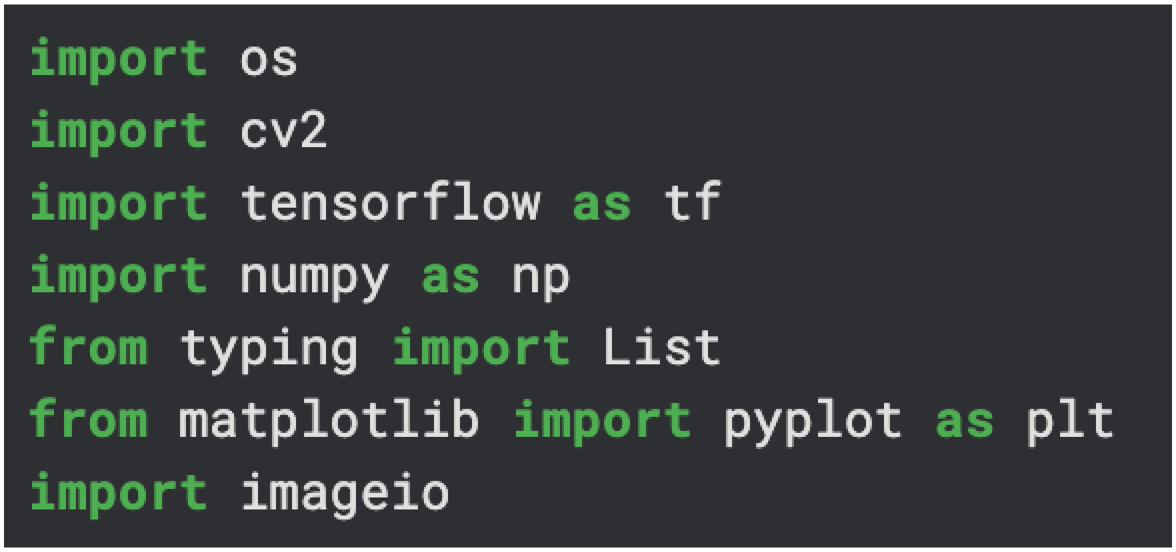
Project Development

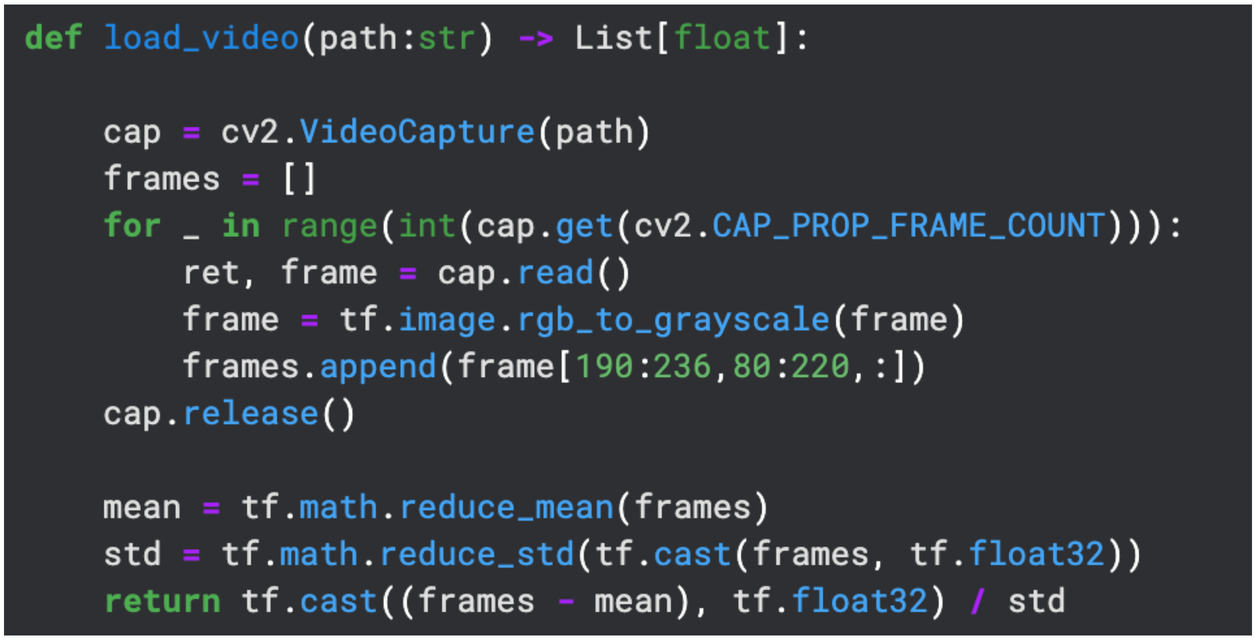
1. importing necessary libraries



```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
```

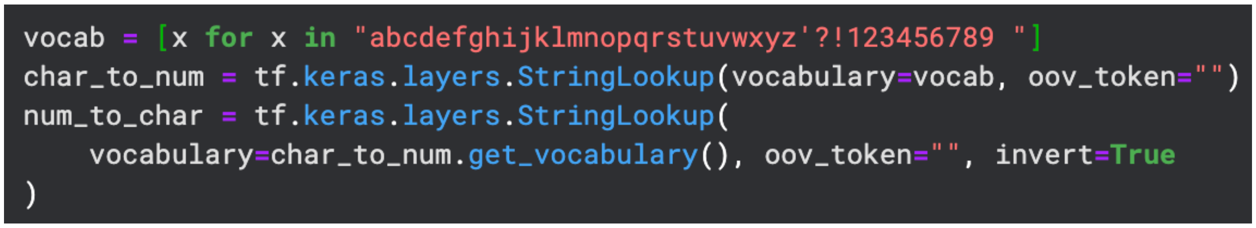
2. Define the required functions:

