Install and Import Dependencies

```
!pip list
```

Show hidden output

```
!pip install opencv-python matplotlib imageio gdown tensorflow
```

Show hidden output

```
!pip install tensorflow==2.10
```

Show hidden output

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

```
tf.config.list_physical_devices('GPU')
```

```
[]
```

```
physical_devices = tf.config.list_physical_devices('GPU')
try:
    tf.config.experimental.set_memory_growth(physical_devices[0], True)
except:
    pass
```

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')
```

Show hidden output

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

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

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

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

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

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

```
The vocabulary is: ['', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', '
```

```
char_to_num.get_vocabulary()
```

Show hidden output

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

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

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

```
<tf.Tensor: shape=(4,), dtype=string, numpy=array([b'n', b'i', b'c', b'k'], dtype=object)>
```

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

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

    return frames, alignments
```

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

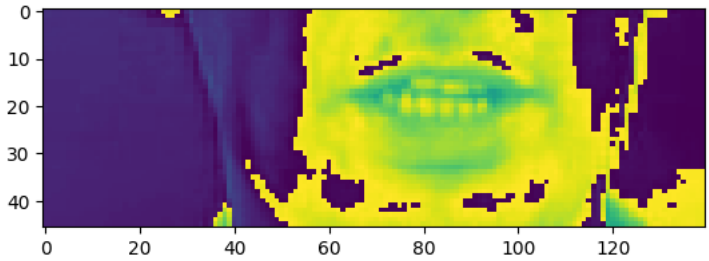
```
tf.convert_to_tensor(test_path).numpy().decode('utf-8').split('/')[-1].split('.')[0]
```

```
'bbal6n'
```

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

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

```
<matplotlib.image.AxesImage at 0x7a0d0d0deb60>
```




```
alignments
```

```
<tf.Tensor: shape=(21,), dtype=int64, numpy=
array([ 2,  9, 14, 39,  2, 12, 21,  5, 39,  1, 20, 39, 12, 39, 19,  9, 24,
       39, 14, 15, 23])>
```

```
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
```

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
```

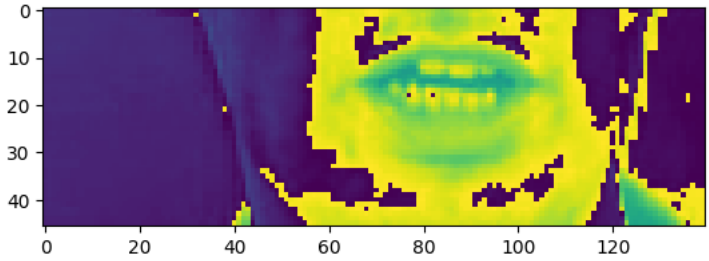
```
sample = data.as_numpy_iterator()
```

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

Show hidden output

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

```
<matplotlib.image.AxesImage at 0x7a0cefff4610>
```




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

```
<tf.Tensor: shape=(), dtype=string, numpy=b'place blue at v five again'>
```

Design the Deep Neural Network

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

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

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

```
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()
```

Show hidden output

```
model.input_shape
```

```
(None, 75, 46, 140, 1)
```

```
model.output_shape
```

```
(None, 75, 41)
```

Setup Training Options and Train

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


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

    input_length = input_length * tf.ones(shape=(batch_len, 1), dtype="int64")
    label_length = label_length * tf.ones(shape=(batch_len, 1), dtype="int64")

    loss = tf.keras.backend.ctc_batch_cost(y_true, y_pred, input_length, label_length)
    return loss


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

    def on_epoch_end(self, epoch, logs=None) -> None:
        data = self.dataset.next()
        yhat = self.model.predict(data[0])
        decoded = tf.keras.backend.ctc_decode(yhat, [75,75], greedy=False)[0][0].numpy()
        for x in range(len(yhat)):
            print('Original:', tf.strings.reduce_join(num_to_char(data[1][x])).numpy().decode('utf-8'))
            print('Prediction:', tf.strings.reduce_join(num_to_char(decoded[x])).numpy().decode('utf-8'))
            print('~'*100)


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


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


schedule_callback = LearningRateScheduler(scheduler)


example_callback = ProduceExample(test)


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

Make a Prediction

