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.

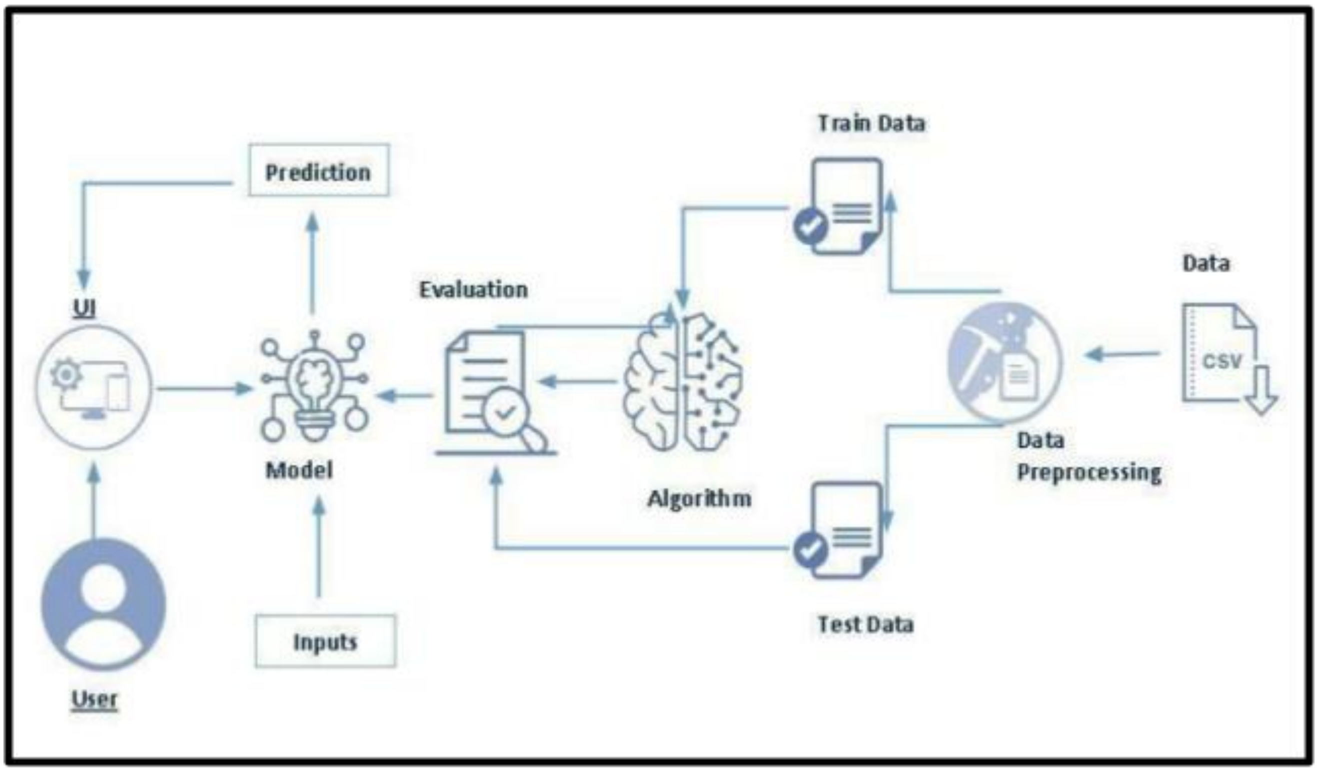
2. **No Need for Audio Data**: Traditional speech recognition models require large amounts of transcribed audio data for training. In contrast, an end-to-end lip reading system can be trained solely on video data, eliminating the need for transcribed audio, which can be expensive and time-consuming to 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
  - Specify the business problem
  - Business requirements
  - Literature Survey.
  - Social or Business Impact.
- Data Collection & Preparation
  - Collect the dataset
  - Data Preparation
- Exploratory Data Analysis
  - Descriptive statistical
  - Visual Analysis
- Model Building
  - Building a model.
  - Training the model.
  - Testing the model
- Model Deployment
  - Save the best model

        - ○Integrate with Web Framework
- Project Demonstration & Documentation
        - 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 :
    - Refer to the link below to download VS Code.
    - Link : https://code.visualstudio.com/download
- Create Project: o
    - Open VS Code. o
    - Create a Folder and name it "Lip_Reading" .
- Machine Learning Concepts
    - Deep Learning:
        - ♦ https://towardsdatascience.com/introducing-deep-learning-and-neural-networks-deep-learning-forr ookies-1-bd68f9cf5883 o
        - NLP Models :
        - ♦ https://medium.com/voice-tech-podcast/an-overview-of-rnn-lstm-gru-79ed642751c6
- Web concepts:
    - 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.

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.