1. *load_video()*



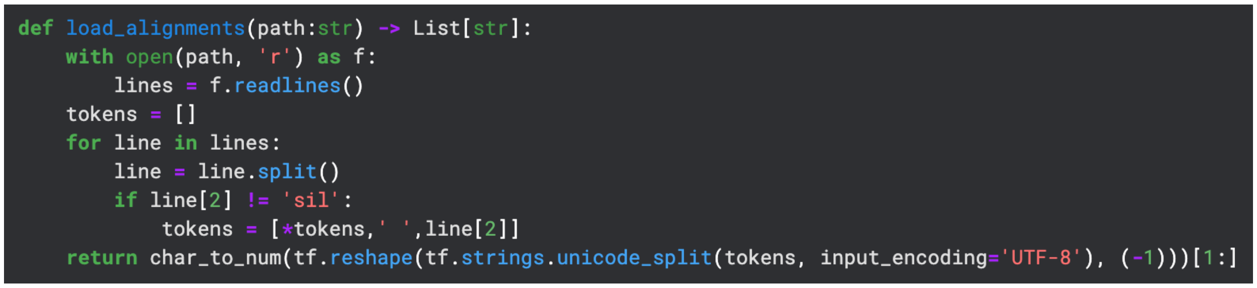
```python
def load_video(path:str) -> List[float]:

    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():*



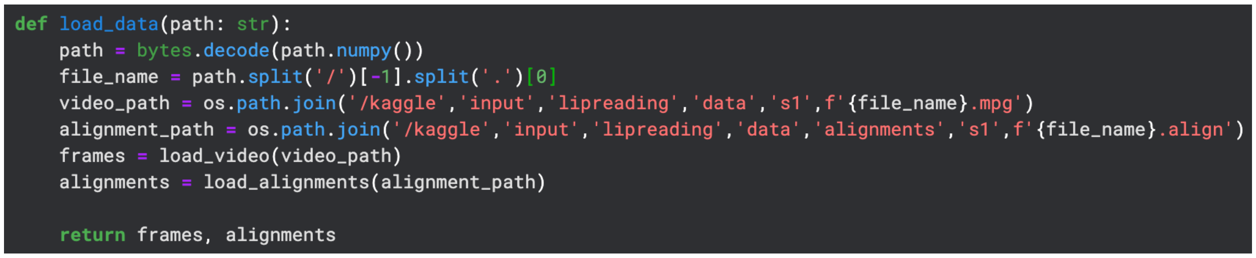
```python
vocab = [x for x in "abcdefghijklmnopqrstuvwxyz'?!123456789 "]
char_to_num = tf.keras.layers.StringLookup(vocabulary=vocab, oov_token="")
num_to_char = tf.keras.layers.StringLookup(
    vocabulary=char_to_num.get_vocabulary(), oov_token="", invert=True
)
```

3. *load_allignments():*



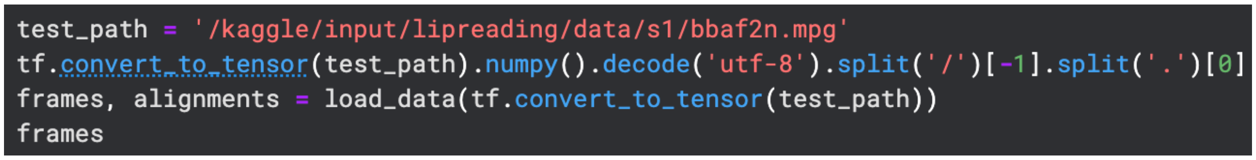
```python
def load_alignments(path:str) -> List[str]:
    with open(path, 'r') as f:
        lines = f.readlines()
    tokens = []
    for line in lines:
        line = line.split()
        if line[2] != 'sil':
            tokens = [*tokens,' ',line[2]]
    return char_to_num(tf.reshape(tf.strings.unicode_split(tokens,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

4. *load_data():*



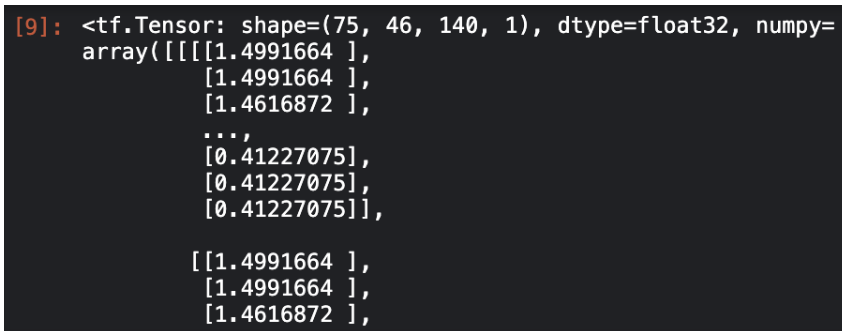
```python
def load_data(path: str):
    path = bytes.decode(path.numpy())
    file_name = path.split('/')[-1].split('.')[0]
    video_path = os.path.join('/kaggle','input','lipreading','data','s1',f'{file_name}.mpg')
    alignment_path = os.path.join('/kaggle','input','lipreading','data','alignments','s1',f'{file_name}.align')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

    return frames, alignments
```

now lets test these functions and convert a test input into tensor:



```python
test_path = '/kaggle/input/lipreading/data/s1/bbaf2n.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))
frames
```

