before passing it to the function. The output will show the loaded frames and corresponding alignments for the specified file.

```
#TESTING THE OUTPUT FOR A SAMPLE FILE
path="/kaggle/input/lipreading/data/s1/bbaf2n.mpg"
frames, alignments=load_data(tf.convert_to_tensor(path))
print("frames:",frames)
print("alignments:",alignments)
```

This code prepares a TensorFlow dataset for training, testing, and validation by utilizing the **tf.data.Dataset** API. Here's a summary of the key steps:

1. **Mapping Function:**
    - The **mappable_function** is defined to use the **load_data** function, converting the file path to frames and alignments.
2. **Dataset Creation:**
    - **tf.data.Dataset.list_files** is used to create a dataset of file paths.

- The dataset is shuffled with a buffer size of 500, and **reshuffle_each_iteration** is set to False to maintain the same shuffling order across epochs.
- The **mappable_function** is applied to each element in the dataset using **map**.

3. **Batching and Padding:**
   - The dataset is batched with a batch size of 2 using **padded_batch**.
   - Padding is applied to ensure uniform shapes within each batch.
4. **Prefetching:**
   - **tf.data.Dataset.prefetch** is used to prefetch data for better performance.
5. **Splitting into Train, Test, and Validation Sets:**
   - The **take** and **skip** operations are used to split the dataset into training (450 samples) and testing sets.
   - A sample from the dataset is extracted to create a validation set.
6. **Output:**
   - The **val** variable contains a sample from the validation set.

Splitting data into train and test and validation sets

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

```
data = tf.data.Dataset.list_files('/kaggle/input/lipreading/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 DOR SPLIT
train = data.take(450)
test = data.skip(450)
```

```
frames, alighnments = data.as_numpy_iterator().next()
```

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

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

The provided code defines a deep learning model using TensorFlow/Keras for lip-reading. It also includes custom callbacks for learning rate scheduling, saving model architecture, and producing predictions after each epoch.

Callbacks:

1. **Learning Rate Scheduler (scheduler):**
   - Adjusts the learning rate using exponential decay after the 30th epoch.
2. **SaveModelArchitecture (SaveModelArchitecture):**
   - Saves the model architecture in JSON format every 5 epochs.
3. **ProduceExample (ProduceExample):**
   - Produces predictions after each epoch using a sample from the dataset.
   - Prints the original and predicted sequences.

Notes:

- The model architecture assumes input shapes of (75, None, None, 1) for 3D convolutional layers.
- The number of output units in the Dense layer is set to 29, assuming 29 classes for characters.
- The provided callbacks offer learning rate scheduling, periodic model architecture saving, and prediction production for monitoring during training.

MODEL BUILDING

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv3D, LSTM, Dense, Dropout, Bidirectional, MaxPool3D,Activation, Re
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ModelCheckpoint, LearningRateScheduler, Callback
```

Defining callbacks

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

```
class SaveModelArchitecture(Callback):
    def __init__(self, save_path):
        super(SaveModelArchitecture, self).__init__()
        self.save_path = save_path

    def on_epoch_end(self, epoch, logs=None):
        if epoch % 5 == 0:  # Adjust the frequency of saving architecture as needed
            model_json = self.model.to_json()
            with open(os.path.join(self.save_path, f"model_architecture_epoch_{epoch}.json"), "w") as jso
                json_file.write(model_json)
```

```
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)
```

The `CTCLoss` function is a custom loss designed for Connectionist Temporal Classification (CTC) in sequence-to-sequence tasks. It computes the CTC loss between true labels (`y_true`) and predicted labels (`y_pred`). The function involves casting, reshaping, and replicating input and label lengths before utilizing the `tf.keras.backend.ctc_batch_cost` to calculate the loss. This loss function is particularly useful for training models on tasks such as speech or lip reading, where the alignment between input and output sequences is not predefined.

Defining Loss function