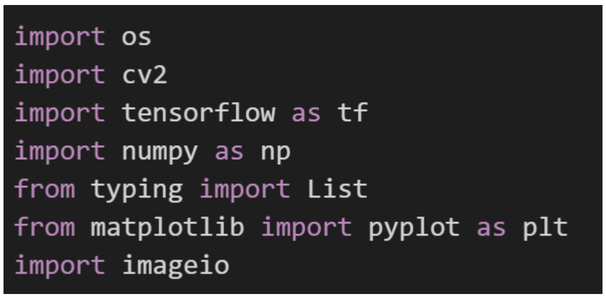
So, this section allows you to acquire the required dataset.

Activity 1: Loading the data.

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

Activity 2: Data Preparation

Import the required packages

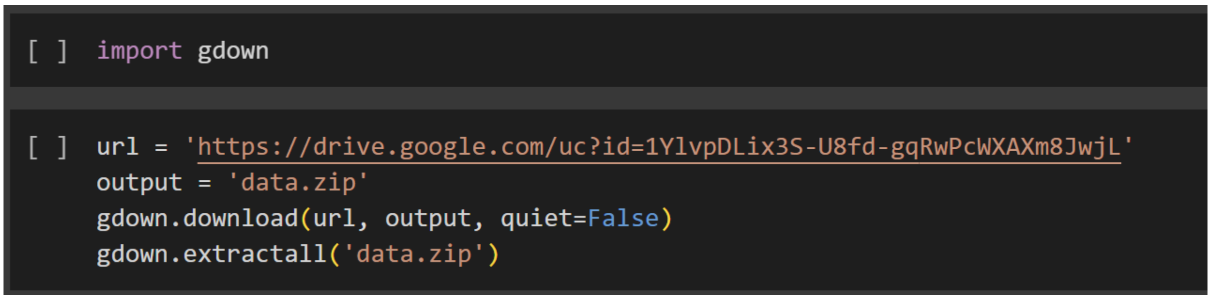


```python
import os
import cv2
import tensorflow as tf
import numpy as np
from typing import List
from matplotlib import pyplot as plt
import imageio
```

We require 5 functions to process the data

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

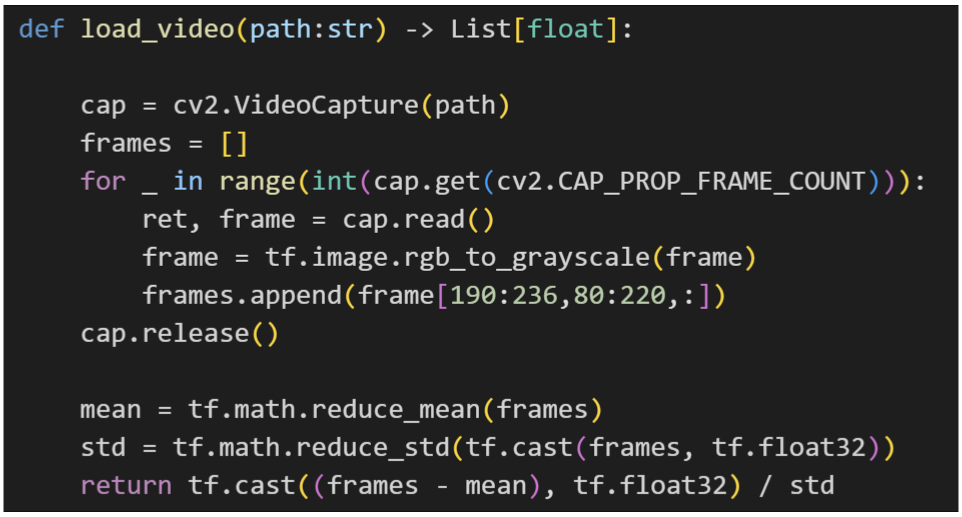
Build Data Loading Functions:



```
[ ]  import gdown
```

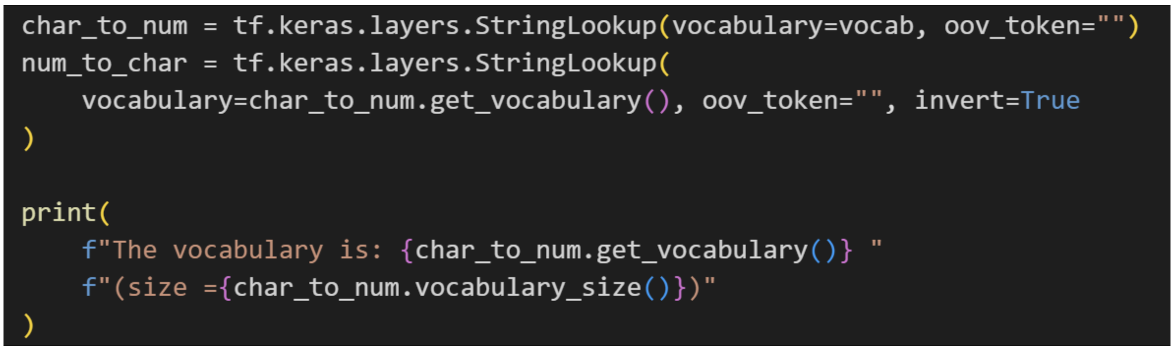
```
[ ]  url = 'https://drive.google.com/uc?id=1YlvpDLix3S-U8fd-gqRwPcWXAXm8JwjL'
     output = 'data.zip'
     gdown.download(url, output, quiet=False)
     gdown.extractall('data.zip')
```