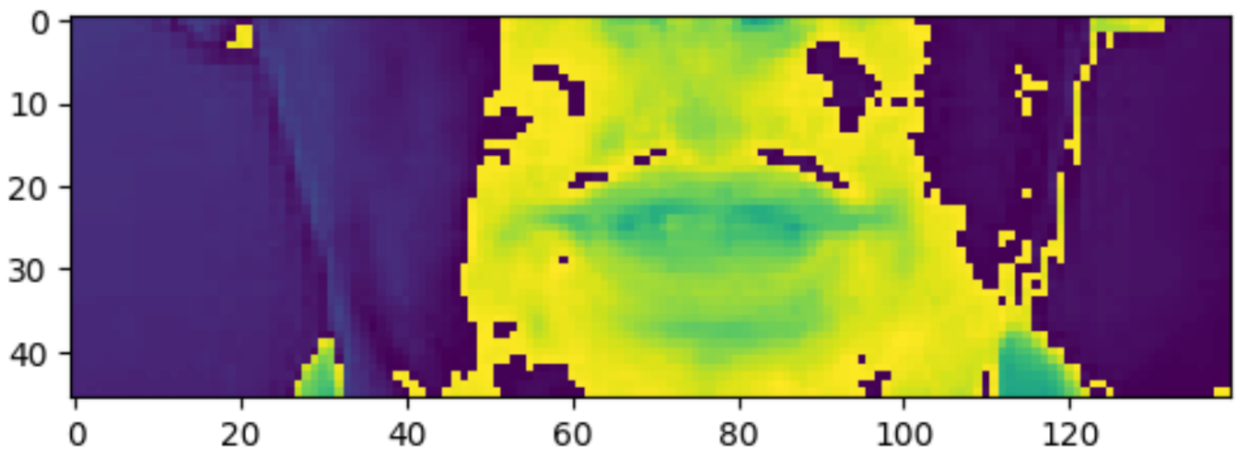
the output will be something like:



```
[9]: <tf.Tensor: shape=(75, 46, 140, 1), dtype=float32, numpy=
     array([[[[1.4991664 ],
              [1.4991664 ],
              [1.4616872 ],
              ...,
              [0.41227075],
              [0.41227075],
              [0.41227075]],

             [[1.4991664 ],
              [1.4991664 ],
              [1.4616872 ],
```

3. visualize a frame of the video:

```python
plt.imshow(frames[35])
```



4. Spliting the data into training and testing

first we will need a mappable function:

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

next we can use the code shown below to split into train and test equally:

```
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 for split
train = data.take(450)
test = data.skip(450)
```

7.5 Building a model

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv3D, LSTM, Dense, Dropout
from tensorflow.keras.layers import Bidirectional, MaxPool3D, Activation, Reshape
from tensorflow.keras.layers import SpatialDropout3D, BatchNormalization
from tensorflow.keras.layers import TimeDistributed, Flatten
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'))
```

5. defining the callback functions, loss function and compiling the model

```
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
```

```python
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)
```

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

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

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

6. training the model:

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

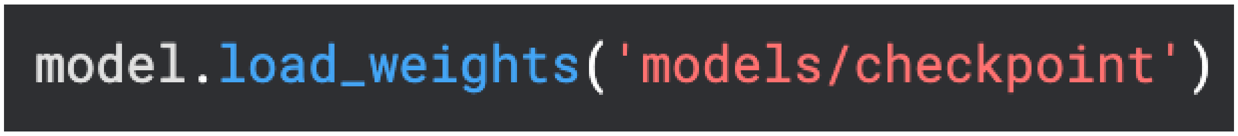
```
Epoch 1/100
450/450 [==============================] - ETA: 0s - loss: 85.5234
[mpeg1video @ 0x793cdc021ec0] ac-tex damaged at 22 17
[mpeg1video @ 0x793cdc021ec0] Warning MVs not available
[mpeg1video @ 0x793cc406d580] ac-tex damaged at 22 17
[mpeg1video @ 0x793cc406d580] Warning MVs not available
1/1 [==============================] - 2s 2s/step
Original: place blue in o six please
Prediction: l e e e e n
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Original: set red with v three again
Prediction: l e e e e e n
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
450/450 [==============================] - 785s 2s/step - loss: 85.5234 - val_loss: 72.2875 - lr: 1.0000e-04
Epoch 2/100
111/450 [======>.......................] - ETA: 4:02 - loss: 72.8648
[mpeg1video @ 0x793cc4057100] ac-tex damaged at 22 17
[mpeg1video @ 0x793cc4057100] Warning MVs not available
450/450 [==============================] - ETA: 0s - loss: 70.7376
[mpeg1video @ 0x7940db18c180] ac-tex damaged at 22 17
[mpeg1video @ 0x7940db18c180] Warning MVs not available
1/1 [==============================] - 0s 122ms/step
Original: bin red at g one again
Prediction: la e e e e o
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Original: bin blue by z eight now
Prediction: la e e e e o
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
450/450 [==============================] - 551s 1s/step - loss: 70.7376 - val_loss: 65.0712 - lr: 1.0000e-04
Epoch 3/100
141/450 [========>.....................] - ETA: 3:39 - loss: 68.1182
[mpeg1video @ 0x794278075100] ac-tex damaged at 22 17
[mpeg1video @ 0x794278075100] Warning MVs not available
450/450 [==============================] - ETA: 0s - loss: 66.5817
[mpeg1video @ 0x793cd883b600] ac-tex damaged at 22 17
[mpeg1video @ 0x793cd883b600] Warning MVs not available
1/1 [==============================] - 0s 117ms/step
Original: lay green in l seven again
Prediction: la e e e e ea
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Original: bin white in t zero please
Prediction: la e e e e ea
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

now wait for the model to get train for upto 100 epoches

PERFORMANCE TESTING:

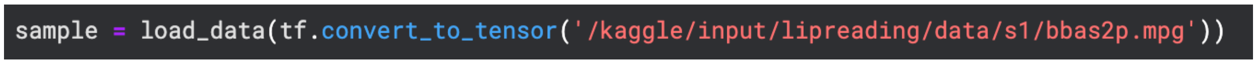
now since we have saved the model at the location "models/checkpoint"

so first load the weights:



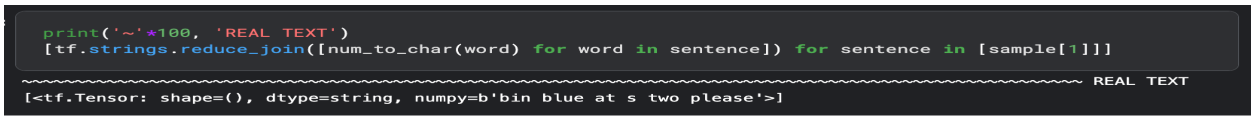
```
model.load_weights('models/checkpoint')
```

now lets load a video to prerdict



```
sample = load_data(tf.convert_to_tensor('/kaggle/input/lipreading/data/s1/bbas2p.mpg'))
```

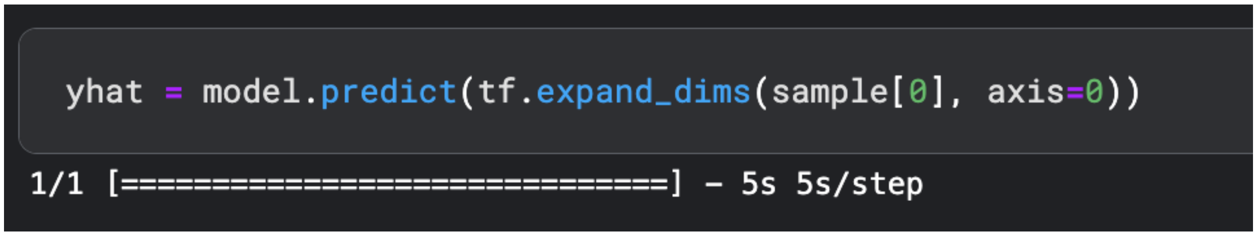
lets see the real text



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

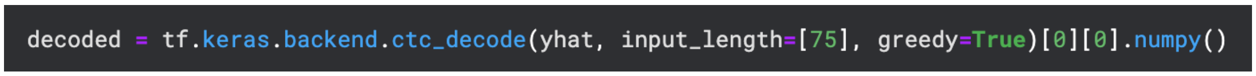
```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ REAL TEXT
[<tf.Tensor: shape=(), dtype=string, numpy=b'bin blue at s two please'>]
```

Now lets predict using the model

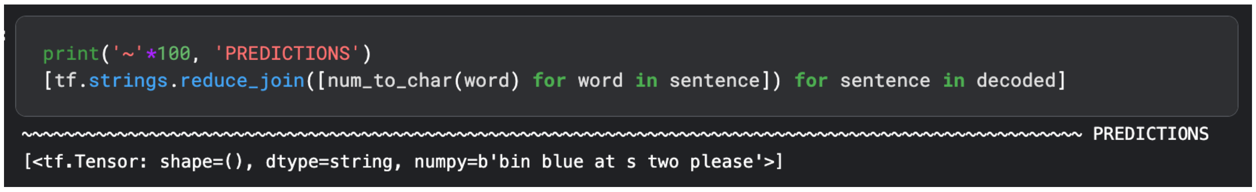


```
yhat = model.predict(tf.expand_dims(sample[0], axis=0))
```

```
1/1 [==============================] - 5s 5s/step
```



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



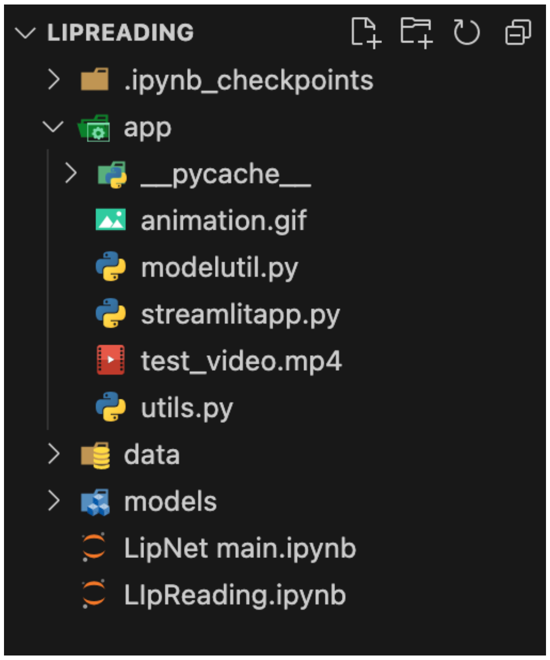
```
print('~'*100, 'PREDICTIONS')
[tf.strings.reduce_join([num_to_char(word) for word in sentence]) for sentence in decoded]
```

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

we can see that the predicted text matches exactly with the real text.

Integration with the Web App

the file structure should look something like this:

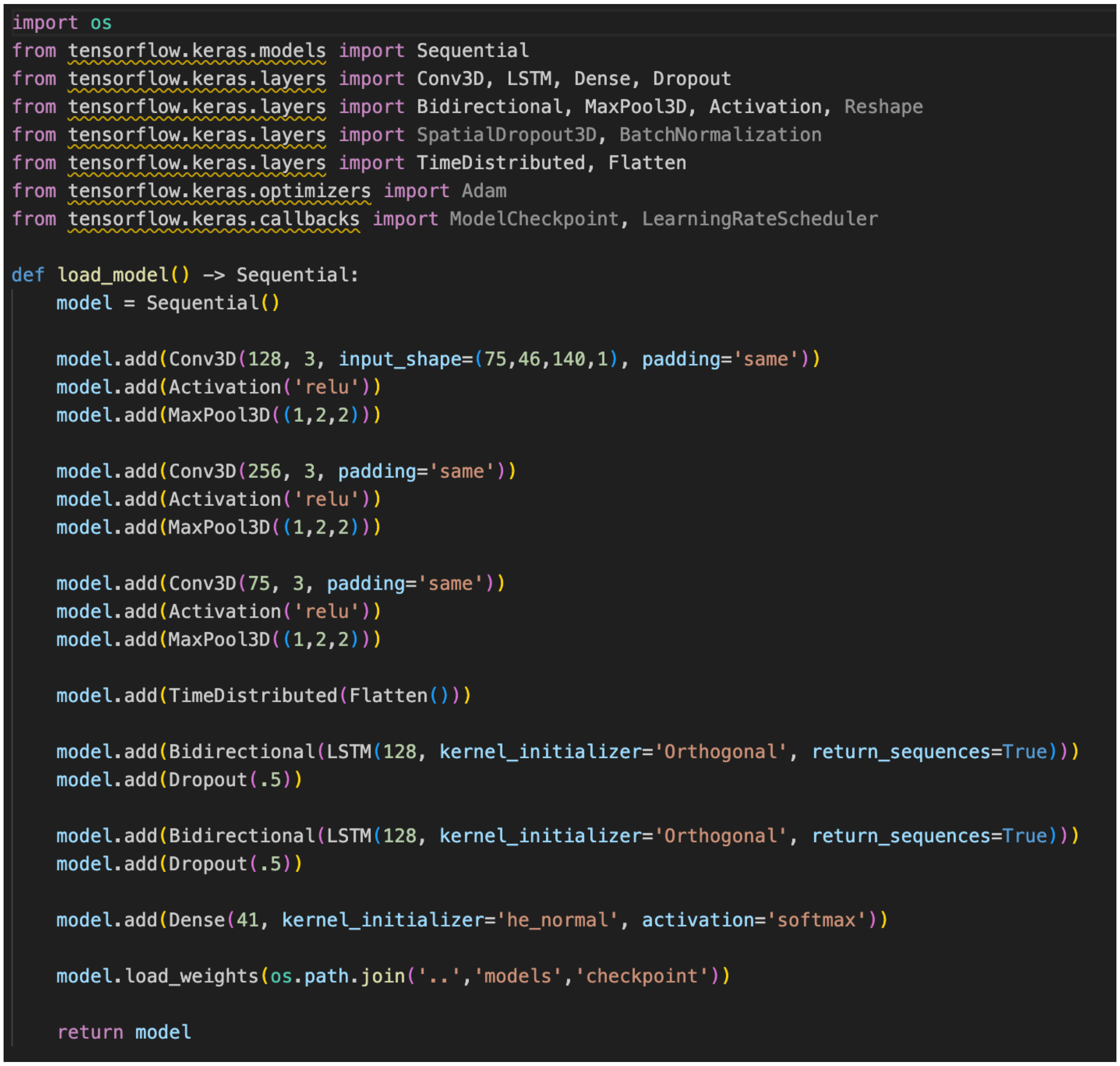


```
LIPREADING
  > .ipynb_checkpoints
  v app
    > __pycache__
      animation.gif
      modelutil.py
      streamlitapp.py
      test_video.mp4
      utils.py
  > data
  > models
    LipNet main.ipynb
    LIpReading.ipynb
```

As you can see from the file structure above we need to create thrree python files inside app folder

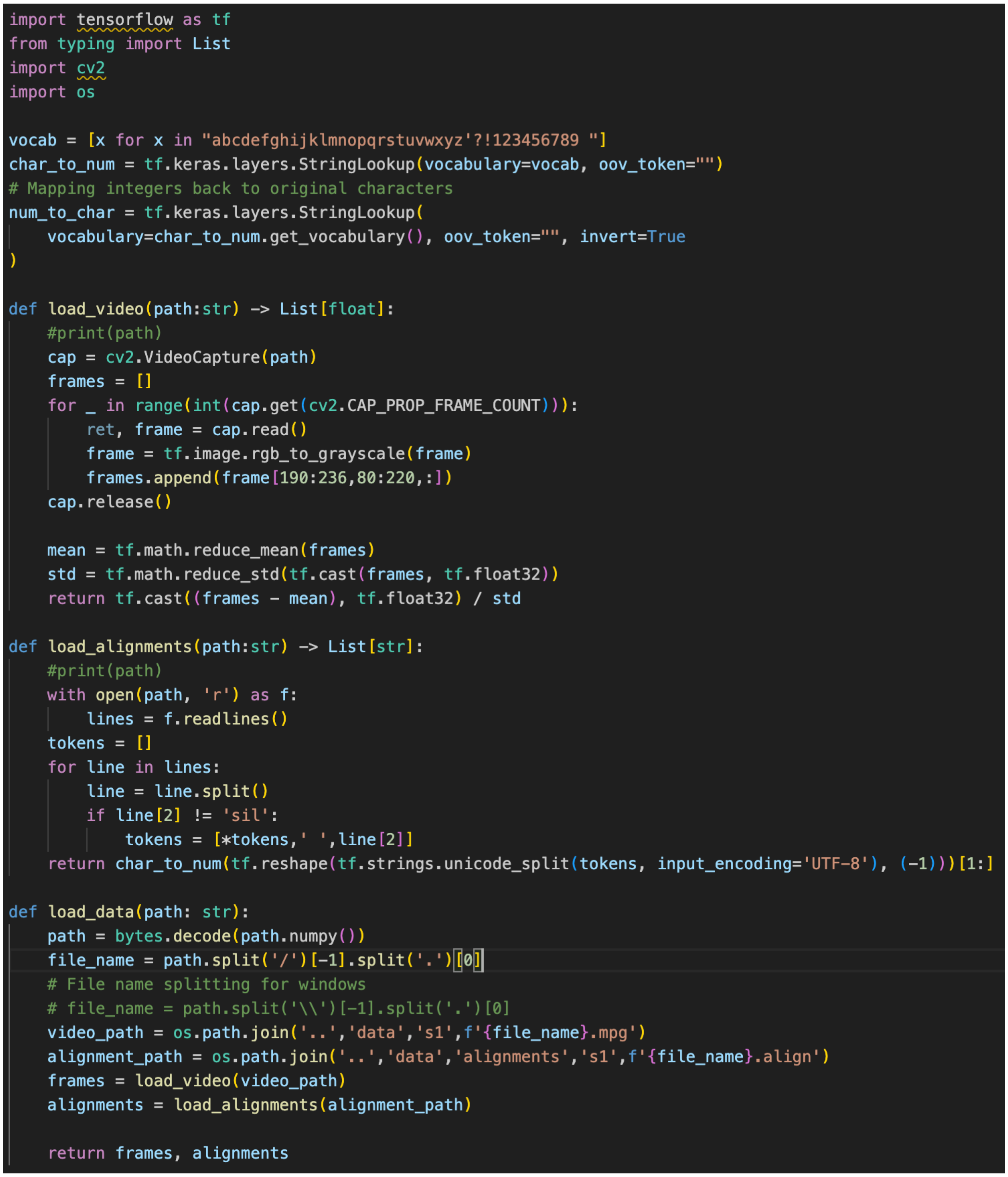
1. modelutil.py
2. utils.py
3. streamlitapp.py

