End-to-end Lip Reading deep learning project.

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

Lip reading using machine learning can offer several benefits:

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.

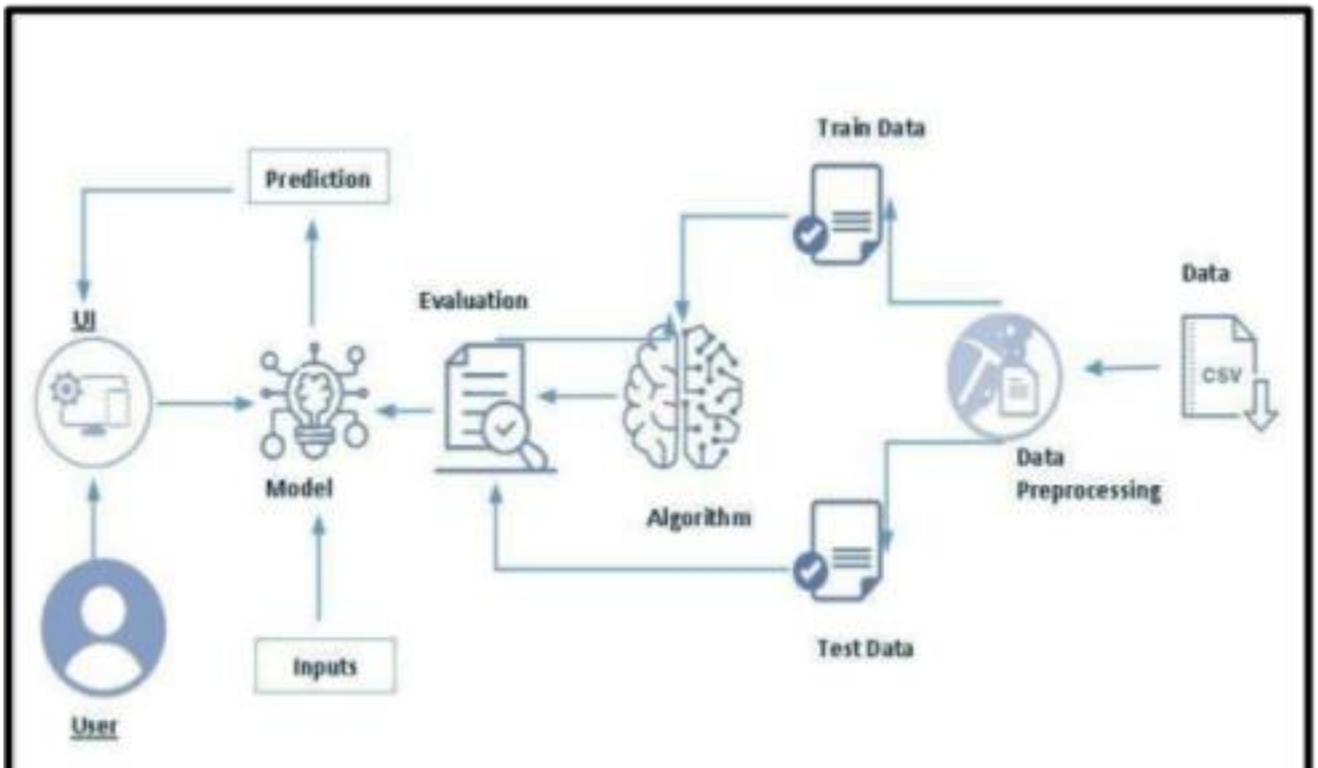
2. **No Need for Audio Data**: Traditional speech recognition models require large amounts of transcribed audio data for training. In contrast, an end-to-end lip reading system can be trained solely on video data, eliminating the need for transcribed audio, which can be expensive and time-consuming to 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.

Let us look at the Technical Architecture of the project.

Technical Architecture:



Project Flow:

- The user interacts with the UI to select the file.
- Selected input is analyzed by the model which is integrated/developed by you.
- Once the model analyzes the input, the prediction is showcased on the UI. To accomplish this, we have to complete all the activities listed below,

- Define Problem / Problem Understanding
    - 0 Specify the business problem
    - ○ Business requirements
    - ○ Literature Survey.
    - ○ Social or Business Impact.
- Data Collection & Preparation
    - ○ Collect the dataset
    - ○ Data Preparation
- Exploratory Data Analysis
    - 0 Descriptive statistical
    - ○ Visual Analysis
- Model Building
    - ○ Building a model.
    - 0 Training the model.
    - ○ Testing the model
- Model Deployment
    - 0 Save the best model
    - ○ Integrate with Web Framework
- Project Demonstration & Documentation
    - 0 Record explanation Video for project end to end solution
    - ○ Project Documentation-Step by step project development procedure

Prior Knowledge:

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

- VS Code :
    - o Refer to the link below to download VS Code.
    - o Link : https://code.visualstudio.com/download
- Create Project:
    - o Open VS Code.
    - o Create a Folder and name it "Lip_Reading" .
- Machine Learning Concepts
    - o Deep Learning:
      https://towardsdatascience.com/introducing-deep-learning-and-neural-networks-deep-learning-for r ookies-1-bd68f9cf5883
    - o NLP Models :
      https://medium.com/voice-tech-podcast/an-overview-of-rnn-lstm-gru-79ed642751c6
- Web concepts:
    - o Get the gist on streamlit :
      https://www.geeksforgeeks.org/a-beginners-guide-to-streamlit/

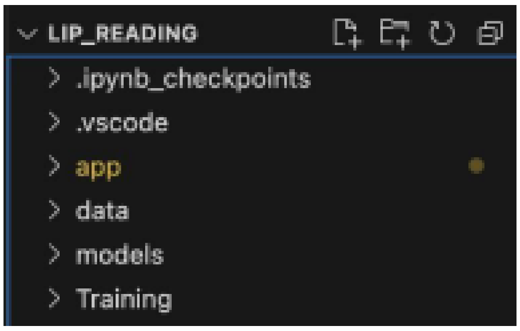
Project Objectives:

By the end of this project you will:

- Know fundamental concepts and techniques of Deep Learning.
- Gain a broad understanding of Lip Reading.
- Know how to train models in an efficient way.
- Know how to build a web application using the Streamlit framework.

Project Structure:

Create the Project folder which contains files as shown below



- We are building a Streamlit application which needs an app folder for a website.
- models folder contains your saved models.
- The Training folder contains the code for building/training/testing the model.
- The Dataset folder contains the videos and alignments.

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 anyone.

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 (Student Will Write)

A literature survey would involve researching and reviewing existing studies, articles, and other publications on the topic of the project. The survey would aim to gather information on current systems, their strengths and weaknesses, and any gaps in knowledge that the project could address. The literature survey would also look at the methods and
techniques used in previous projects, and any relevant data or findings that could inform the design and implementation of the current project.

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.

Milestone 2: Data Collection & Preparation

DL depends heavily on data. It is the most crucial aspect that makes algorithm training possible. So, this section allows you to acquire the required dataset.

NOTE : If you don't have a GPU, then train your model on kaggle.

Activity 1: Loading the data.

Link for the required dataset :

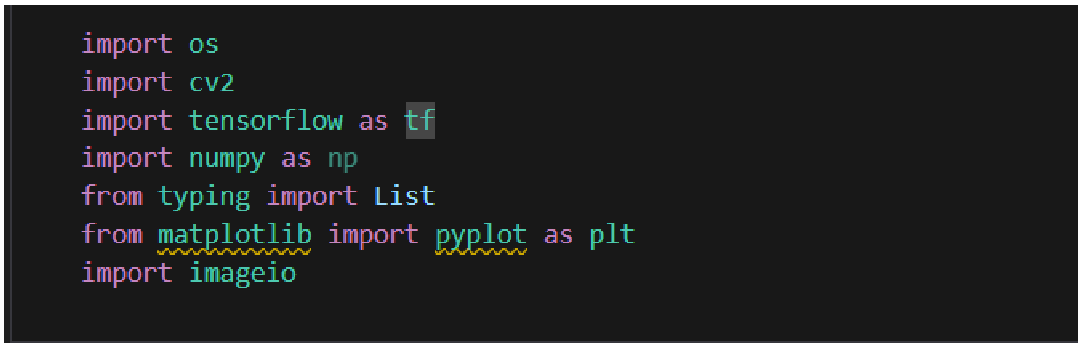
https://www.kaggle.com/datasets/rishisrdy/lipreading Download the provided data

and load the data into the data folder

Activity 2: Data Preparation

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

Let's import the required packages

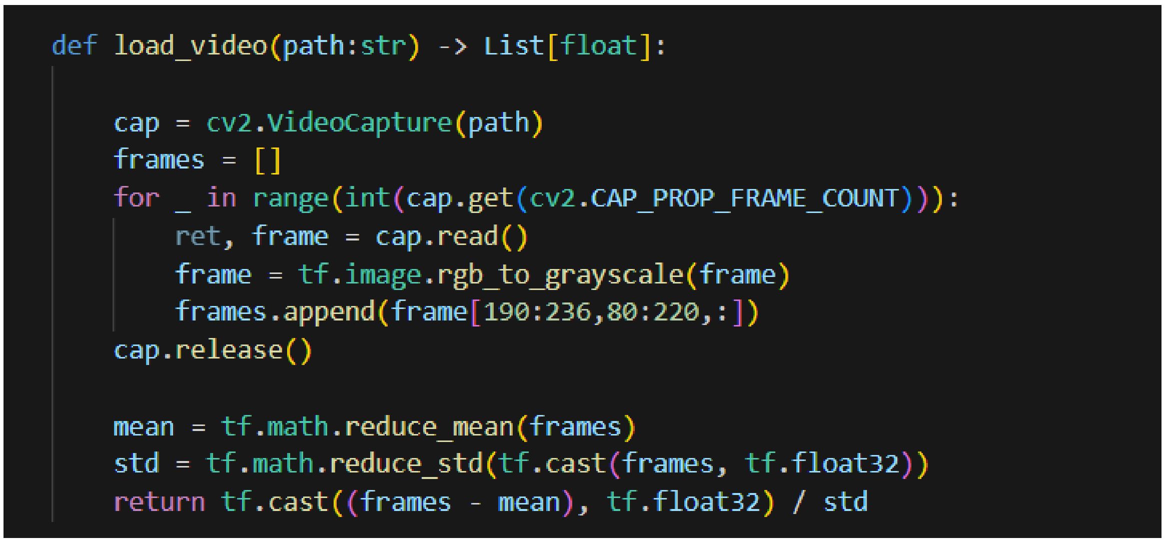


```python
import os
import cv2
import tensorflow as tf
import numpy as np
from typing import List
from matplotlib import pyplot as plt
import imageio
```