```
import gdown
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')
```

```
Downloading...
From (original): https://drive.google.com/uc?id=1vWscXs4Vt0a_1IH1-ct2TCgXAZT-N3_Y
From (redirected): https://drive.google.com/uc?id=1vWscXs4Vt0a_1IH1-ct2TCgXAZT-N3_Y&confirm=t&uuid=8980afbd-5c60-4775-b2
To: /content/checkpoints.zip
100%|██████████| 94.5M/94.5M [00:02<00:00, 32.8MB/s]
['models/checkpoint.index',
 'models/__MACOSX/._checkpoint.index',
 'models/checkpoint.data-00000-of-00001',
 'models/__MACOSX/._checkpoint.data-00000-of-00001',
 'models/checkpoint',
 'models/__MACOSX/._checkpoint']
```

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

```
<tensorflow.python.checkpoint.checkpoint.CheckpointLoadStatus at 0x7a0d0134cf10>
```

```
# model.save('lip_reading_96_epochs.keras')
```

Double-click (or enter) to edit

```
# new_model = tf.keras.models.load_model(
#     'lip_reading_96_epochs.keras',
#     custom_objects={'CTCLoss': CTCLoss}  # Replace with actual CTCLoss if available
# )
```

```
# new_model.summary()
```

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

```
sample = data.next()
```

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

```
1/1 [==============================] - 9s 9s/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 white at n one again'>,
 <tf.Tensor: shape=(), dtype=string, numpy=b'lay red in k three soon'>]
```

```
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 white at n one again'>,
 <tf.Tensor: shape=(), dtype=string, numpy=b'lay red in three soon'>]
```

Test on a Video

```
input_video = '/content/data/s1/bbaf2n.mpg'  # replace with your file path
output_video = '/content/converted_video.mp4'

!ffmpeg -i "{input_video}" -c:v libx264 "{output_video}"
```

```
ffmpeg version 4.4.2-0ubuntu0.22.04.1 Copyright (c) 2000-2021 the FFmpeg developers
  built with gcc 11 (Ubuntu 11.2.0-19ubuntu1)
  configuration: --prefix=/usr --extra-version=0ubuntu0.22.04.1 --toolchain=hardened --libdir=/usr/lib/x86_64-linux-gnu
  libavutil      56. 70.100 / 56. 70.100
  libavcodec     58.134.100 / 58.134.100
  libavformat    58. 76.100 / 58. 76.100
  libavdevice    58. 13.100 / 58. 13.100
  libavfilter     7.110.100 /  7.110.100
  libswscale      5.  9.100 /  5.  9.100
  libswresample   3.  9.100 /  3.  9.100
  libpostproc    55.  9.100 / 55.  9.100
Input #0, mpeg, from '/content/data/s1/bbaf2n.mpg':
  Duration: 00:00:03.00, start: 0.000000, bitrate: 1206 kb/s
  Stream #0:0[0x1e0]: Video: mpeg1video, yuv420p(tv), 360x288 [SAR 1:1 DAR 5:4], 104857 kb/s, 25 fps, 25 tbr, 90k tbn, 2
  Stream #0:1[0x1c0]: Audio: mp2, 44100 Hz, stereo, s16p, 224 kb/s
Stream mapping:
```