1) load_video():



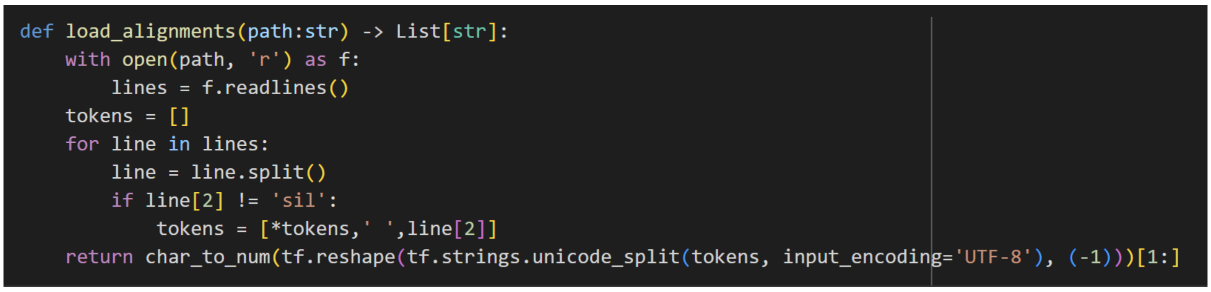
```python
def load_video(path:str) -> List[float]:

    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
```

2) char_to_num and num_to_char



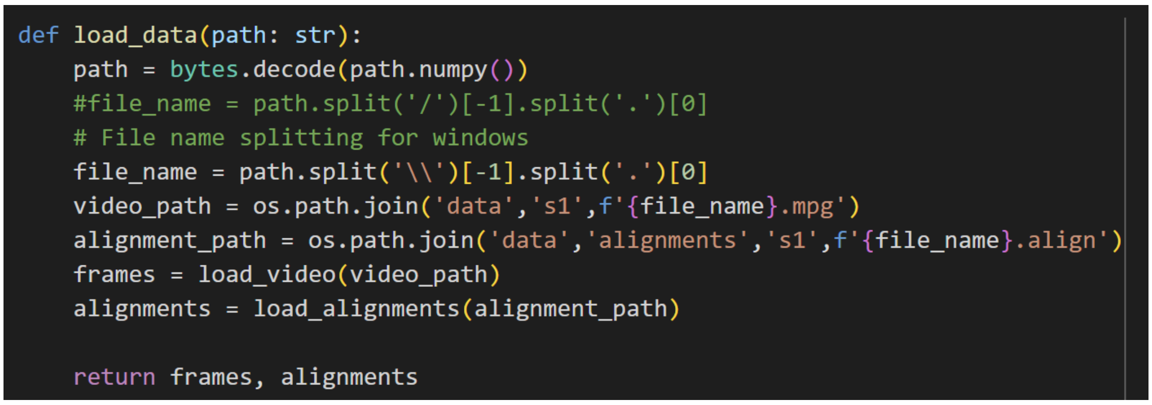
```python
char_to_num = tf.keras.layers.StringLookup(vocabulary=vocab, oov_token="")
num_to_char = tf.keras.layers.StringLookup(
    vocabulary=char_to_num.get_vocabulary(), oov_token="", invert=True
)

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

3) 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:]
```

4)load_data()



```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('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
```

Convert Data to Tensor:

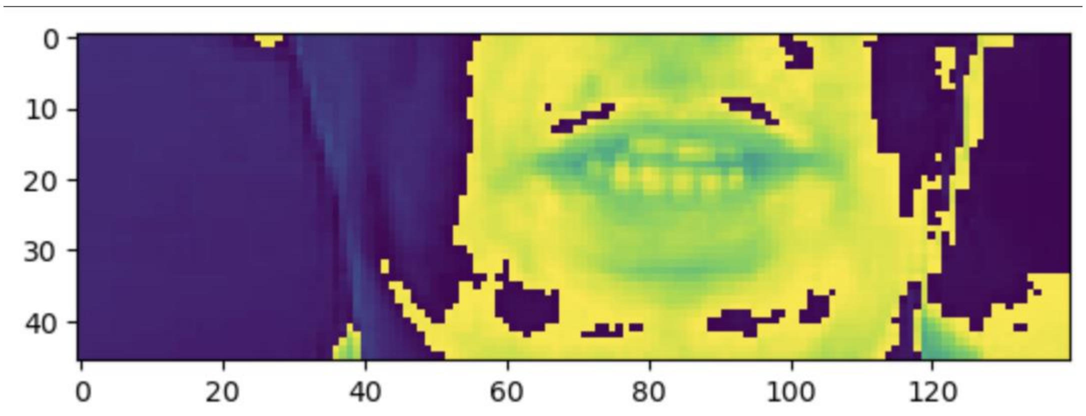
```
test_path = '.\\data\\s1\\bbal6n.mpg'
```

```
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))
```