We require 5 functions to process the data

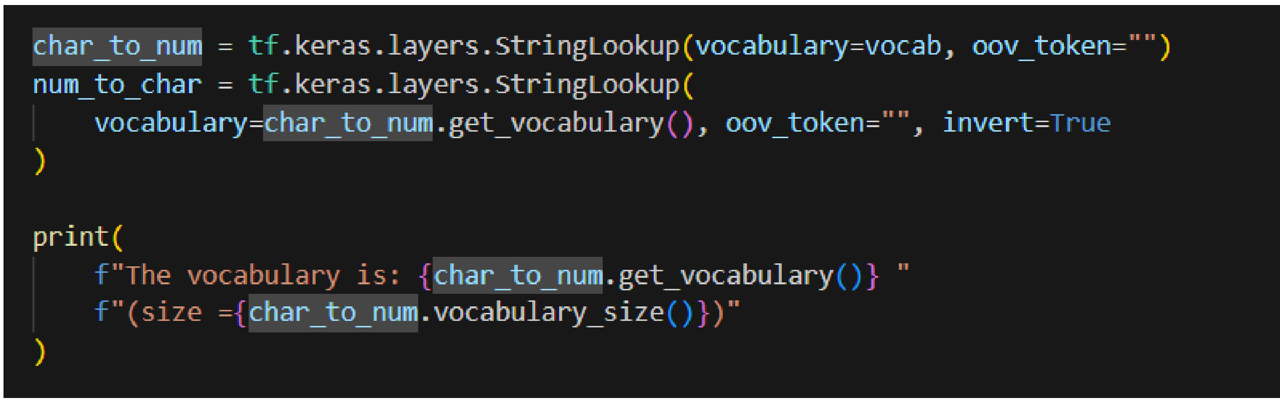
- load_video()
- char_to_num
- num_to_char
- load_alignments()
- load_data()

- **load_video():**



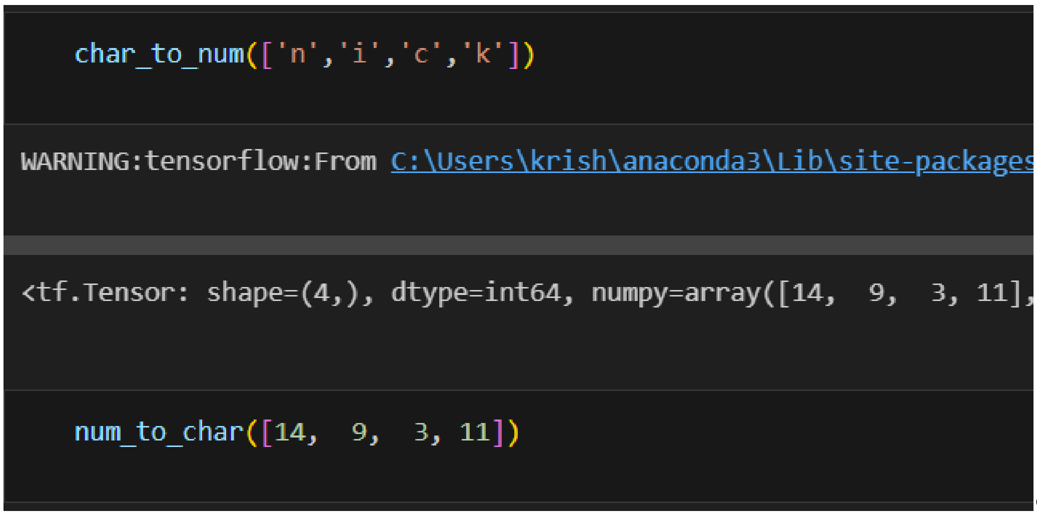
```python
def load_video(path:str) -> List[float]:

    cap = cv2.VideoCapture(path)
    frames = []
    for _ in range(int(cap.get(cv2.CAP_PROP_FRAME_COUNT))):
        ret, frame = cap.read()
        frame = tf.image.rgb_to_grayscale(frame)
        frames.append(frame[190:236,80:220,:])
    cap.release()

    mean = tf.math.reduce_mean(frames)
    std = tf.math.reduce_std(tf.cast(frames, tf.float32))
    return tf.cast((frames - mean), tf.float32) / std
```

- **char_to_num:**



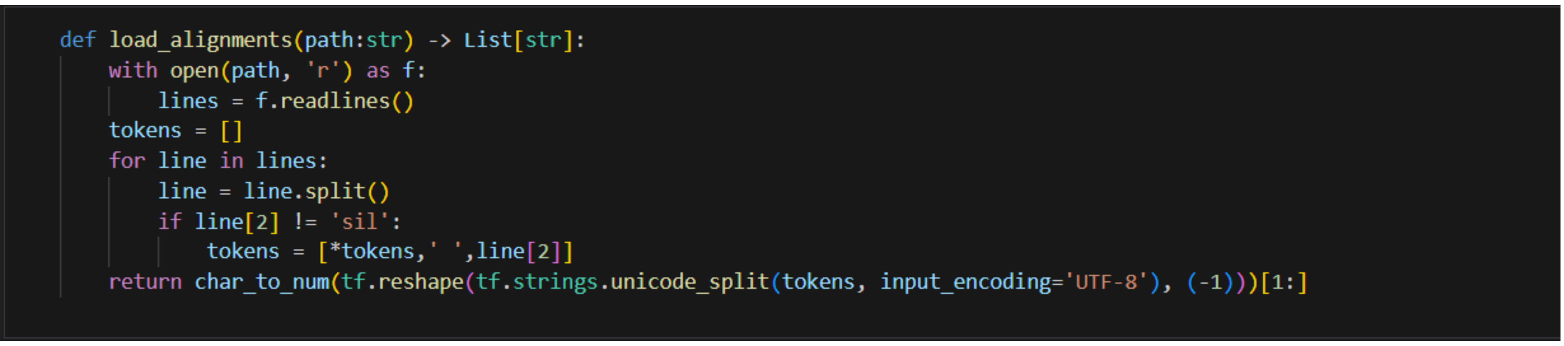
```python
char_to_num = tf.keras.layers.StringLookup(vocabulary=vocab, oov_token="")
num_to_char = tf.keras.layers.StringLookup(
    vocabulary=char_to_num.get_vocabulary(), oov_token="", invert=True
)

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

- **num_to_char:**



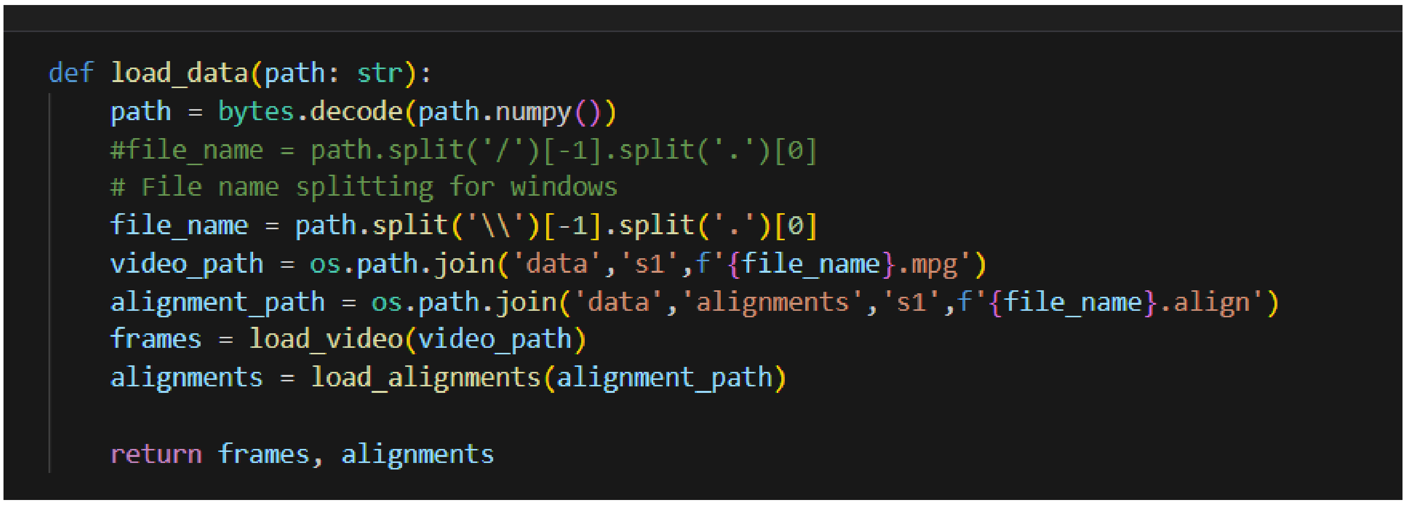
```
    char_to_num(['n','i','c','k'])

WARNING:tensorflow:From C:\Users\krish\anaconda3\Lib\site-packages

<tf.Tensor: shape=(4,), dtype=int64, numpy=array([14,  9,  3, 11],

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

- **load_alignments():**

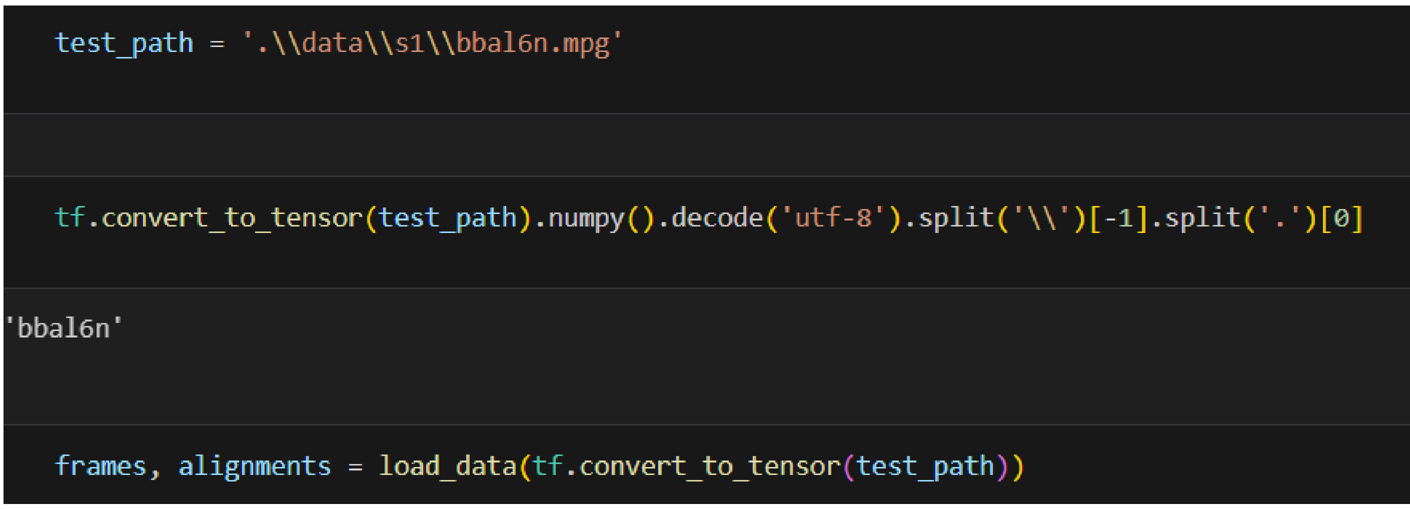


```python
def load_alignments(path:str) -> List[str]:
    with open(path, 'r') as f:
        lines = f.readlines()
    tokens = []
    for line in lines:
        line = line.split()
        if line[2] != 'sil':
            tokens = [*tokens,' ',line[2]]
    return char_to_num(tf.reshape(tf.strings.unicode_split(tokens,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

- **load_data():**


```
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
```

Let’s convert the data into tensors using the above functions


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

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