Inside of modelutil.py file we will need to add the model:



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

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('..','models','checkpoint'))

    return model
```

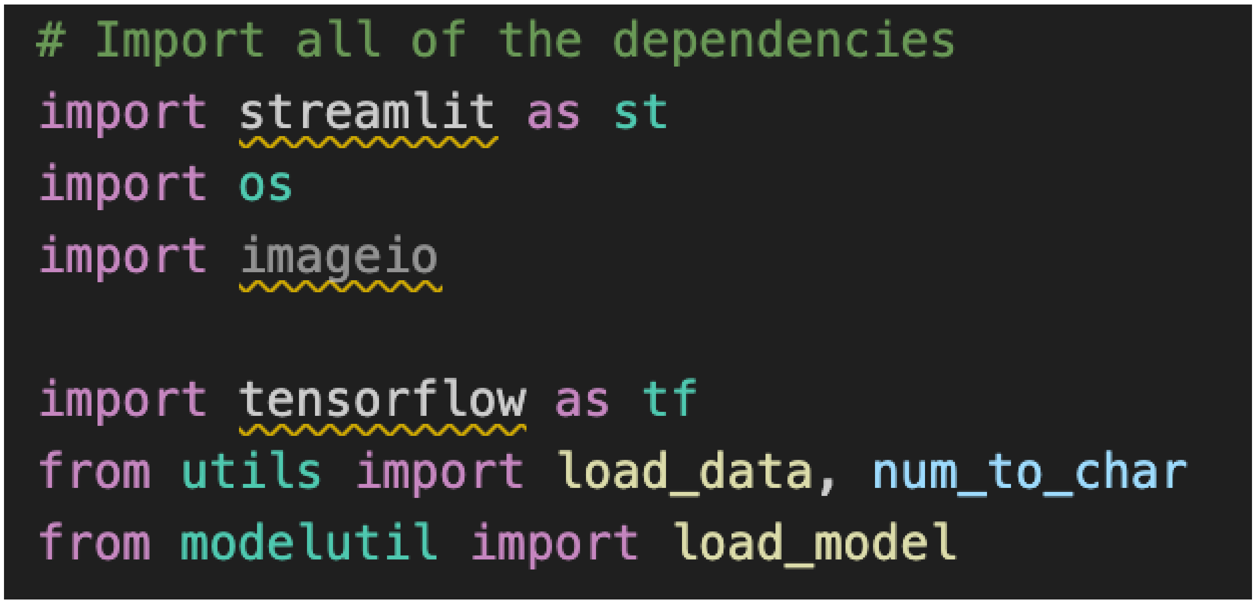
Inside of modelutil.py file we will need to add all the user defined functions which are char_to_num, num_to_char, load_video(), load_alignments(), load_data() functions:



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

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:]

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
```

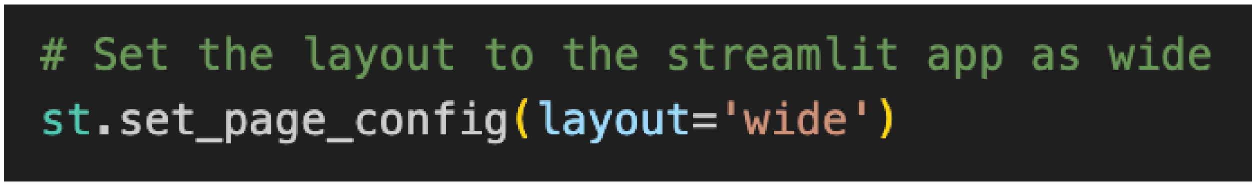
Inside streamlitapp.py we will add the logic to host our app on the internet:

first let's import all the necessary libraries:



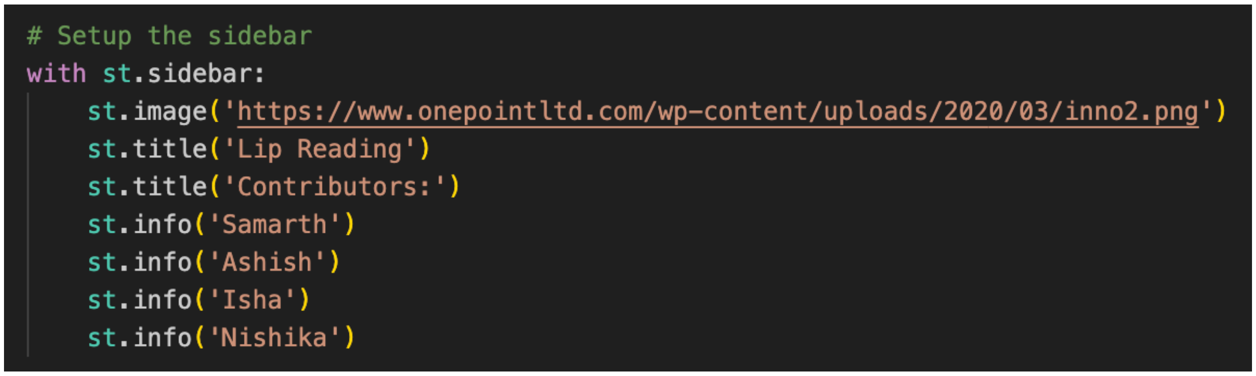
```python
# Import all of the dependencies
import streamlit as st
import os
import imageio

import tensorflow as tf
from utils import load_data, num_to_char
from modelutil import load_model
```

