In [1]:

```
# mounting drive

from google.colab import drive
drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

In [13]:

```
# importing libraries

import os
import numpy as np
import pandas as pd
import cv2
import h5py
from keras import backend as K
from keras.models import Sequential
from keras.layers import Conv2D, BatchNormalization, Activation, MaxPooling2D
from keras.layers import Flatten, Dense
import tensorflow as tf
import tensorflow_addons as tfa
from os import listdir
from os.path import isfile, join
import sys
import dlib
import moviepy.editor as mpy
import wave
import scipy.io.wavfile as wav
import contextlib
```

In [18]:

```
pip install speechpy
```

```
Collecting speechpy
  Downloading
https://files.pythonhosted.org/packages/8f/12/dbda397a998063d9541d9e149c4f523ed138a48824d20598e3763
3b1/speechpy-2.4-py2.py3-none-any.whl
Requirement already satisfied: numpy in /usr/local/lib/python3.6/dist-packages (from speechpy)
(1.18.5)
Requirement already satisfied: scipy in /usr/local/lib/python3.6/dist-packages (from speechpy)
(1.4.1)
Installing collected packages: speechpy
Successfully installed speechpy-2.4
```

In [19]:

```
import speechpy
```

In [4]:

```
#http://www.practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-
coefficients-mfccs/

def audio_processing(wav_file, verbose):
    ''' takes audio file as input and creates mfcc features '''

    """To extract mfcc features of audio, clips 0.2 seconds in length each,
    i.e. of 20 MFCC features in each clip (acc. to syncnet paper)
    Output mfcc_clips shape === (N, 12, 20, 1),
    where N = len(mfcc_features) // 20
    """

    rate, sig = wav.read(wav_file)
    if verbose:
        print("Sig length: {}, sample rate: {}".format(len(sig), rate))
```

```python
    try:
        mfcc_features = speechpy.feature.mfcc(sig, sampling_frequency=rate, frame_length=0.010, frame_stride=0.010)
    except IndexError:
        raise ValueError("ERROR: Index error occurred while extracting mfcc")

    if verbose:
        print("mfcc_features shape:", mfcc_features.shape)

    # Number of audio clips = len(mfcc_features) // length of each audio clip
    number_of_audio_clips = len(mfcc_features) // AUDIO_TIME_STEPS

    if verbose:
        print("Number of audio clips:", number_of_audio_clips)

    # Don't consider the first MFCC feature, only consider the next 12 (Checked in syncnet_demo.m)
    # Also, only consider AUDIO_TIME_STEPS*number_of_audio_clips features
    mfcc_features = mfcc_features[:AUDIO_TIME_STEPS*number_of_audio_clips, 1:]

    # Reshape mfcc_features from (x, 12) to (x//20, 12, 20, 1)
    mfcc_features = np.expand_dims(np.transpose(np.split(mfcc_features, number_of_audio_clips), (0, 2, 1)), axis=-1)

    if verbose:
        print("Final mfcc_features shape:", mfcc_features.shape)
    return mfcc_features
```

In [5]:

```python
def make_rect_shape_square(rect):
    # Rect: (x, y, x+w, y+h)

    x = rect[0]
    y = rect[1]
    w = rect[2] - x
    h = rect[3] - y
    # If width > height
    if w > h:
        new_x = x
        new_y = int(y - (w-h)/2)
        new_w = w
        new_h = w
    # Else (height > width)
    else:
        new_x = int(x - (h-w)/2)
        new_y = y
        new_w = h
        new_h = h
    return [new_x, new_y, new_x + new_w, new_y + new_h]


def expand_rect(rect, scale, frame_shape, scale_w=1.5, scale_h=1.5):

    if scale is not None:
        scale_w = scale
        scale_h = scale
    # Rect: (x, y, x+w, y+h)
    x = rect[0]
    y = rect[1]
    w = rect[2] - x
    h = rect[3] - y
    # new_w, new_h
    new_w = int(w * scale_w)
    new_h = int(h * scale_h)
    # new_x
    new_x = int(x - (new_w - w)/2)
    if new_x < 0:
        new_w = new_x + new_w
        new_x = 0
    elif new_x + new_w > (frame_shape[1] - 1):
        new_w = (frame_shape[1] - 1) - new_x
    # new_y
    new_y = int(y - (new_h - h)/2)
    if new_y < 0:
        new_h = new_y + new_h
```

```
        new_y = 0
    elif new_y + new_h > (frame_shape[0] - 1):
        new_h = (frame_shape[0] - 1) - new_y
    return [new_x, new_y, new_x + new_w, new_y + new_h]

def detect_mouth_in_frame(frame, detector, predictor, prevFace, verbose):
    ''' takes frames as input and detect face and mouth from it, then return it with proper coordinates '''