```
'bbal6n'
```

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

Milestone 3: Exploratory Data Analysis

Activity 1: Descriptive statistical

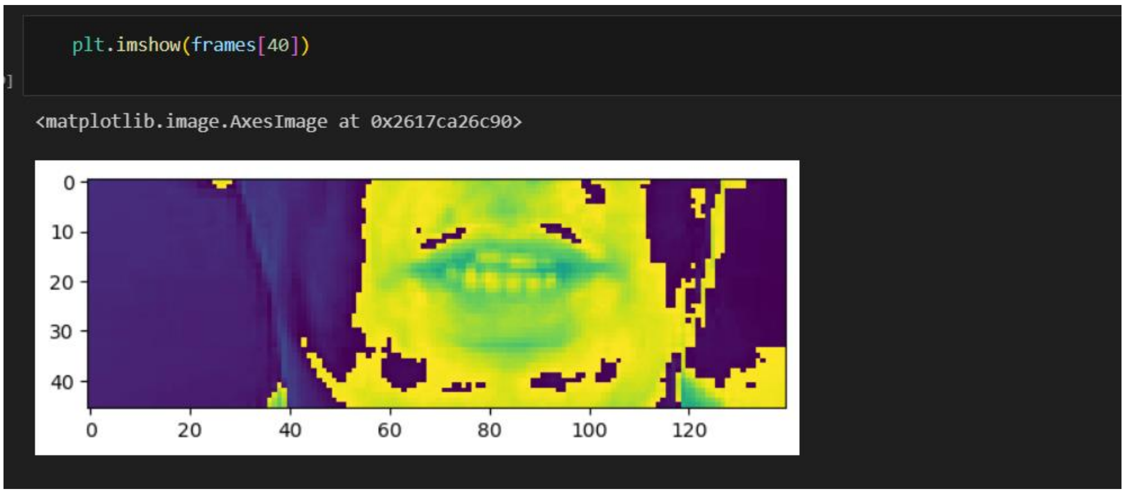
Descriptive analysis is to study the basic features of data with the statistical process. With this describe function we can understand the unique, top and frequent values of categorical features. And we can find mean, std, min, max and percentile values of continuous features. Which are not suitable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

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

Activity 2: Visual analysis

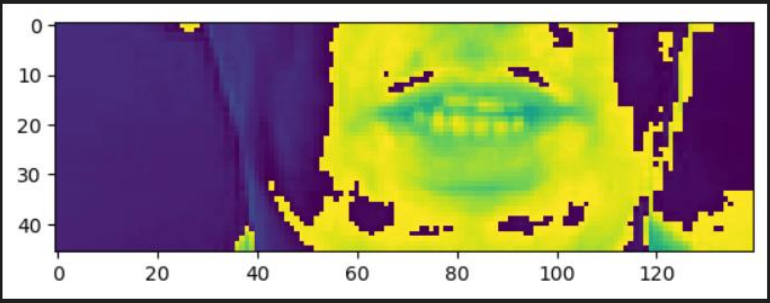
Visual analysis is the process of using visual representations, such as charts, plots, and graphs, to explore and understand data. It is a way to quickly identify patterns, trends, and outliers in the data, which can help to gain insights and make informed decisions.

Let’s plot the converted frames



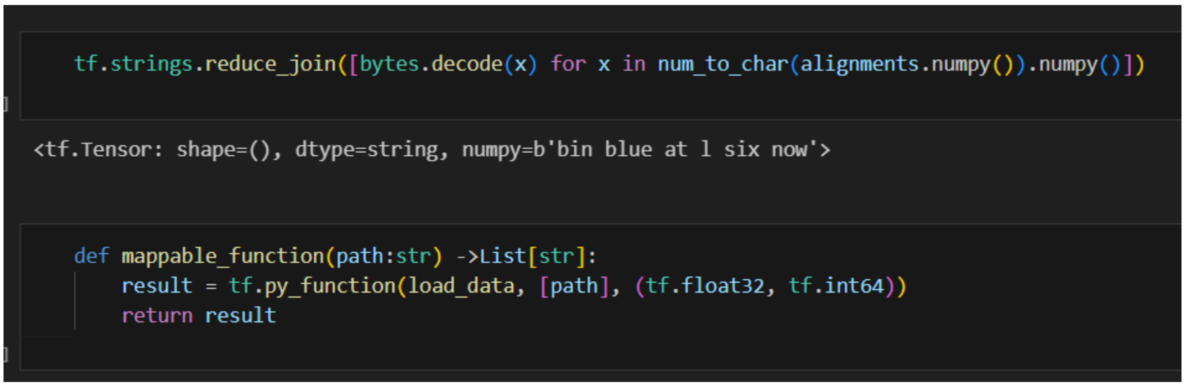
```
plt.imshow(frames[40])
```

```
<matplotlib.image.AxesImage at 0x2617ca26c90>
```



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

First we will create a mapable function

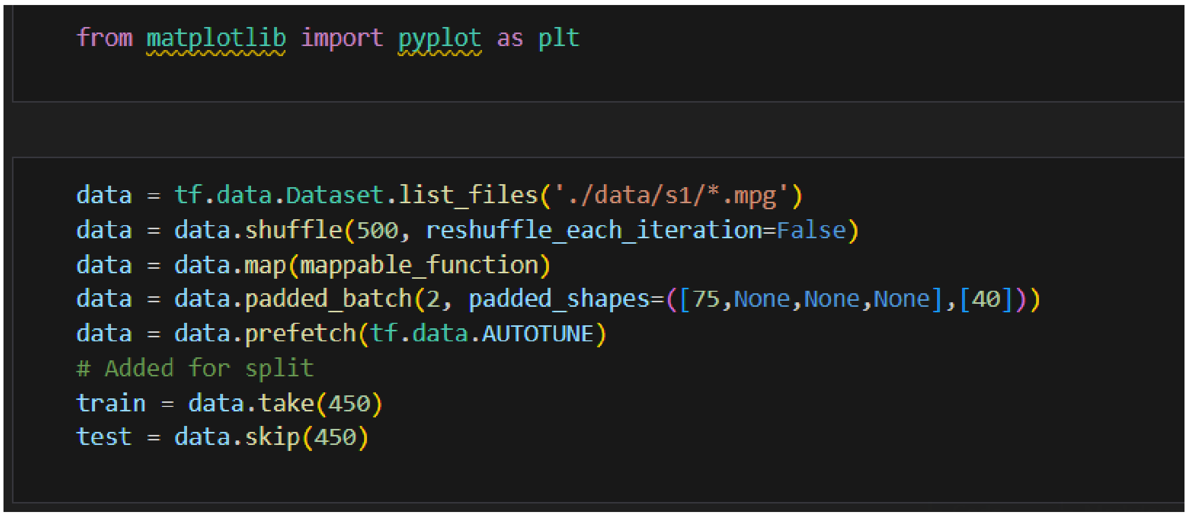


```
tf.strings.reduce_join([bytes.decode(x) for x in num_to_char(alignments.numpy()).numpy()])
```

```
<tf.Tensor: shape=(), dtype=string, numpy=b'bin blue at l six now'>
```

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

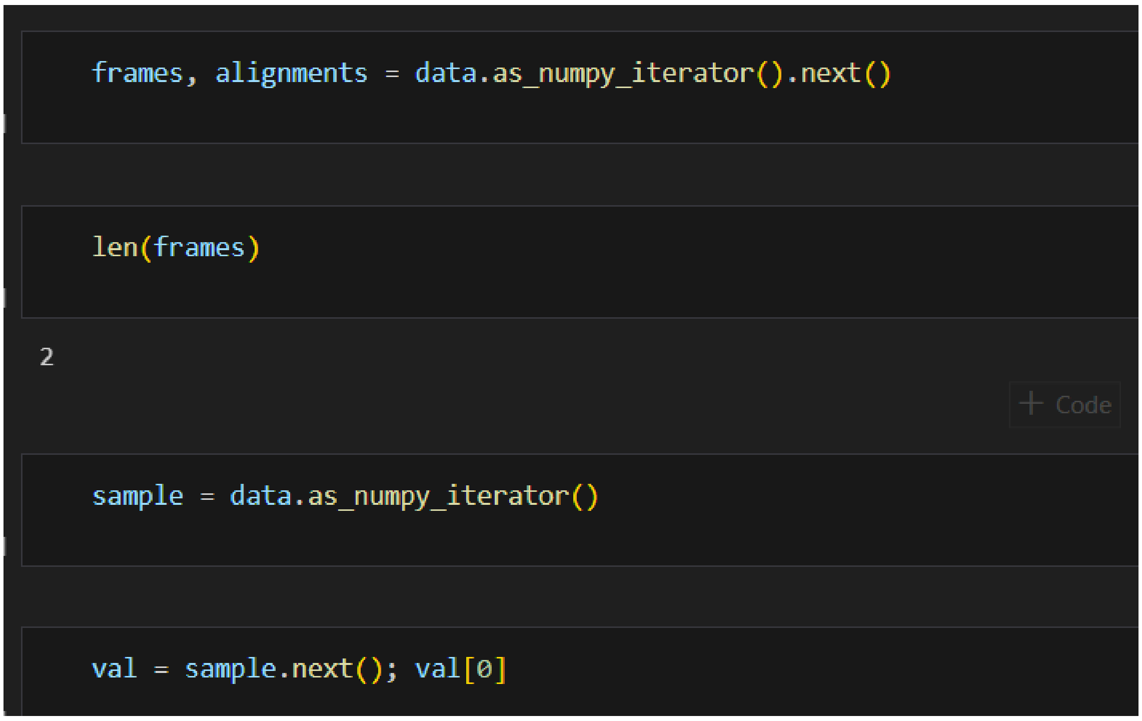
The following code is to divide the preprocessed data into train and test data equally.



```
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)
```

The below codes are to print the preprocessed data



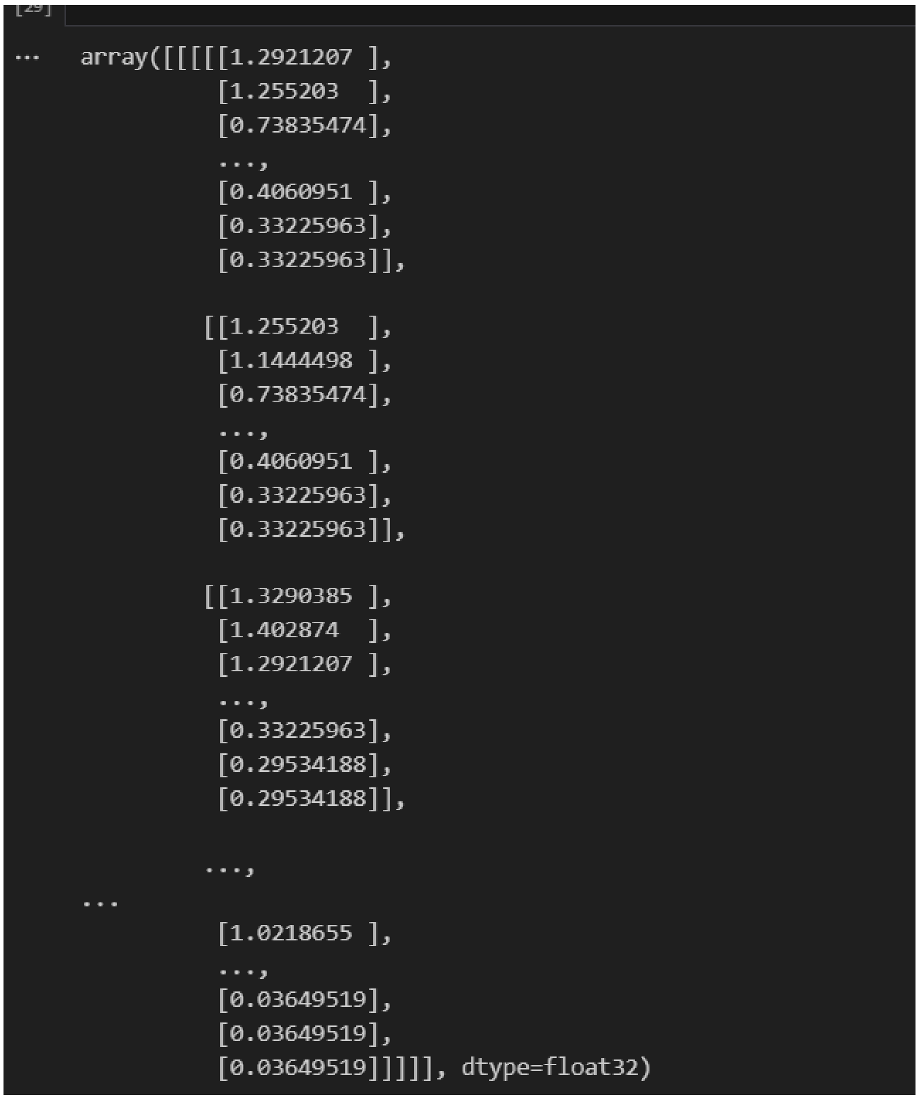
```
frames, alignments = data.as_numpy_iterator().next()
```