Output:



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)
```

```
50
```

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

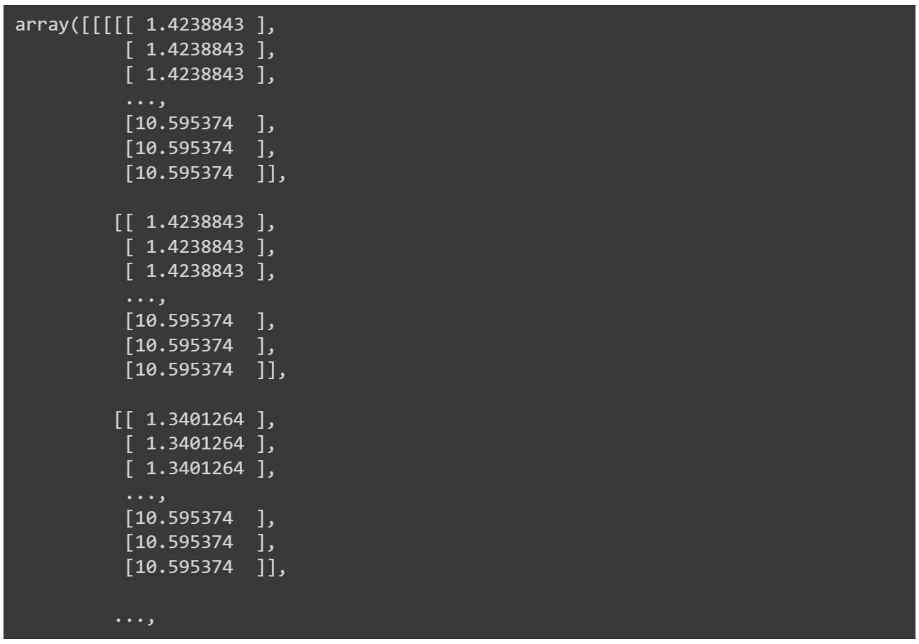
```
len(frames)
```

```
2
```

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

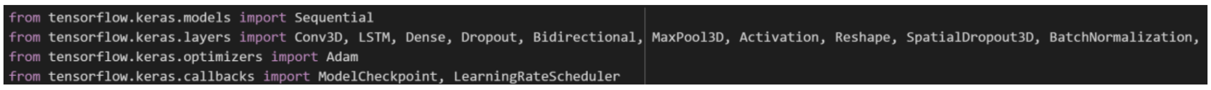
```
val = sample.next(); val[0]
```

Output:

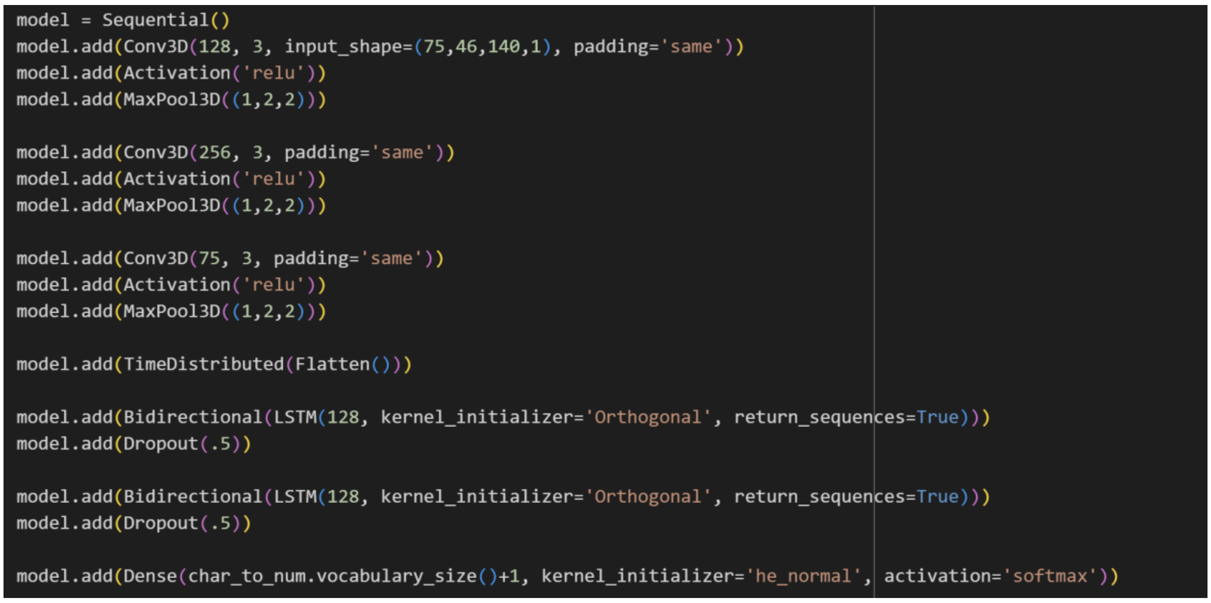