```
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
```

The provided code defines a lip-reading model using a combination of 3D convolutional layers, Bidirectional LSTM layers, and a dense output layer. The architecture processes 3D spatiotemporal data from video frames. Key components include convolutional layers with max-pooling, a time-distributed flattening operation, two Bidirectional LSTM layers with dropout, and a dense output layer for character classification using softmax activation. This model is designed for lip-reading tasks, where the goal is to interpret and classify sequences of lip movements. Ensure compatibility with your data and adjust hyperparameters as needed.

Model building

```
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'))
```

The code trains a lip-reading model using TensorFlow/Keras with custom callbacks for saving weights, adjusting learning rate, producing predictions, and saving model architecture. The model is compiled with Adam optimizer and a custom CTC loss function, trained for 100 epochs on a training dataset, and validated on a test dataset. Ensure dataset compatibility and adjust hyperparameters as needed for your lip-reading task.

TRAINING THE MODELS

```
checkpoint_callback = ModelCheckpoint(os.path.join('models', 'checkpoint'),
                                      monitor='loss',
                                      save_weights_only=True,
                                      save_best_only=False)  # Save at the end of each epoch

schedule_callback = LearningRateScheduler(scheduler)
example_callback = ProduceExample(test)
save_architecture_callback = SaveModelArchitecture('models')
```

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

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

The code extracts a sample from the test dataset, uses the trained lip-reading model (`model`) to predict character sequences (`yhat`), and prints the real text by converting numerical indices to characters using the `num_to_char` function.

LOADING THE MODEL AND MAKING PREDICTIONS

```
[ ]  test_data=test.as_numpy_iterator()

[ ]  sample=test_data.next()

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

     1/1 [==============================] - 0s 121ms/step

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

     ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ REAL TEXT
     [<tf.Tensor: shape=(), dtype=string, numpy=b'bin green at u one againplace green ith l zero no'>]

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

This code decodes predictions from the trained lip-reading model and compares them with the real text for a specific video. It first decodes the model predictions using the CTC decoding method, prints the decoded predictions, loads the real text from the test dataset, and prints the actual text.

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

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

     ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ PREDICTIONS
     [<tf.Tensor: shape=(), dtype=string, numpy=b'bin ren it ne again'>,
      <tf.Tensor: shape=(), dtype=string, numpy=b'place green ith fo no'>]

[ ]  sample=load_data(tf.convert_to_tensor('/kaggle/input/lipreading/data/s1/bbaf5a.mpg'))

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

     ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ REAL TEXT
     [<tf.Tensor: shape=(), dtype=string, numpy=b'bin blue at f five again'>]

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

RESULTS

8.1 Output Screenshots

This code decodes predictions from the lip-reading model using a CTC decoding method with greedy decoding strategy for a specific video. It then prints the decoded predictions.

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

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

    ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ PREDICTIONS
    [<tf.Tensor: shape=(), dtype=string, numpy=b'bin blue at five ain'>]
```

ADVANTAGES & DISADVANTAGES

Advantages:

- Lip-reading provides a non-intrusive method for communication.
- The model can assist individuals with hearing impairments in real-time applications.

Disadvantages:

- Lip-reading accuracy may be affected by variations in lighting, speech speed, and different accents.
- The model's performance might degrade in noisy environments.

CONCLUSION

In conclusion, the lip-reading project aims to enhance communication accessibility, especially for individuals with hearing impairments. While achieving promising results, challenges such as environmental factors and accent variations need to be addressed. The project lays the foundation for future improvements and advancements in the field.

FUTURE SCOPE

The future scope of this project includes:

- Improving model robustness to handle diverse accents and environmental conditions.
- Exploring multi-modal approaches by integrating additional features like audio for enhanced accuracy.
- Conducting user studies to gather feedback and further refine the system based on real-world usage scenarios.

APPENDIX

Github link: https://github.com/smartinternz02/SI-GuidedProject-601538-1697561153/blob/87c226f98404958c6d600a42f169ca4d5ccea4bc/lip-reading-current%20(1).ipynb