```
len(frames)
```

```
2
```

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

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

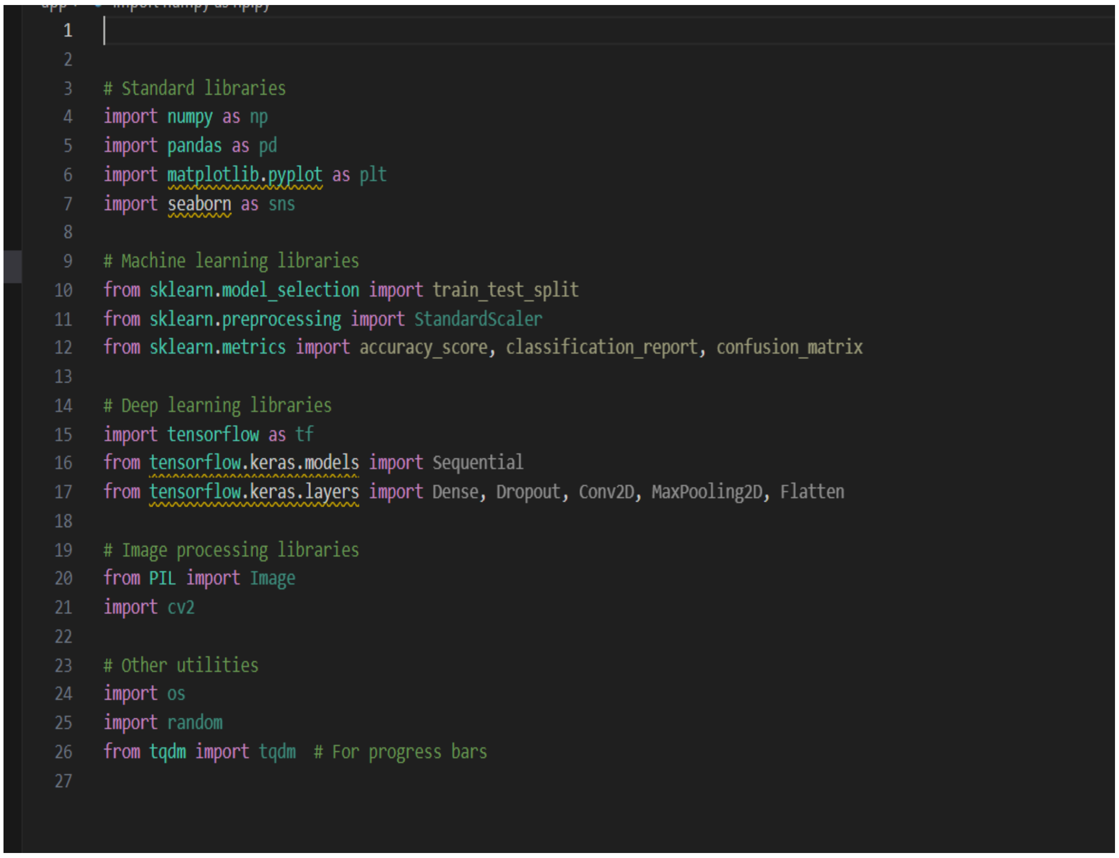


```
...   array([[[[[1.2921207 ],
                [1.255203  ],
                [0.73835474],
                ...,
                [0.4060951 ],
                [0.33225963],
                [0.33225963]],

               [[1.255203  ],
                [1.1444498 ],
                [0.73835474],
                ...,
                [0.4060951 ],
                [0.33225963],
                [0.33225963]],

               [[1.3290385 ],
                [1.402874  ],
                [1.2921207 ],
                ...,
                [0.33225963],
                [0.29534188],
                [0.29534188]],

               ...,
      ...
                [1.0218655 ],
                ...,
                [0.03649519],
                [0.03649519],
                [0.03649519]]]]], dtype=float32)
```

Milestone 4: Model Building

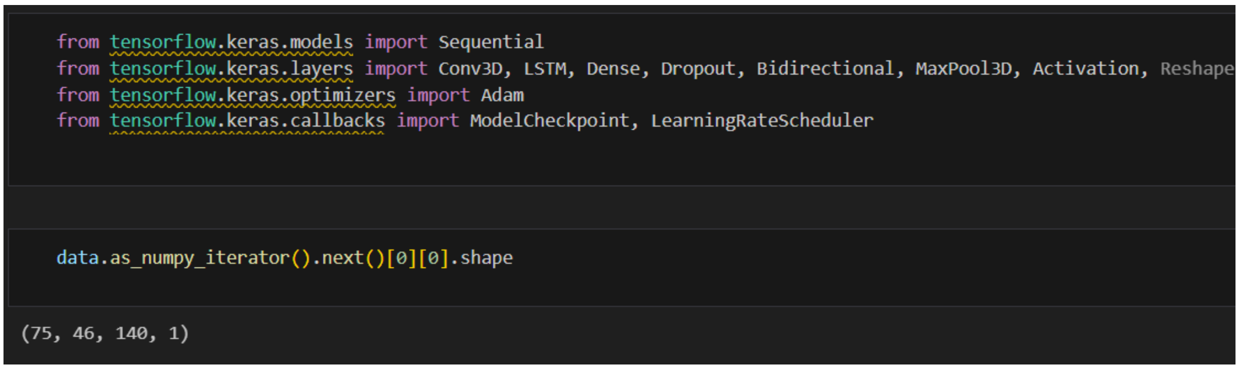
Activity 1: Importing necessary libraries

We'll import some necessary libraries using the following code



```python

# Standard libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Machine learning libraries
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Deep learning libraries
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Conv2D, MaxPooling2D, Flatten

# Image processing libraries
from PIL import Image
import cv2

# Other utilities
import os
import random
from tqdm import tqdm  # For progress bars

```

Activity 2: Design the Deep Neural Network and building model

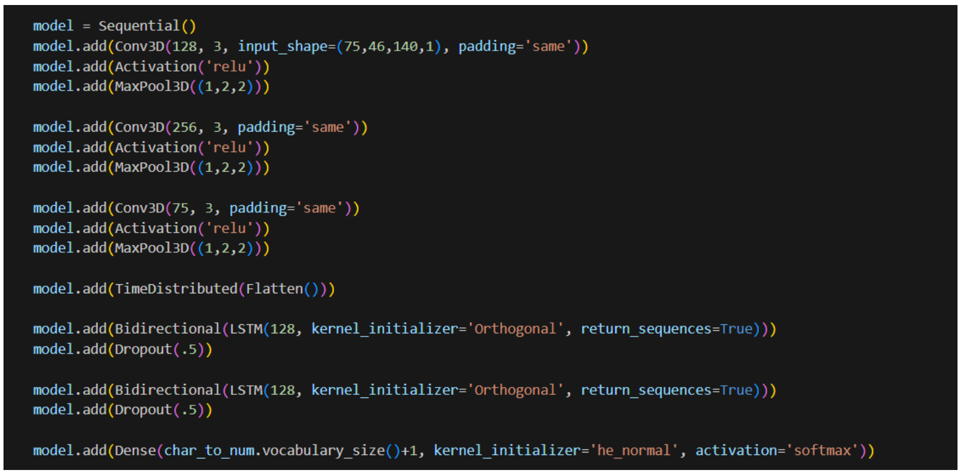


```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv3D, LSTM, Dense, Dropout, Bidirectional, MaxPool3D, Activation, Reshape
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ModelCheckpoint, LearningRateScheduler
```

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

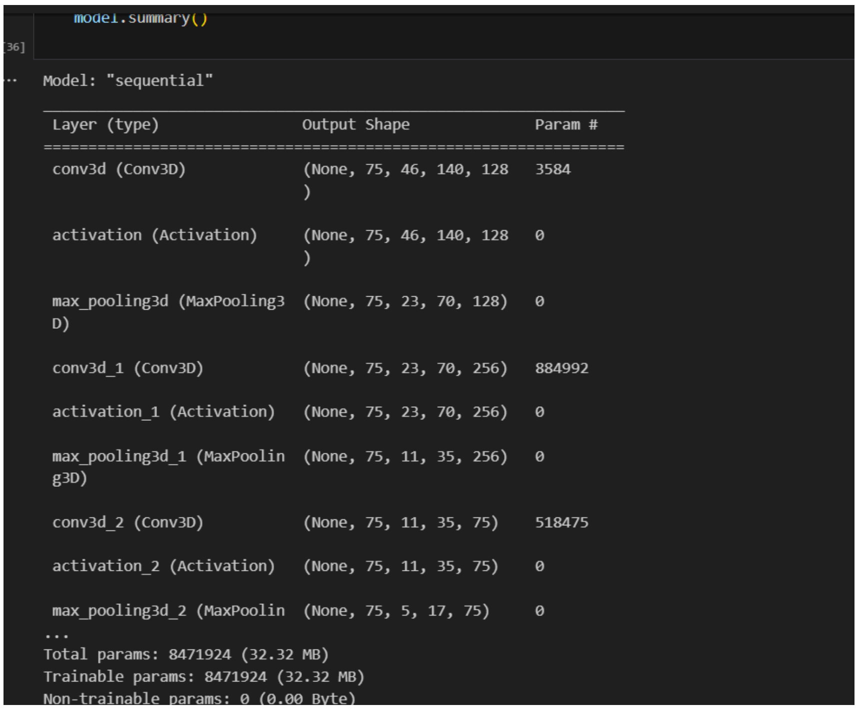
```
(75, 46, 140, 1)
```

- Model Building :

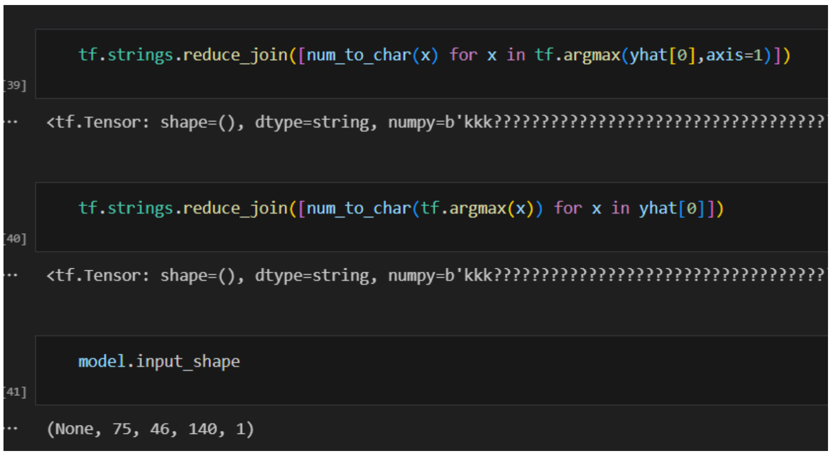