Next set the layout of the web app



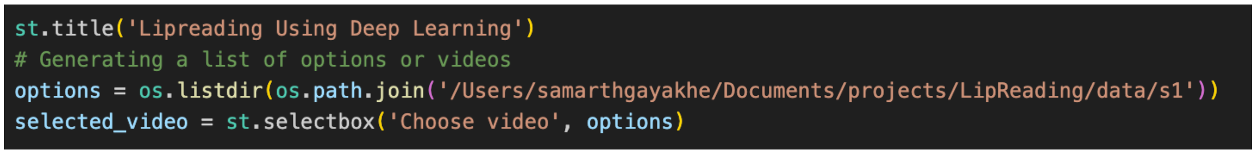
```python
# Set the layout to the streamlit app as wide
st.set_page_config(layout='wide')
```

let's cofigure the sidebar now:



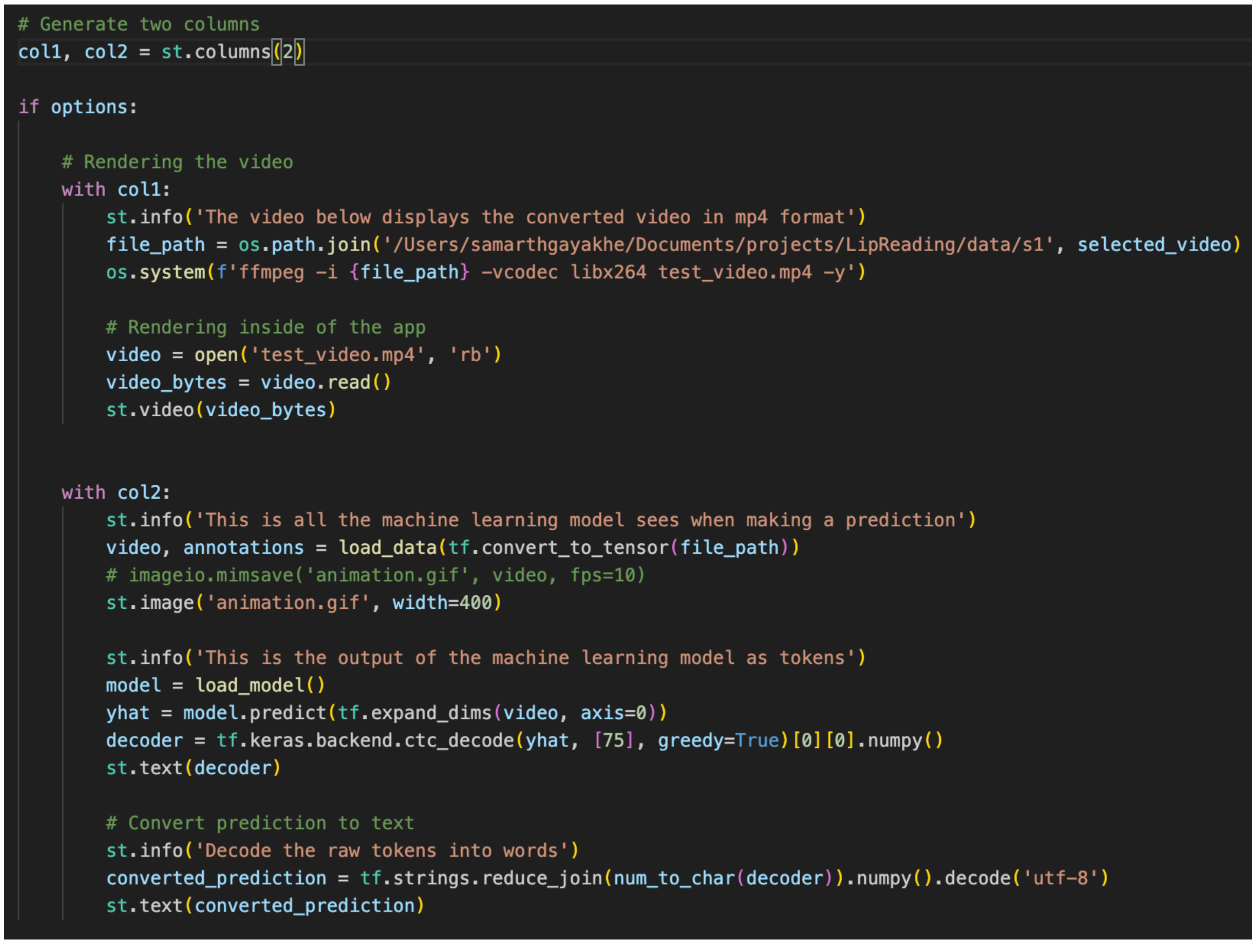
```python
# Setup the sidebar
with st.sidebar:
    st.image('https://www.onepointltd.com/wp-content/uploads/2020/03/inno2.png')
    st.title('Lip Reading')
    st.title('Contributors:')
    st.info('Samarth')
    st.info('Ashish')
    st.info('Isha')
    st.info('Nishika')
```

Now add a little and create a drop down menu for choosing the input:



```python
st.title('Lipreading Using Deep Learning')
# Generating a list of options or videos
options = os.listdir(os.path.join('/Users/samarthgayakhe/Documents/projects/LipReading/data/s1'))
selected_video = st.selectbox('Choose video', options)
```

Now create two columns and the configure it:



```python
# Generate two columns
col1, col2 = st.columns(2)

if options:

    # Rendering the video
    with col1:
        st.info('The video below displays the converted video in mp4 format')
        file_path = os.path.join('/Users/samarthgayakhe/Documents/projects/LipReading/data/s1', selected_video)
        os.system(f'ffmpeg -i {file_path} -vcodec libx264 test_video.mp4 -y')

        # Rendering inside of the app
        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)

        # Convert prediction to text
        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)
```

col1 will display the selected video and play it.