```
array([[[[[ 1.4238843 ],
          [ 1.4238843 ],
          [ 1.4238843 ],
          ...,
          [10.595374  ],
          [10.595374  ],
          [10.595374  ]],

         [[ 1.4238843 ],
          [ 1.4238843 ],
          [ 1.4238843 ],
          ...,
          [10.595374  ],
          [10.595374  ],
          [10.595374  ]],

         [[ 1.3401264 ],
          [ 1.3401264 ],
          [ 1.3401264 ],
          ...,
          [10.595374  ],
          [10.595374  ],
          [10.595374  ]],

         ...,
```

Designing The deep neural network:

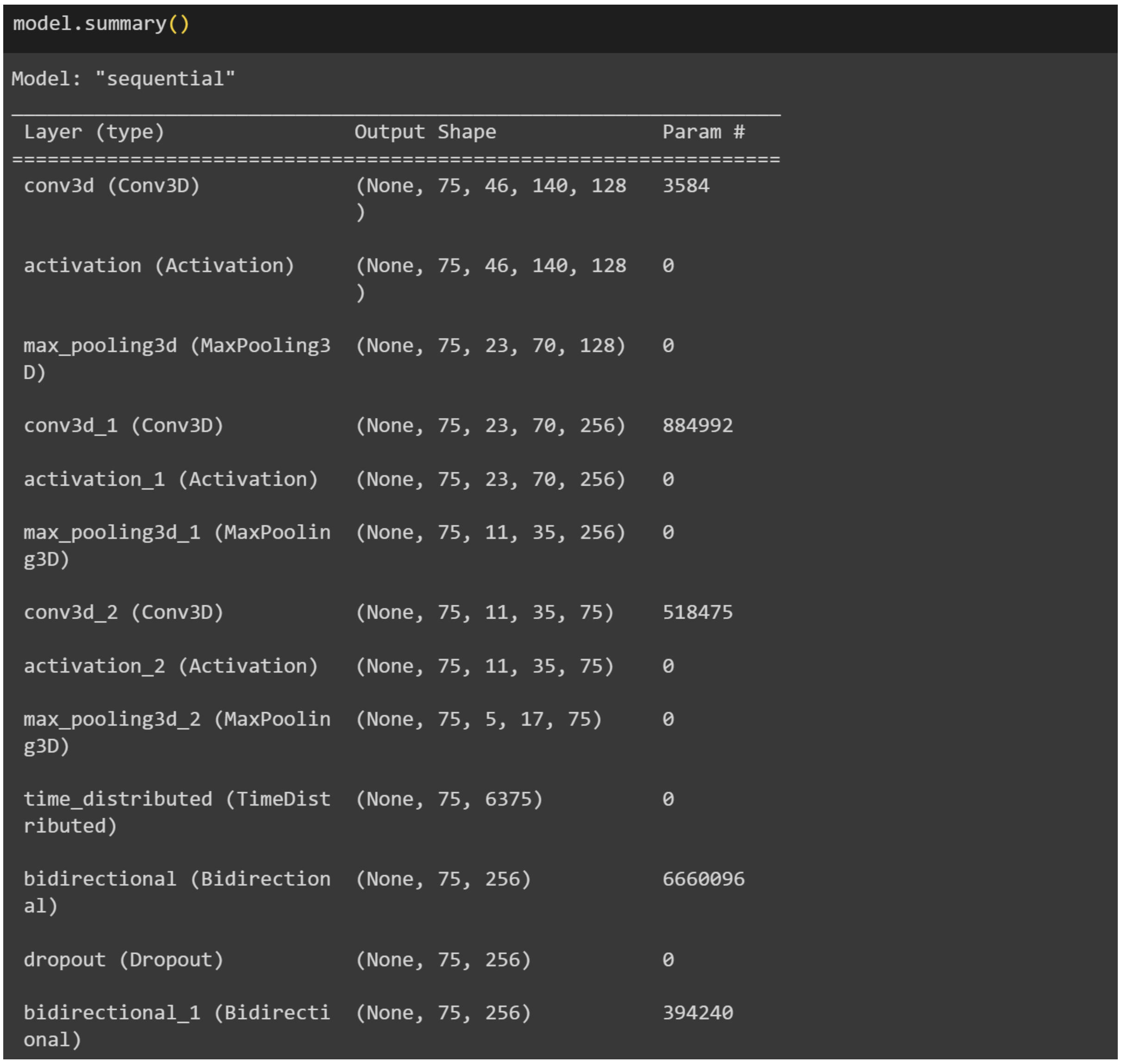