    # Detect all faces
    faces = detector(frame, 1)

    # If no faces are detected
    if len(faces) == 0:
        if verbose:
            print("No faces detected, using prevFace", prevFace, "(detect_mouth_in_frame)")
        faces = [prevFace]

    # Note first face (ASSUMING FIRST FACE IS THE REQUIRED ONE!)
    face = faces[0]
    # Predict facial landmarks
    shape = predictor(frame, face)
    # Note all mouth landmark coordinates
    mouthCoords = np.array([[shape.part(i).x, shape.part(i).y] for i in range(48, 68)])

    # Mouth Rect: x, y, x+w, y+h
    mouthRect = [np.min(mouthCoords[:, 1]), np.min(mouthCoords[:, 0]),
                 np.max(mouthCoords[:, 1]), np.max(mouthCoords[:, 0])]

    # Make mouthRect square
    mouthRect = make_rect_shape_square(mouthRect)

    # Expand mouthRect square
    expandedMouthRect = expand_rect(mouthRect, scale=(MOUTH_TO_FACE_RATIO * face.width() / mouthRect[2]), frame_shape=(frame.shape[0], frame.shape[1]))

    # Mouth
    mouth = frame[expandedMouthRect[1]:expandedMouthRect[3],
                  expandedMouthRect[0]:expandedMouthRect[2]]

    # # Resize to 120x120
    # resizedMouthImage = np.round(resize(mouth, (120, 120), preserve_range=True)).astype('uint8')

    # Return mouth
    return mouth, face

def video_processing(video):
  ''' takes video as input and returns array for the detected mouth '''

  predictor_path = '/content/gdrive/My Drive/shape_predictor_68_face_landmarks.dat'
  detector = dlib.get_frontal_face_detector()
  predictor = dlib.shape_predictor(predictor_path)

  cap = cv2.VideoCapture(video)

  # Default face rect
  face = dlib.rectangle(30, 30, 220, 220)
  lip_model_input = []
  frame_index = 0
  while(cap.isOpened()):

          frames = []
          for i in range(5):
              _, frame = cap.read()
              frame_index += 1
              # print("Frame", frame_index+1, "of", frameCount, end="\r")

              # If no frame is read, break
              if frame is None:
                  break

              # Detect mouth in the frame
              mouth, _ = detect_mouth_in_frame(frame, detector, predictor, prevFace=face, verbose=False)

              # Convert mouth to grayscale
              mouth = cv2.cvtColor(mouth, cv2.COLOR_BGR2GRAY)
```

```
            mouth = cv2.cvtColor(mouth, cv2.COLOR_BGR2GRAY)

            # Resize mouth to syncnet input shape
            mouth = cv2.resize(mouth, (MOUTH_W, MOUTH_H))

            # Subtract 110 from all mouth values (Checked in syncnet_demo.m)
            mouth = mouth - 110.

            frames.append(mouth)

        if len(frames) == 5:
            stacked = np.stack(frames, axis=-1) #syncnet requires (112,112,5)
            lip_model_input.append(stacked)
        else:
            break

  return np.array(lip_model_input)
```

In [6]:

```
def syncnet_lip_model_v4():
    ''' model layers for lip area from video '''

    # Image data format
    K.set_image_data_format(IMAGE_DATA_FORMAT)
    input_shape = ( MOUTH_H, MOUTH_W, SYNCNET_VIDEO_CHANNELS)

    lip_model = Sequential()     # ( None, 112, 112, 5)

    # conv1_lip
    lip_model.add(Conv2D(96, (3, 3), padding='valid', input_shape=input_shape, name='conv1_lip'))
# (None, 110, 110, 96)
    # bn1_lip
    lip_model.add(BatchNormalization(name='bn1_lip'))
    # relu1_lip
    lip_model.add(Activation('relu', name='relu1_lip'))
    # pool1_lip
    lip_model.add(MaxPooling2D(pool_size=(3, 3), strides=(2, 2), padding='valid', name='pool1_lip')
)   # (None, 54, 54, 96)


    # conv2_lip
    lip_model.add(Conv2D(256, (5, 5), padding='valid', name='conv2_lip'))   # (None, 256, 50, 50)
    # bn2_lip
    lip_model.add(BatchNormalization(name='bn2_lip'))
    # relu2_lip
    lip_model.add(Activation('relu', name='relu2_lip'))
    # pool2_lip
    lip_model.add(MaxPooling2D(pool_size=(3, 3), strides=(2, 2), padding='valid', name='pool2_lip')
)   # (None, 24, 24, 256)


    # conv3_lip
    lip_model.add(Conv2D(512, (3, 3), padding='valid', name='conv3_lip'))   # (None, 22, 22, 512)
    # bn3_lip
    lip_model.add(BatchNormalization(name='bn3_lip'))
    # relu3_lip
    lip_model.add(Activation('relu', name='relu3_lip'))