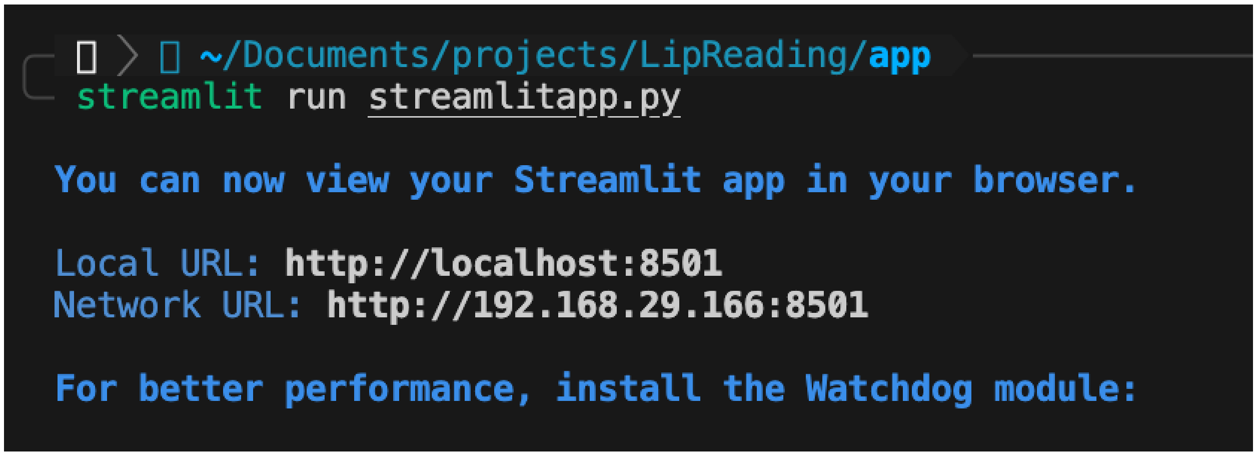
col2 will display :

➔ animation of lips.

➔ raw output of your model

➔ result prediction

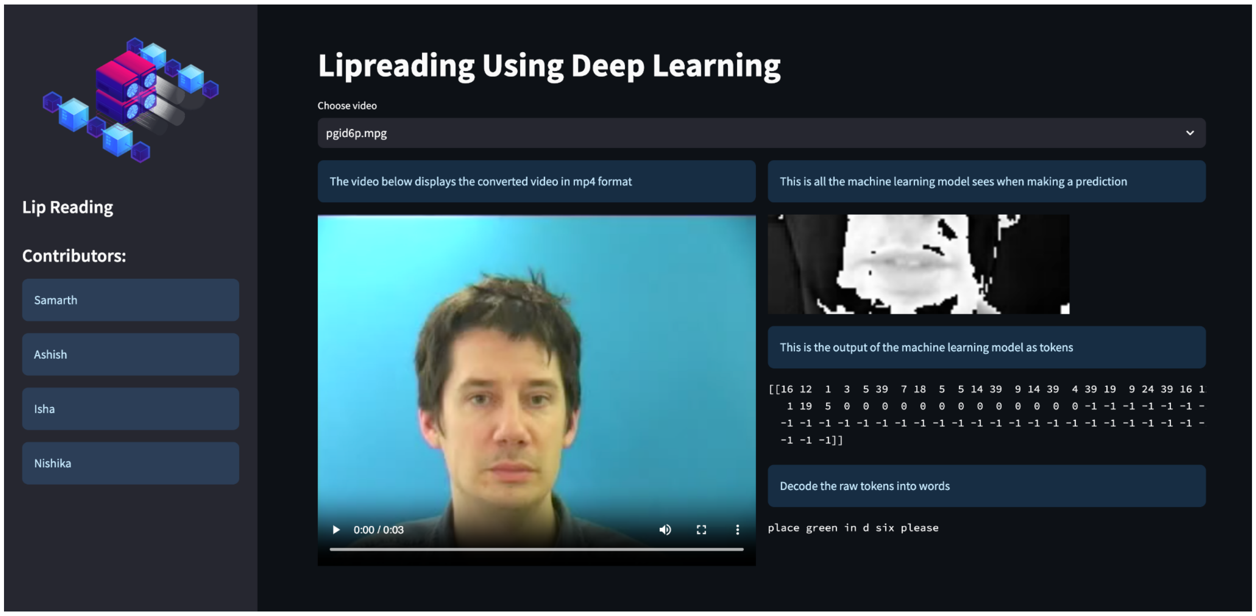
Now you can host the webapp by executing the following command in the terminal:



```
~/Documents/projects/LipReading/app
streamlit run streamlitapp.py

  You can now view your Streamlit app in your browser.

  Local URL: http://localhost:8501
  Network URL: http://192.168.29.166:8501

  For better performance, install the Watchdog module:
```

and my clicking on the link we can launch the web app

the web app will look something like this:



Lipreading Using Deep Learning

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

```
[[16 12  1  3  5 39  7 18  5  5 14 39  9 14 39  4 39 19  9 24 39 16 1
   1 19  5  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 -1 -1 -1 -
  -1 -1 -1]]
```

Decode the raw tokens into words

```
place green in d six please
```