```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv3D, LSTM, Dense, Dropout, Bidirectional, MaxPool3D, Activation, Reshape, SpatialDropout3D, BatchNormalization,
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ModelCheckpoint, LearningRateScheduler
```



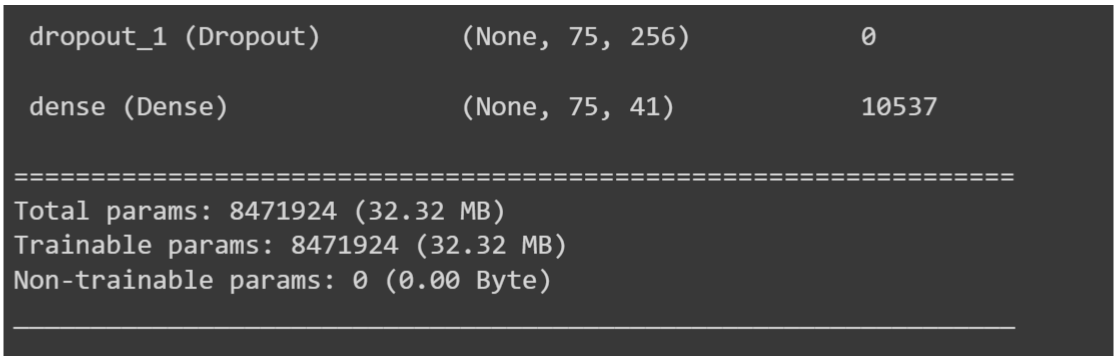
```
model = Sequential()
model.add(Conv3D(128, 3, input_shape=(75,46,140,1), padding='same'))
model.add(Activation('relu'))
model.add(MaxPool3D((1,2,2)))

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:



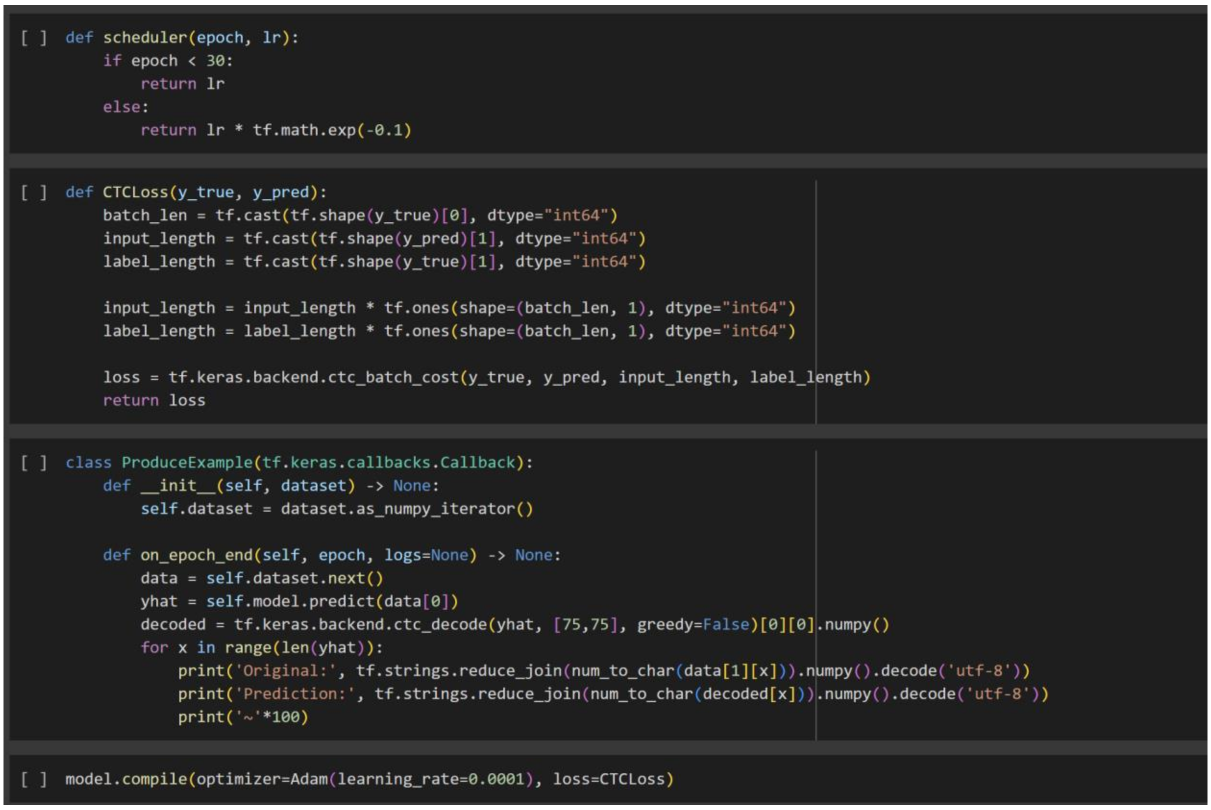
```
model.summary()
```

```
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 (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
 g3D)

 time_distributed (TimeDist  (None, 75, 6375)          0
 ributed)

 bidirectional (Bidirection  (None, 75, 256)           6660096
 al)

 dropout (Dropout)           (None, 75, 256)           0

 bidirectional_1 (Bidirecti  (None, 75, 256)           394240
 onal)
```



```
 dropout_1 (Dropout)         (None, 75, 256)           0

 dense (Dense)               (None, 75, 41)            10537

=================================================================
Total params: 8471924 (32.32 MB)
Trainable params: 8471924 (32.32 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Setup Training Options and Train:

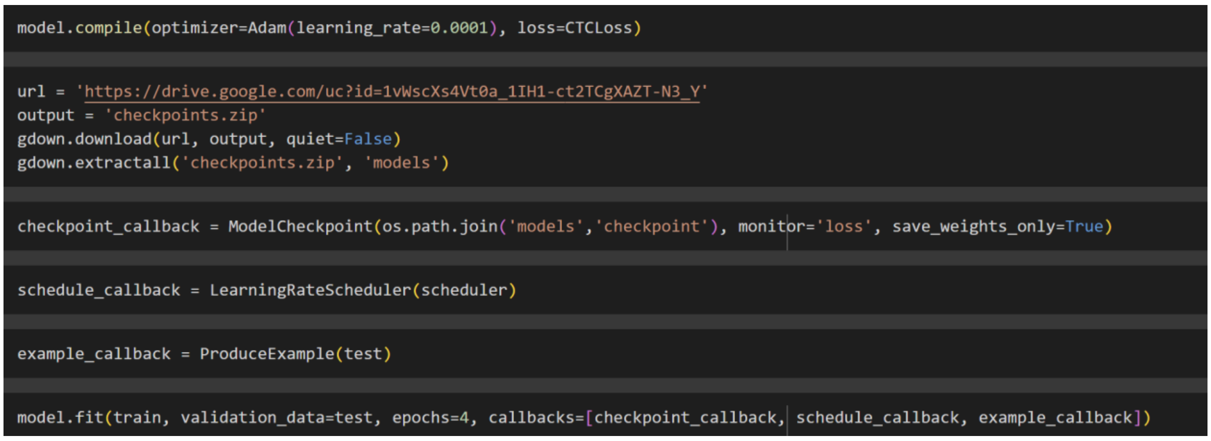