```python
model = Sequential()
model.add(Conv3D(128, 3, input_shape=(75,46,140,1), padding='same'))
model.add(Activation('relu'))
model.add(MaxPool3D((1,2,2)))

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'))
```

Summary:



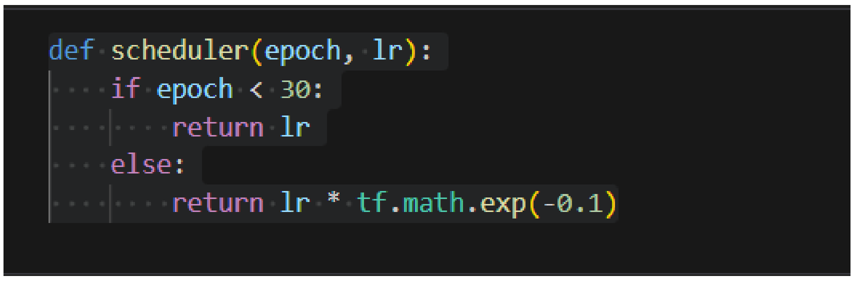
```
model.summary()
[36]
... Model: "sequential"
    _________________________________________________________________
     Layer (type)                Output Shape              Param #
    =================================================================
     conv3d (Conv3D)             (None, 75, 46, 140, 128   3584
                                 )

     activation (Activation)     (None, 75, 46, 140, 128   0
                                 )

     max_pooling3d (MaxPooling3  (None, 75, 23, 70, 128)   0
     D)

     conv3d_1 (Conv3D)           (None, 75, 23, 70, 256)   884992

     activation_1 (Activation)   (None, 75, 23, 70, 256)   0

     max_pooling3d_1 (MaxPoolin  (None, 75, 11, 35, 256)   0
     g3D)

     conv3d_2 (Conv3D)           (None, 75, 11, 35, 75)    518475

     activation_2 (Activation)   (None, 75, 11, 35, 75)    0

     max_pooling3d_2 (MaxPoolin  (None, 75, 5, 17, 75)     0
     ...
    Total params: 8471924 (32.32 MB)
    Trainable params: 8471924 (32.32 MB)
    Non-trainable params: 0 (0.00 Byte)
```



```
    tf.strings.reduce_join([num_to_char(x) for x in tf.argmax(yhat[0],axis=1)])
[39]
... <tf.Tensor: shape=(), dtype=string, numpy=b'kkk???????????????????????????????????

    tf.strings.reduce_join([num_to_char(tf.argmax(x)) for x in yhat[0]])
[40]
... <tf.Tensor: shape=(), dtype=string, numpy=b'kkk???????????????????????????????????

    model.input_shape
[41]
... (None, 75, 46, 140, 1)
```

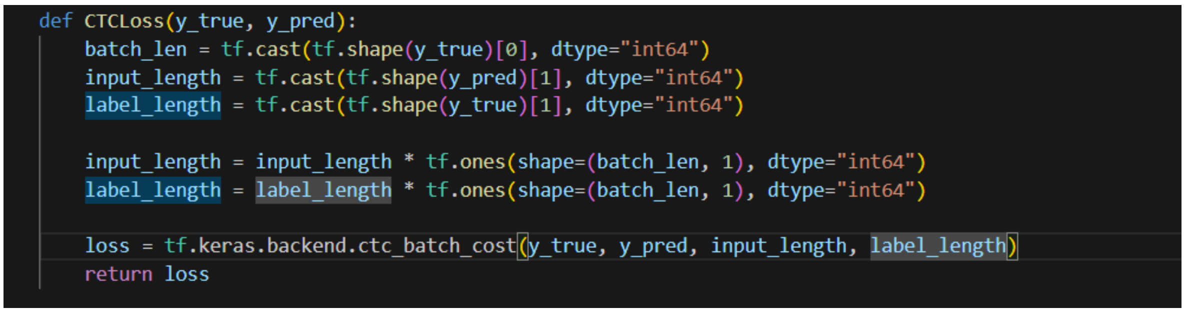
Activity 3: Train the models

First initialize the learning rate scheduler function:



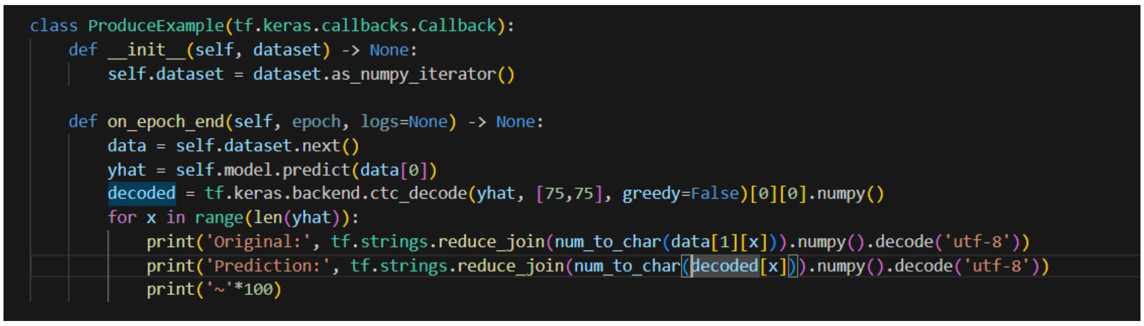
```python
def scheduler(epoch, lr):
    if epoch < 30:
        return lr
    else:
        return lr * tf.math.exp(-0.1)
```

custom CTC (Connectionist Temporal Classification) loss function:



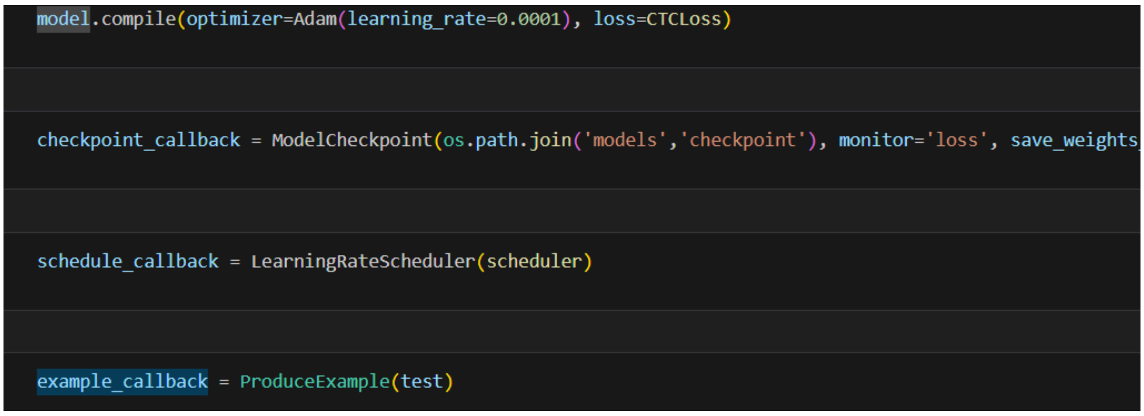
```
def CTCLoss(y_true, y_pred):
    batch_len = tf.cast(tf.shape(y_true)[0], dtype="int64")
    input_length = tf.cast(tf.shape(y_pred)[1], dtype="int64")
    label_length = tf.cast(tf.shape(y_true)[1], dtype="int64")

    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
```

custom callback:



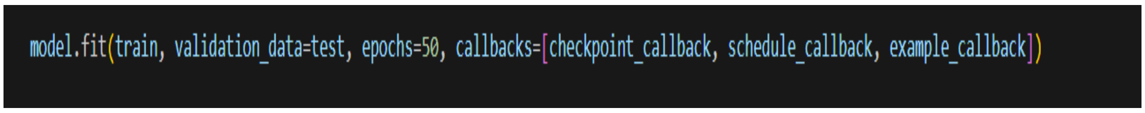
```
class ProduceExample(tf.keras.callbacks.Callback):
    def __init__(self, dataset) -> None:
        self.dataset = dataset.as_numpy_iterator()

    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)
```

Then we compile the model



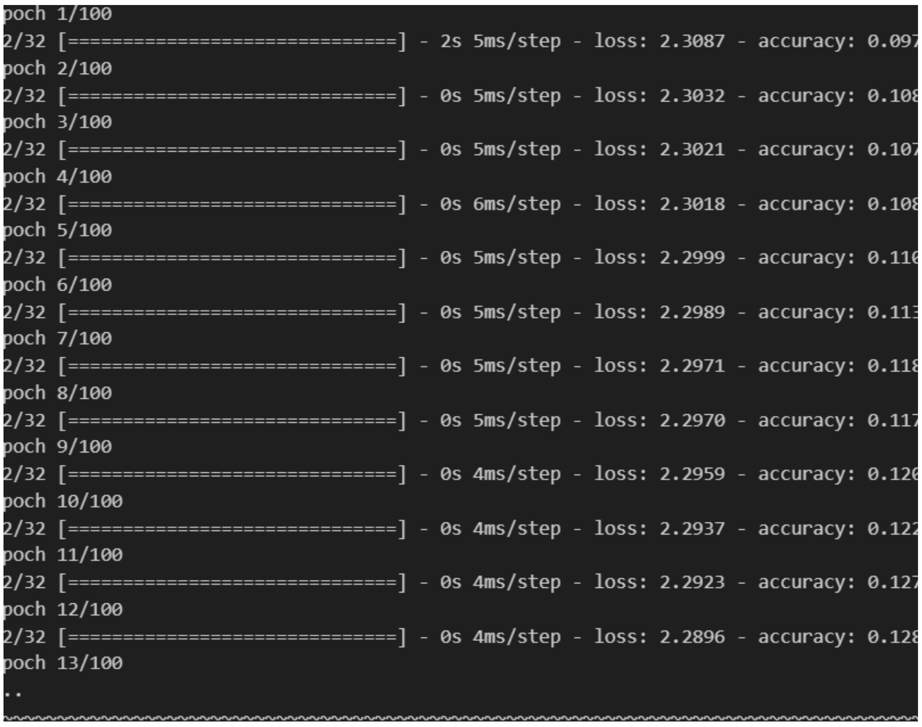
```
model.compile(optimizer=Adam(learning_rate=0.0001), loss=CTCLoss)

checkpoint_callback = ModelCheckpoint(os.path.join('models','checkpoint'), monitor='loss', save_weights

schedule_callback = LearningRateScheduler(scheduler)

example_callback = ProduceExample(test)
```

Now it’s time to train the model



```
model.fit(train, validation_data=test, epochs=50, callbacks=[checkpoint_callback, schedule_callback, example_callback])
```

Let’s wait for the model to train until 100 epochs.