```
  Stream #0:0 -> #0:0 (mpeg1video (native) -> h264 (libx264))
  Stream #0:1 -> #0:1 (mp2 (native) -> aac (native))
Press [q] to stop, [?] for help
[libx264 @ 0x59ebb7bb9680] using SAR=1/1
[libx264 @ 0x59ebb7bb9680] using cpu capabilities: MMX2 SSE2Fast SSSE3 SSE4.2 AVX FMA3 BMI2 AVX2 AVX512
[libx264 @ 0x59ebb7bb9680] profile High, level 2.1, 4:2:0, 8-bit
[libx264 @ 0x59ebb7bb9680] 264 - core 163 r3060 5db6aa6 - H.264/MPEG-4 AVC codec - Copyleft 2003-2021 - http://www.video
Output #0, mp4, to '/content/converted_video.mp4':
  Metadata:
    encoder         : Lavf58.76.100
  Stream #0:0: Video: h264 (avc1 / 0x31637661), yuv420p(tv, progressive), 360x288 [SAR 1:1 DAR 5:4], q=2-31, 25 fps, 128
    Metadata:
      encoder         : Lavc58.134.100 libx264
    Side data:
      cpb: bitrate max/min/avg: 0/0/0 buffer size: 0 vbv_delay: N/A
  Stream #0:1: Audio: aac (LC) (mp4a / 0x6134706D), 44100 Hz, stereo, fltp, 128 kb/s
    Metadata:
      encoder         : Lavc58.134.100 aac
frame=   75 fps=0.0 q=-1.0 Lsize=     112kB time=00:00:02.97 bitrate= 308.8kbits/s speed=5.29x
video:62kB audio:47kB subtitle:0kB other streams:0kB global headers:0kB muxing overhead: 3.319973%
[libx264 @ 0x59ebb7bb9680] frame I:1     Avg QP:21.85  size:  6320
[libx264 @ 0x59ebb7bb9680] frame P:47    Avg QP:21.91  size:  1020
[libx264 @ 0x59ebb7bb9680] frame B:27    Avg QP:23.47  size:   297
[libx264 @ 0x59ebb7bb9680] consecutive B-frames: 49.3%  5.3%  8.0% 37.3%
[libx264 @ 0x59ebb7bb9680] mb I  I16..4: 32.4% 66.9%  0.7%
[libx264 @ 0x59ebb7bb9680] mb P  I16..4:  3.2%  2.7%  0.0%  P16..4: 40.8%  8.9%  6.7%  0.0%  0.0%    skip:37.8%
[libx264 @ 0x59ebb7bb9680] mb B  I16..4:  0.7%  0.8%  0.0%  B16..8: 30.9%  1.1%  0.1%  direct: 1.3%  skip:65.2%  L0:53.1
[libx264 @ 0x59ebb7bb9680] 8x8 transform intra:51.9% inter:83.9%
[libx264 @ 0x59ebb7bb9680] coded y,uvDC,uvAC intra: 27.3% 67.7% 8.8% inter: 9.4% 25.0% 1.1%
[libx264 @ 0x59ebb7bb9680] i16 v,h,dc,p: 14% 30% 13% 43%
[libx264 @ 0x59ebb7bb9680] i8 v,h,dc,ddl,ddr,vr,hd,vl,hu: 18% 29% 41%  2%  1%  2%  1%  2%  3%
[libx264 @ 0x59ebb7bb9680] i4 v,h,dc,ddl,ddr,vr,hd,vl,hu: 17% 31%  6%  4%  2% 38%  2%  0%  0%
[libx264 @ 0x59ebb7bb9680] i8c dc,h,v,p: 42% 26% 26%  6%
[libx264 @ 0x59ebb7bb9680] Weighted P-Frames: Y:0.0% UV:0.0%
[libx264 @ 0x59ebb7bb9680] ref P L0: 64.8%  7.6% 18.4%  9.2%
[libx264 @ 0x59ebb7bb9680] ref B L0: 79.2% 14.6%  6.2%
[libx264 @ 0x59ebb7bb9680] ref B L1: 94.6%  5.4%
[libx264 @ 0x59ebb7bb9680] kb/s:166.16
[aac @ 0x59ebb7bda6c0] Qavg: 738.618
```

```
from moviepy.editor import VideoFileClip
from IPython.display import HTML

# Load the video file
video_path = './data/s1/bras9a.mpg' #if you already converted

# Load the video clip
clip = VideoFileClip(video_path)

# Display the video inline
clip.ipython_display(width=640, height=480)
```

```
WARNING:py.warnings:/usr/local/lib/python3.10/dist-packages/moviepy/video/io/sliders.py:61: SyntaxWarning: "is" with a l
  if event.key is 'enter':

Moviepy - Building video __temp__.mp4.
```

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

```
from typing_extensions import Text
print('REAL TEXT')
text=[tf.strings.reduce_join([num_to_char(word) for word in sentence]) for sentence in [sample[1]]]
print(text[0].numpy().decode('utf-8'))
```

```
REAL TEXT
bin red at s nine again
```

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

```
1/1 [==============================] - 3s 3s/step
```

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

```
print('PREDICTIONS')
text = [tf.strings.reduce_join([num_to_char(word) for word in sentence]) for sentence in decoded]
print(text[0].numpy().decode('utf-8'))
```

```
PREDICTIONS
bin red at s nine again
```

hence we can see real text and predictions are matching !!