```
[ ] def scheduler(epoch, lr):
        if epoch < 30:
            return lr
        else:
            return lr * tf.math.exp(-0.1)
```

```
[ ] 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)
```



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

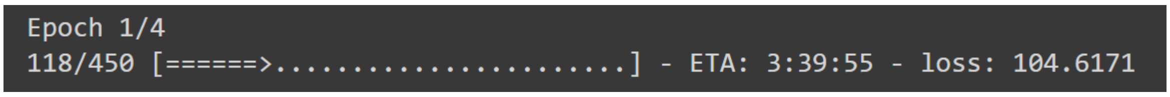
```
url = 'https://drive.google.com/uc?id=1vWscXs4Vt0a_1IH1-ct2TCgXAZT-N3_Y'
output = 'checkpoints.zip'
gdown.download(url, output, quiet=False)
gdown.extractall('checkpoints.zip', 'models')
```

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

```
schedule_callback = LearningRateScheduler(scheduler)
```

```
example_callback = ProduceExample(test)
```

```
model.fit(train, validation_data=test, epochs=4, callbacks=[checkpoint_callback, schedule_callback, example_callback])
```



```
Epoch 1/4
118/450 [======>......................] - ETA: 3:39:55 - loss: 104.6171
```

Predictions:

```
model.load_weights('models/checkpoint')
```

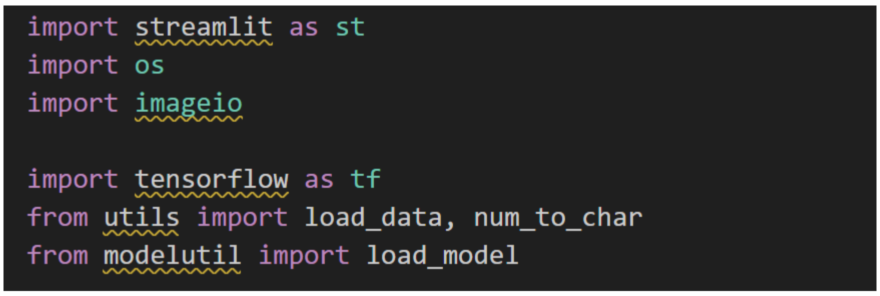
```
test_data = test.as_numpy_iterator()

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]
```

On 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]
```