```
poch 1/100
2/32 [==============================] - 2s 5ms/step - loss: 2.3087 - accuracy: 0.097
poch 2/100
2/32 [==============================] - 0s 5ms/step - loss: 2.3032 - accuracy: 0.108
poch 3/100
2/32 [==============================] - 0s 5ms/step - loss: 2.3021 - accuracy: 0.107
poch 4/100
2/32 [==============================] - 0s 6ms/step - loss: 2.3018 - accuracy: 0.108
poch 5/100
2/32 [==============================] - 0s 5ms/step - loss: 2.2999 - accuracy: 0.110
poch 6/100
2/32 [==============================] - 0s 5ms/step - loss: 2.2989 - accuracy: 0.113
poch 7/100
2/32 [==============================] - 0s 5ms/step - loss: 2.2971 - accuracy: 0.118
poch 8/100
2/32 [==============================] - 0s 5ms/step - loss: 2.2970 - accuracy: 0.117
poch 9/100
2/32 [==============================] - 0s 4ms/step - loss: 2.2959 - accuracy: 0.120
poch 10/100
2/32 [==============================] - 0s 4ms/step - loss: 2.2937 - accuracy: 0.122
poch 11/100
2/32 [==============================] - 0s 4ms/step - loss: 2.2923 - accuracy: 0.127
poch 12/100
2/32 [==============================] - 0s 4ms/step - loss: 2.2896 - accuracy: 0.128
poch 13/100
..
```

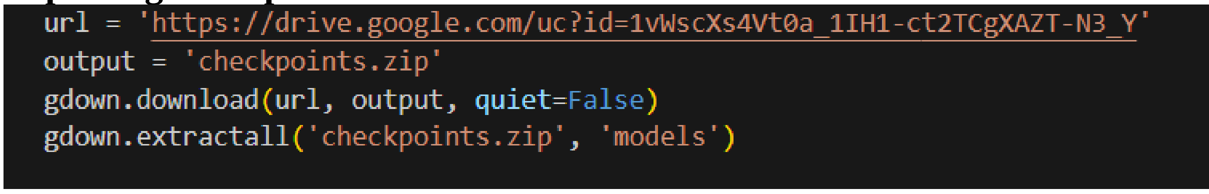
Milestone 5: Model Deployment

Activity 1: Save the model

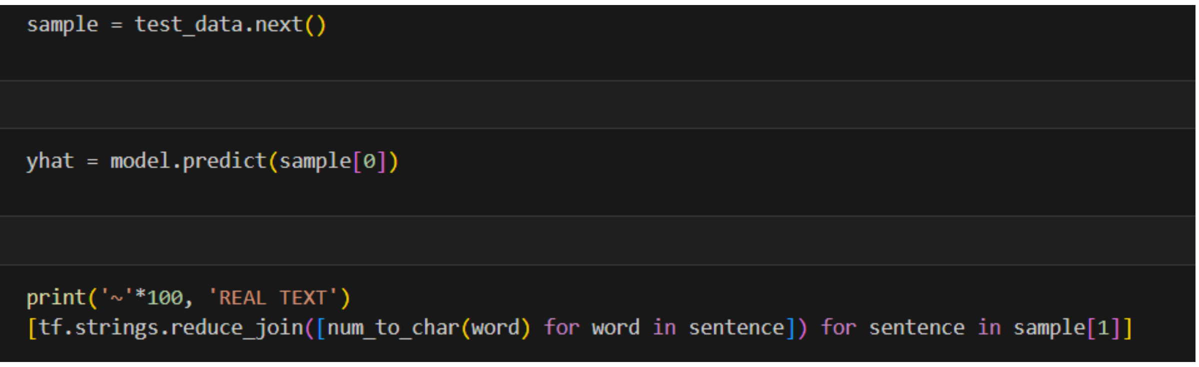
Checkpoints:https://drive.google.com/drive/folders/1YWpxapm0TcjijYf-F8GZiuUaHKDUSAMv?usp=drive_link

Activity 2: Make a Prediction :

importing checkpoints:



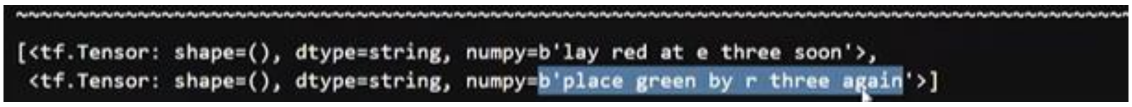
```
url = 'https://drive.google.com/uc?id=1vWscXs4Vt0a_1IH1-ct2TCgXAZT-N3_Y'
output = 'checkpoints.zip'
gdown.download(url, output, quiet=False)
gdown.extractall('checkpoints.zip', 'models')
```



```
sample = test_data.next()
```

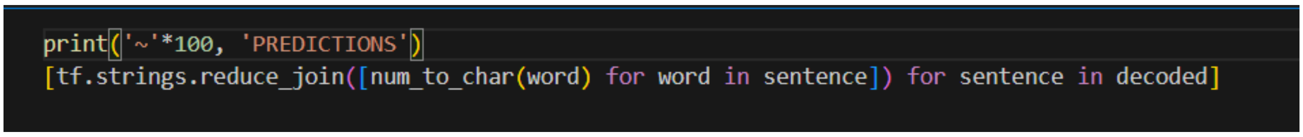
```
yhat = model.predict(sample[0])
```

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



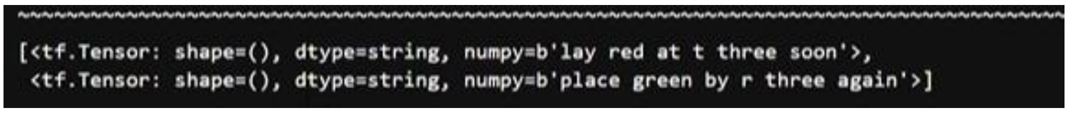
```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

[<tf.Tensor: shape=(), dtype=string, numpy=b'lay red at e three soon'>,
 <tf.Tensor: shape=(), dtype=string, numpy=b'place green by r three again'>]
```

Prediction:



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

Output:



```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
[<tf.Tensor: shape=(), dtype=string, numpy=b'lay red at t three soon'>,
 <tf.Tensor: shape=(), dtype=string, numpy=b'place green by r three again'>]
```

Milestone-6 : Application building:

In this section, we will be building a web application that is integrated to the model we built.

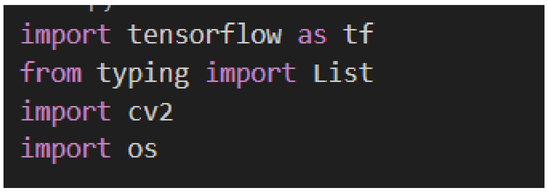
We will be using the streamlit package for our website development.

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

Activity 1: Create an app folder and create following .py files in it:

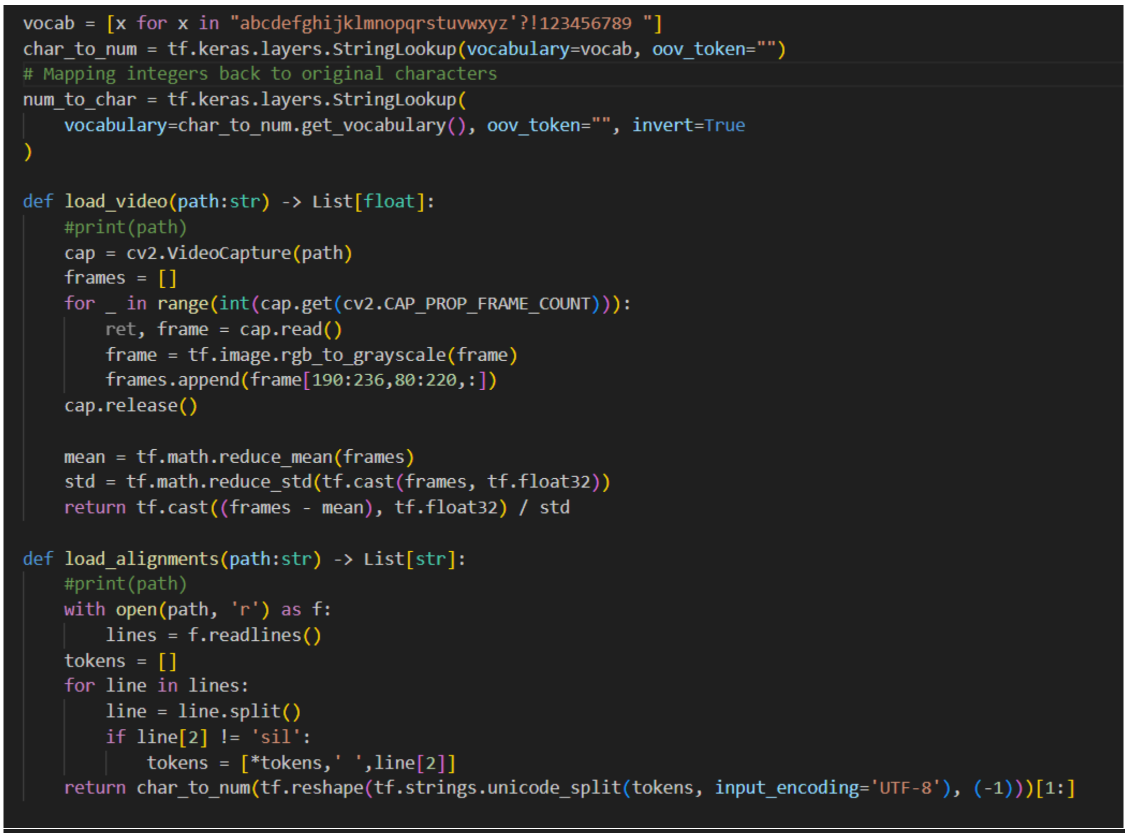
Let’s write the code for **“utils.py”**

- **Importing necessary packages:**

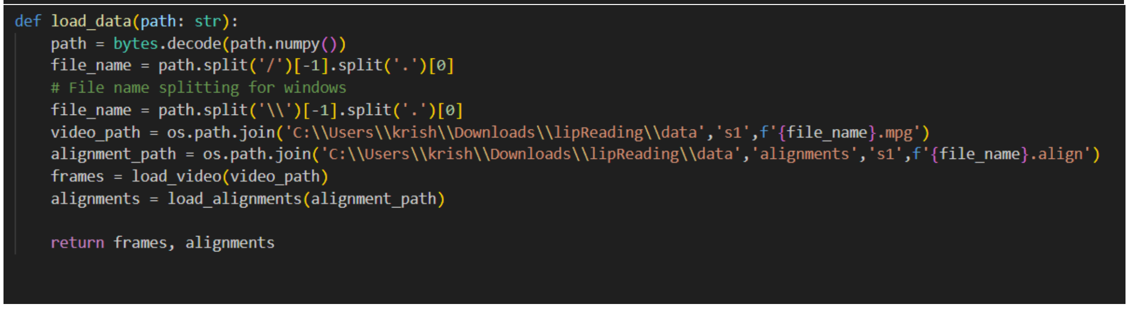