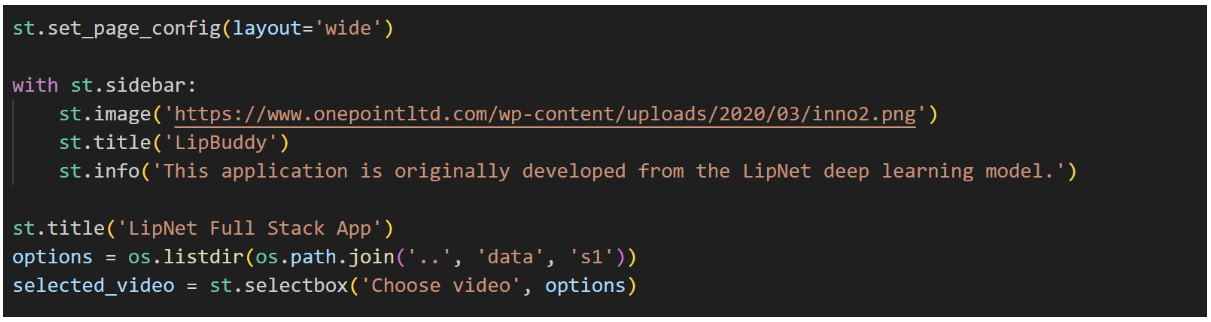
Deployment:

Importing Packages:



```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
```

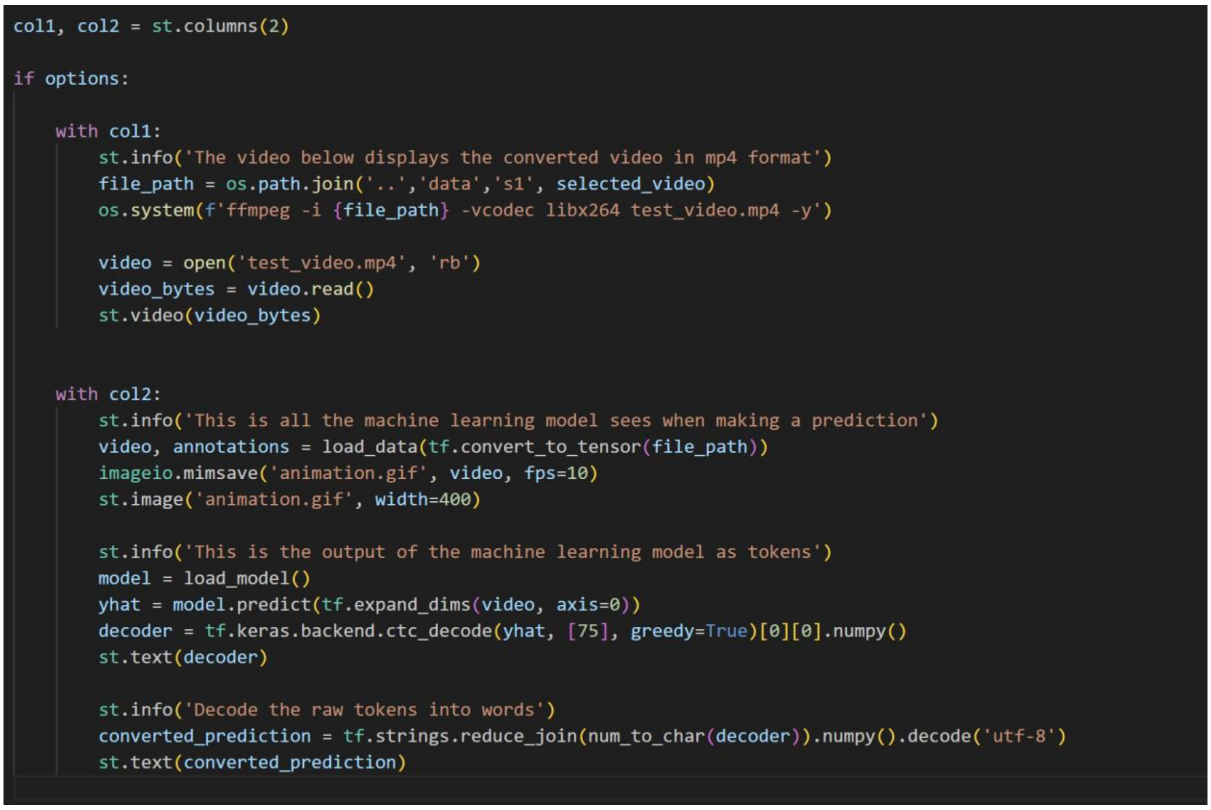
Creating Sidebar:



```python
st.set_page_config(layout='wide')

with st.sidebar:
    st.image('https://www.onepointltd.com/wp-content/uploads/2020/03/inno2.png')
    st.title('LipBuddy')
    st.info('This application is originally developed from the LipNet deep learning model.')

st.title('LipNet Full Stack App')
options = os.listdir(os.path.join('..', 'data', 's1'))
selected_video = st.selectbox('Choose video', options)
```

Creating 2 Columns

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



```python
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('..','data','s1', selected_video)
        os.system(f'ffmpeg -i {file_path} -vcodec libx264 test_video.mp4 -y')

        video = open('test_video.mp4', 'rb')
        video_bytes = video.read()
        st.video(video_bytes)


    with col2:
        st.info('This is all the machine learning model sees when making a prediction')
        video, annotations = load_data(tf.convert_to_tensor(file_path))
        imageio.mimsave('animation.gif', video, fps=10)
        st.image('animation.gif', width=400)

        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)
```