```python
import tensorflow as tf
from typing import List
import cv2
import os
```

- **Defining char_to_num, num_to_char, load_video(), load_alignments(), load_data() functions:**



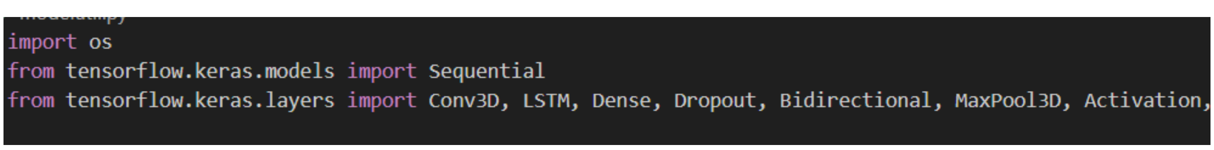
```python
vocab = [x for x in "abcdefghijklmnopqrstuvwxyz'?!123456789 "]
char_to_num = tf.keras.layers.StringLookup(vocabulary=vocab, oov_token="")
# Mapping integers back to original characters
num_to_char = tf.keras.layers.StringLookup(
    vocabulary=char_to_num.get_vocabulary(), oov_token="", invert=True
)

def load_video(path:str) -> List[float]:
    #print(path)
    cap = cv2.VideoCapture(path)
    frames = []
    for _ in range(int(cap.get(cv2.CAP_PROP_FRAME_COUNT))):
        ret, frame = cap.read()
        frame = tf.image.rgb_to_grayscale(frame)
        frames.append(frame[190:236,80:220,:])
    cap.release()

    mean = tf.math.reduce_mean(frames)
    std = tf.math.reduce_std(tf.cast(frames, tf.float32))
    return tf.cast((frames - mean), tf.float32) / std

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



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

    return frames, alignments
```

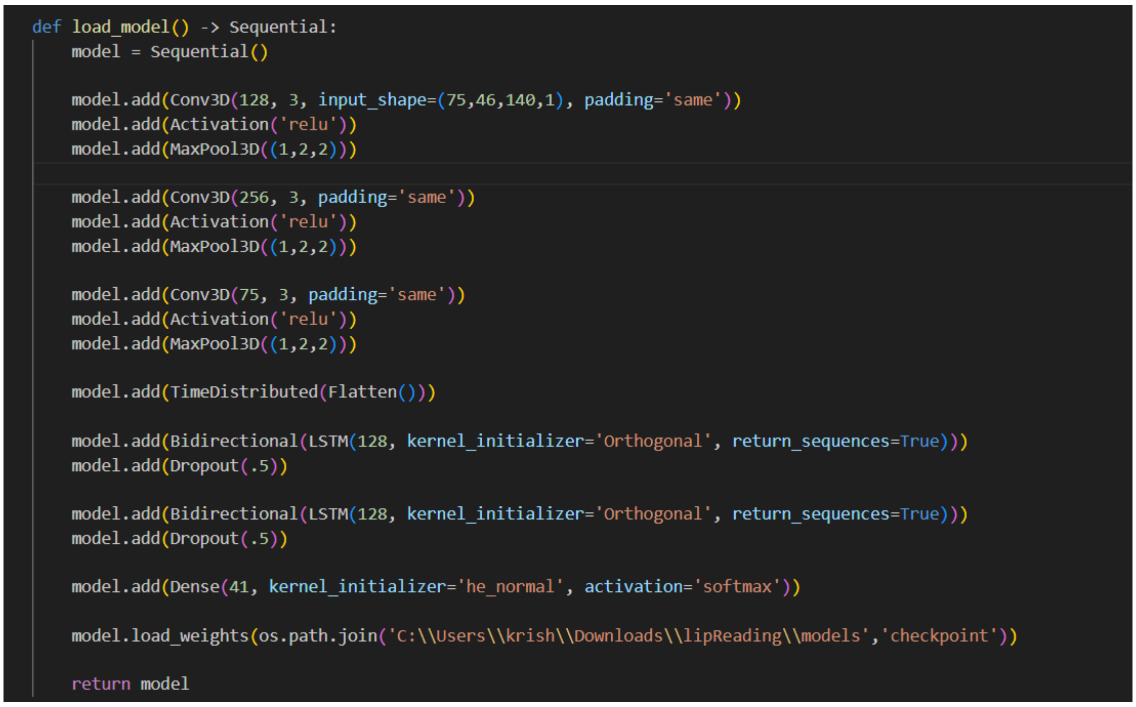
Let's write code for **"modelutils.py"**

- **Import all the necessary packages:**



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

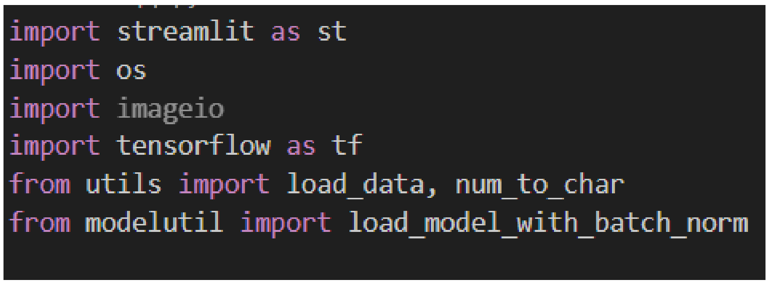
● **Defining load_model() function :**



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

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

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

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

    model.add(TimeDistributed(Flatten()))

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

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

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

    model.load_weights(os.path.join('C:\\Users\\krish\\Downloads\\lipReading\\models','checkpoint'))

    return model
```

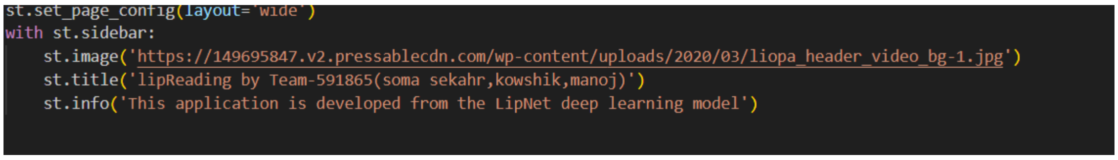
Let’s write code for **“streamlitapp.py”**

● **Importing necessary packages and functions:**



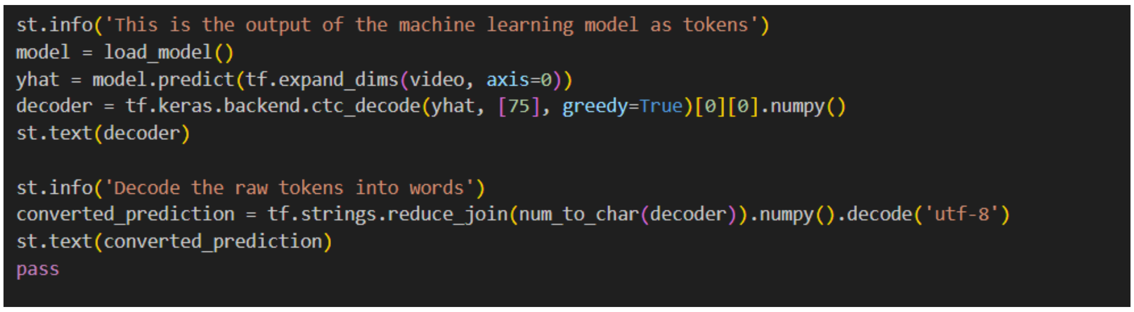
```python
import streamlit as st
import os
import imageio
import tensorflow as tf
from utils import load_data, num_to_char
from modelutil import load_model_with_batch_norm
```

● **Creating sidebar:**



```python
st.set_page_config(layout='wide')
with st.sidebar:
    st.image('https://149695847.v2.pressablecdn.com/wp-content/uploads/2020/03/liopa_header_video_bg-1.jpg')
    st.title('lipReading by Team-591865(soma sekahr,kowshik,manoj)')
    st.info('This application is developed from the LipNet deep learning model')
```

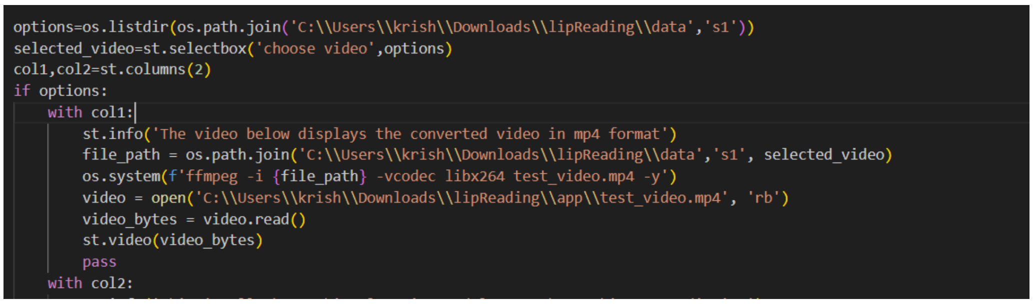
- **Drop down menu for selecting the input :**



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

st.info('Decode the raw tokens into words')
converted_prediction = tf.strings.reduce_join(num_to_char(decoder)).numpy().decode('utf-8')
st.text(converted_prediction)
pass
```

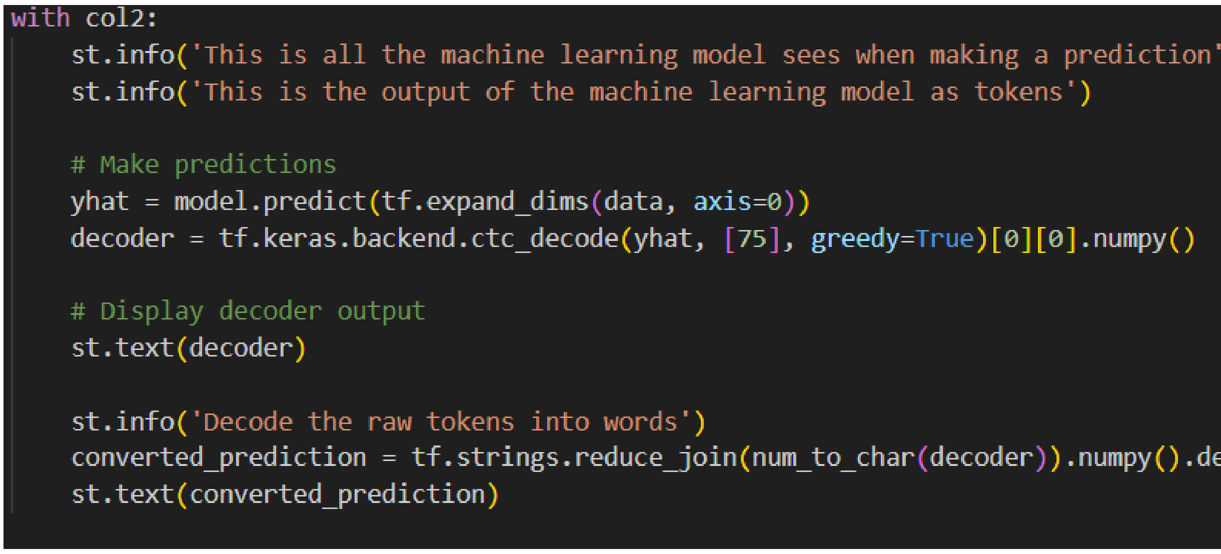
- **Creating two columns :**

  - **col1:**



```python
options=os.listdir(os.path.join('C:\\Users\\krish\\Downloads\\lipReading\\data','s1'))
selected_video=st.selectbox('choose video',options)
col1,col2=st.columns(2)
if options:
    with col1:
        st.info('The video below displays the converted video in mp4 format')
        file_path = os.path.join('C:\\Users\\krish\\Downloads\\lipReading\\data','s1', selected_video)
        os.system(f'ffmpeg -i {file_path} -vcodec libx264 test_video.mp4 -y')
        video = open('C:\\Users\\krish\\Downloads\\lipReading\\app\\test_video.mp4', 'rb')
        video_bytes = video.read()
        st.video(video_bytes)
        pass
    with col2:
```

  - **col2 :**



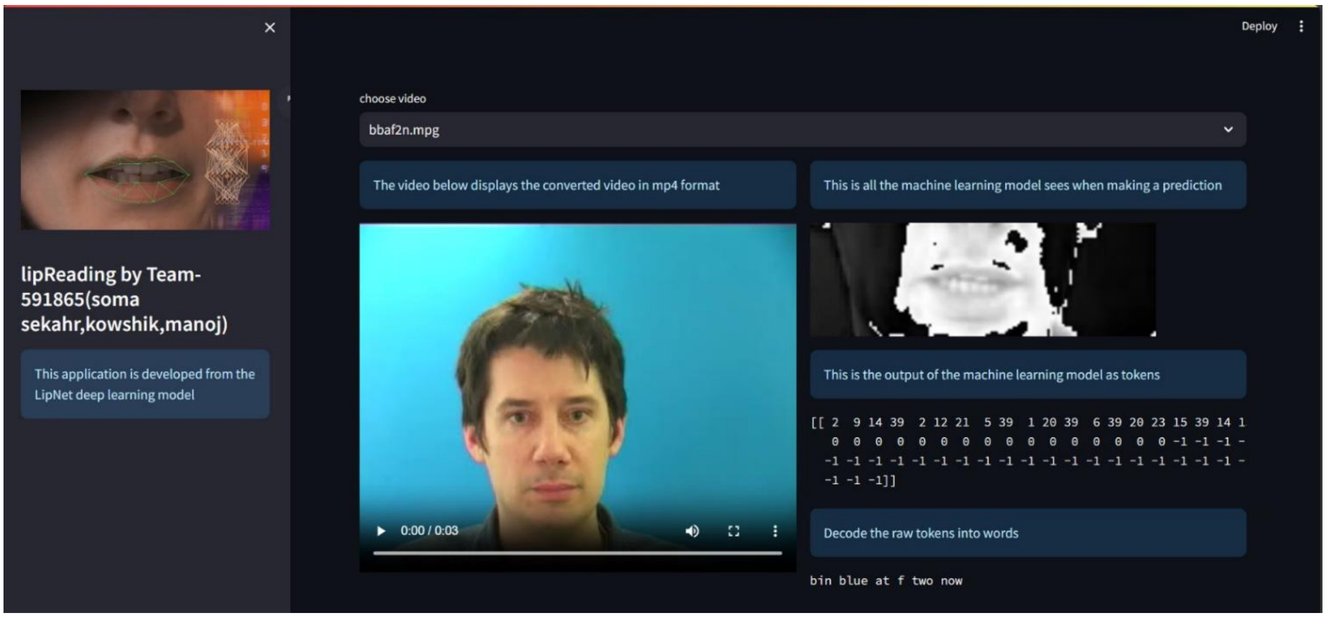
```python
with col2:
    st.info('This is all the machine learning model sees when making a prediction'
    st.info('This is the output of the machine learning model as tokens')

    # Make predictions
    yhat = model.predict(tf.expand_dims(data, axis=0))
    decoder = tf.keras.backend.ctc_decode(yhat, [75], greedy=True)[0][0].numpy()

    # Display decoder output
    st.text(decoder)

    st.info('Decode the raw tokens into words')
    converted_prediction = tf.strings.reduce_join(num_to_char(decoder)).numpy().de
    st.text(converted_prediction)
```

- col1 will display the selected video and play it.
- col2 will display :
  - ➔ animation of lips.
  - ➔ raw output of your model

➔ result prediction

To run your website go to your terminal with your respective directory and run the command : ●

“streamlit run streamlitapp.py”



lipReading by Team-591865(soma sekahr,kowshik,manoj)

This application is developed from the LipNet deep learning model

choose video

bbaf2n.mpg

The video below displays the converted video in mp4 format

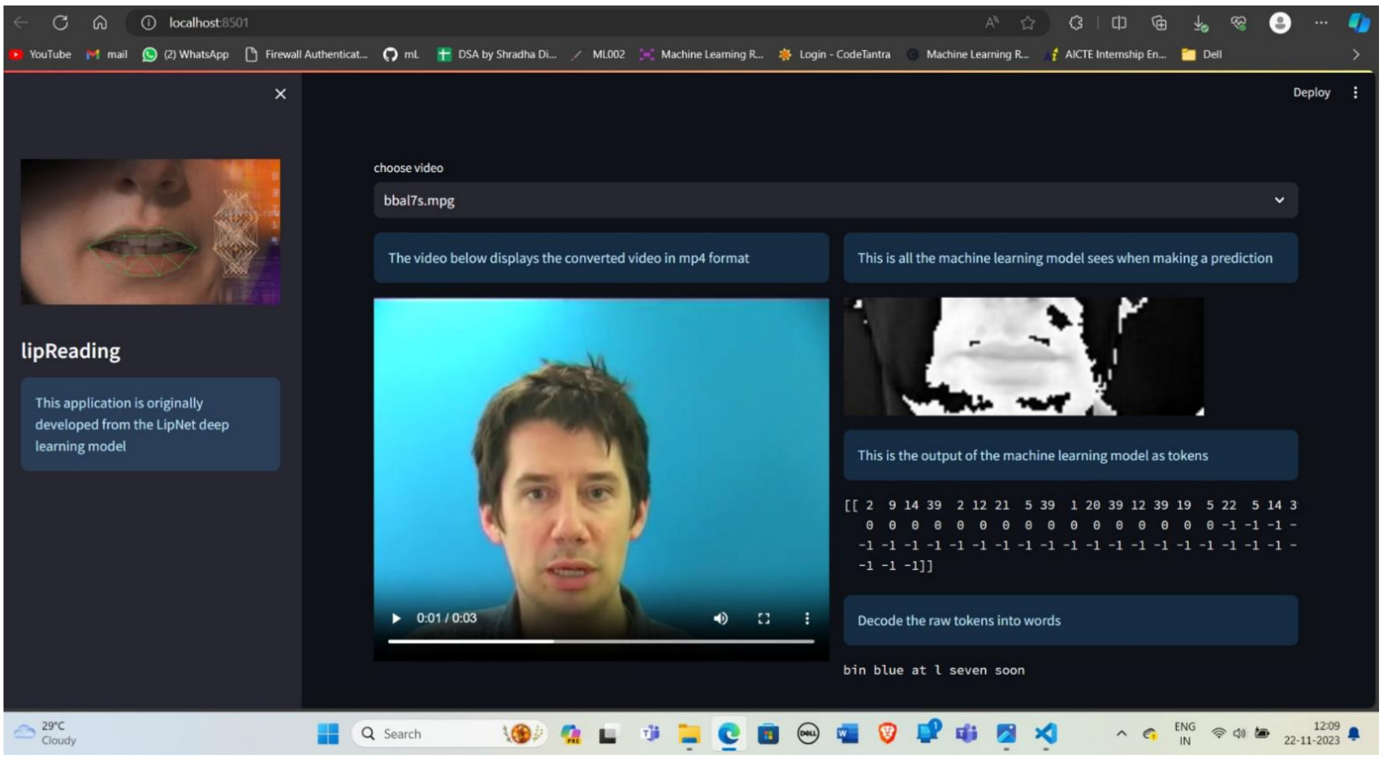
This is all the machine learning model sees when making a prediction

This is the output of the machine learning model as tokens

```
[[ 2  9 14 39  2 12 21  5 39  1 20 39  6 39 20 23 15 39 14 1
   0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 -1 -1 -1 -
  -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -
  -1 -1 -1]]
```

Decode the raw tokens into words

```
bin blue at f two now
```



localhost:8501

lipReading

This application is originally developed from the LipNet deep learning model

choose video

bbal7s.mpg

The video below displays the converted video in mp4 format

This is all the machine learning model sees when making a prediction

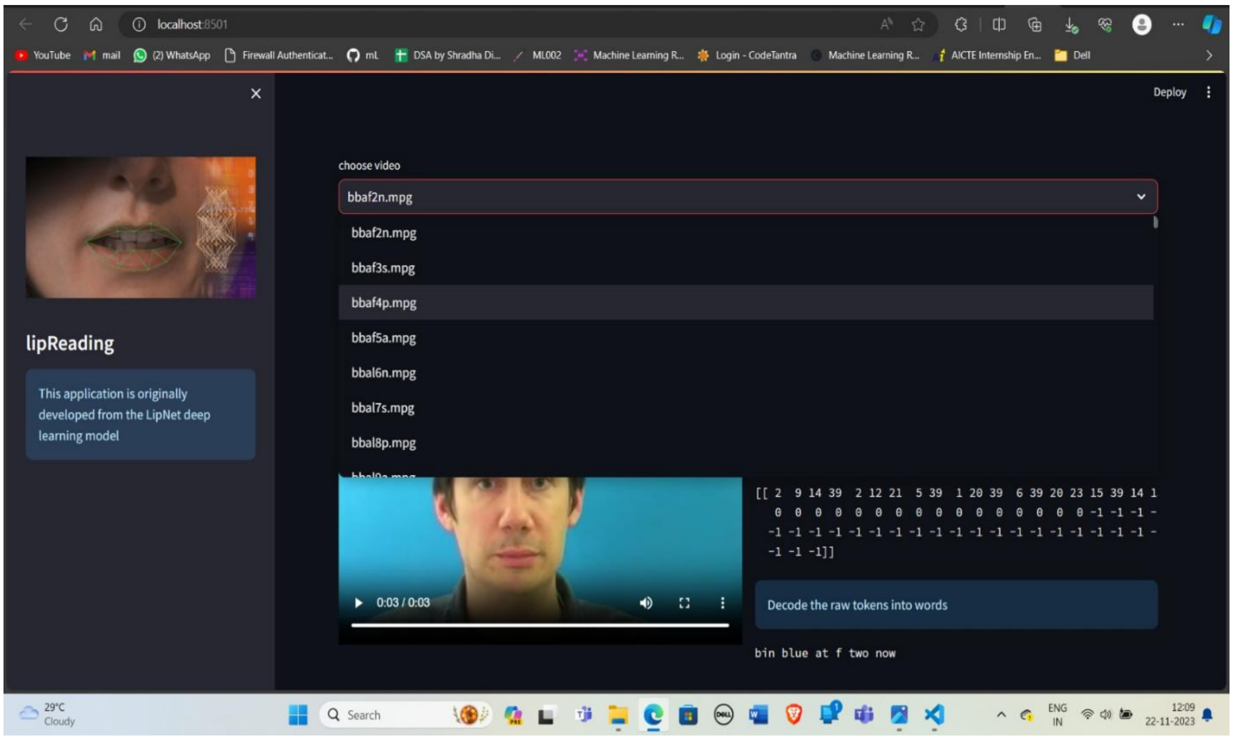
This is the output of the machine learning model as tokens

```
[[ 2  9 14 39  2 12 21  5 39  1 20 39 12 39 19  5 22  5 14 3
   0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 -1 -1 -1 -
  -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -
  -1 -1 -1]]
```

Decode the raw tokens into words

```
bin blue at l seven soon
```

We can select video on website like this:



This application is originally developed from the LipNet deep learning model

```
[[ 2  9 14 39  2 12 21  5 39  1 20 39  6 39 20 23 15 39 14 1
   0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 -1 -1 -1 -
  -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -
  -1 -1 -1]]
```

```
bin blue at f two now
```