    # conv4_lip
    lip_model.add(Conv2D(512, (3, 3), padding='valid', name='conv4_lip'))   # (None, 20, 20, 512)
    # bn4_lip
    lip_model.add(BatchNormalization(name='bn4_lip'))
    # relu4_lip
    lip_model.add(Activation('relu', name='relu4_lip'))


    # conv5_lip
    lip_model.add(Conv2D(512, (3, 3), padding='valid', name='conv5_lip'))   # (None, 18, 18, 512)
    # bn5_lip
    lip_model.add(BatchNormalization(name='bn5_lip'))
    # relu5_lip
    lip_model.add(Activation('relu', name='relu5_lip'))
    # pool5_lip
    lip_model.add(MaxPooling2D(pool_size=(3, 3), strides=(3, 3), padding='valid', name='pool5_lip')
)   # (None, 6, 6, 512)
```

```
)    # (None, 6, 6, 512)


    # fc6_lip
    lip_model.add(Flatten(name='flatten_lip'))
    lip_model.add(Dense(256, name='fc6_lip'))     # (None, 256)
    # bn6_lip
    lip_model.add(BatchNormalization(name='bn6_lip'))
    # relu6_lip
    lip_model.add(Activation('relu', name='relu6_lip'))


    # fc7_lip
    lip_model.add(Dense(128, name='fc7_lip'))     # (None, 128)
    # bn7_lip
    lip_model.add(BatchNormalization(name='bn7_lip'))
    # relu7_lip
    lip_model.add(Activation('relu', name='relu7_lip'))


    return lip_model
```

In [7]:

```
def syncnet_audio_model_v4():
    ''' model layers for audio features '''

    # Audio input shape
    input_shape = ( SYNCNET_MFCC_CHANNELS, AUDIO_TIME_STEPS, 1)

    audio_model = Sequential()      # (None, 12, 20, 1)

    # conv1_audio
    audio_model.add(Conv2D(64, (3, 3), padding='same', name='conv1_audio', input_shape=input_shape)
)   # (None, 12, 20, 64)
    # bn1_audio
    audio_model.add(BatchNormalization(name='bn1_audio'))
    # relu1_audio
    audio_model.add(Activation('relu', name='relu1_audio'))


    # conv2_audio
    audio_model.add(Conv2D(128, (3, 3), padding='same', name='conv2_audio'))    # (None, 12, 20, 128
)
    # bn2_audio
    audio_model.add(BatchNormalization(name='bn2_audio'))
    # relu2_audio
    audio_model.add(Activation('relu', name='relu2_audio'))
    # pool2_audio
    audio_model.add(MaxPooling2D(pool_size=(1, 3), strides=(1, 2), padding='valid', name='pool2_aud
io'))    # (None, 12, 9, 128)


    # conv3_audio
    audio_model.add(Conv2D(256, (3, 3), padding='same', name='conv3_audio'))    # (None, 12, 9, 256)
    # bn3_audio
    audio_model.add(BatchNormalization(name='bn3_audio'))
    # relu3_audio
    audio_model.add(Activation('relu', name='relu3_audio'))


    # conv4_audio
    audio_model.add(Conv2D(256, (3, 3), padding='same', name='conv4_audio'))    # (None, 12, 9, 256)
    # bn4_audio
    audio_model.add(BatchNormalization(name='bn4_audio'))
    # relu4_audio
    audio_model.add(Activation('relu', name='relu4_audio'))


    # conv5_audio
    audio_model.add(Conv2D(256, (3, 3), padding='same', name='conv5_audio'))    # (None, 12, 9, 256)
    # bn5_audio
    audio_model.add(BatchNormalization(name='bn5_audio'))
    # relu5_audio
    audio_model.add(Activation('relu', name='relu5_audio'))
    # pool5_audio
    audio_model.add(MaxPooling2D(pool_size=(3, 3), strides=(2, 2), padding='valid', name='pool5_aud
```

```
io'))    # (None, 5, 4, 256)


    # fc6_audio
    audio_model.add(Flatten(name='flatten_audio'))
    audio_model.add(Dense(256, name='fc6_audio'))     # (None, 256)
    # bn6_audio
    audio_model.add(BatchNormalization(name='bn6_audio'))
    # relu6_audio
    audio_model.add(Activation('relu', name='relu6_audio'))


    # fc7_audio
    audio_model.add(Dense(128, name='fc7_audio'))     # (None, 128)
    # bn7_audio
    audio_model.add(BatchNormalization(name='bn7_audio'))
    # relu7_audio
    audio_model.add(Activation('relu', name='relu7_audio'))


    return audio_model
```

In [8]:

```
def load_syncnet_model(mode, verbose):
    ''' loading the syncnet model '''

    if mode == 'lip' or mode == 'both':
      # Load frontal model
      syncnet_lip_model = syncnet_lip_model_v4()

    if mode == 'audio' or mode == 'both':
      # Load frontal model
      syncnet_audio_model = syncnet_audio_model_v4()

    if mode == 'lip':
        syncnet_model = syncnet_lip_model
    elif mode == 'audio':
        syncnet_model = syncnet_audio_model
    elif mode == 'both':
        syncnet_model = [syncnet_audio_model, syncnet_lip_model]

    return syncnet_model

# https://github.com/voletiv/syncnet-in-keras/blob/master/syncnet-weights/syncnet-weights-
readme.md

def load_syncnet_weights( verbose):
    ''' reading and loading pre trained weights file '''

    syncnet_weights_file = '/content/gdrive/My Drive/lipsync_v4_73.mat'

    if verbose:
        print("Loading syncnet_weights from", syncnet_weights_file)

    if not os.path.isfile(syncnet_weights_file):
        raise ValueError(
            "\n\nERROR: syncnet_weight_file missing!! File: " + syncnet_weights_file + \
            "\nPlease specify correct file name in the syncnet_params.py file and relaunch.\n")

    # Read weights file, with layer names
    with h5py.File(syncnet_weights_file, 'r') as f:
        syncnet_weights = [f[v[0]][:] for v in f['net/params/value']]
        syncnet_layer_names = [[chr(i) for i in  f[n[0]]] \
                               for n in f['net/layers/name']]

    # Find the starting index of audio and lip layers
    audio_found = False
    audio_start_idx = 0
    lip_found = False
    lip_start_idx = 0

    # Join the chars of layer names to make them words
    for i in range(len(syncnet_layer_names)):
        syncnet_layer_names[i] = ''.join(syncnet_layer_names[i])
```

```
        # Finding audio_start_idx
        if not audio_found and 'audio' in syncnet_layer_names[i]:
            audio_found = True
            if verbose:
                print("Found audio")
        elif not audio_found and 'audio' not in syncnet_layer_names[i]:
            if 'conv' in syncnet_layer_names[i]:
                audio_start_idx += 2
            elif 'bn' in syncnet_layer_names[i]:
                audio_start_idx += 3
            elif 'fc' in syncnet_layer_names[i]:
                audio_start_idx += 2

        # Finding lip_start_idx
        if not lip_found and 'lip' in syncnet_layer_names[i]:
            lip_found = True
            if verbose:
                print("Found lip")
        elif not lip_found and 'lip' not in syncnet_layer_names[i]:
            if 'conv' in syncnet_layer_names[i]:
                lip_start_idx += 2
            elif 'bn' in syncnet_layer_names[i]:
                lip_start_idx += 3
            elif 'fc' in syncnet_layer_names[i]:
                lip_start_idx += 2

        if verbose:
            print("  ", i, syncnet_layer_names[i])

    if verbose:
        print("  lip_start_idx =", lip_start_idx)
        print("  audio_start_idx =", audio_start_idx)

    return syncnet_weights, syncnet_layer_names, audio_start_idx, lip_start_idx

def set_syncnet_weights_to_syncnet_model(syncnet_model, syncnet_weights, syncnet_layer_names, mode
, verbose):
    ''' loading pre trained weights into the syncnet model layers '''

    if verbose:
        print("Setting weights to model")

    # Video syncnet-related weights begin at 35 in syncnet_weights
    if mode == 'lip':
        syncnet_weights_idx = 35
    else:
        syncnet_weights_idx = 0

    if mode == 'both':
        syncnet_lip_model = syncnet_model[0]
        syncnet_audio_model = syncnet_model[1]

    # Init syncnet_layer_idx, to be incremented only at 'lip' layers
    syncnet_layer_idx = -1

    # Load weights layer-by-layer
    for i in syncnet_layer_names:

        # Skip the irrelevant layers
        if mode == 'lip' and 'lip' not in i:
            continue
        elif mode == 'audio' and 'audio' not in i:
            continue

        # Increment the index on the model
        syncnet_layer_idx += 1

        if verbose:
            print("  SyncNet Layer", syncnet_layer_idx, ":", i, "; weight index :",
syncnet_weights_idx)

        # Convolutional layer
        if 'conv' in i:
            syncnet_model.layers[syncnet_layer_idx].set_weights(
                [np.transpose(syncnet_weights[syncnet_weights_idx], (2, 3, 1, 0)),
                 np.squeeze(syncnet_weights[syncnet_weights_idx + 1])])
            syncnet_weights_idx += 2
```

```
            syncnet_weights_idx += 2

        # Batch Normalization layer
        elif 'bn' in i:
            syncnet_model.layers[syncnet_layer_idx].set_weights(
                [np.squeeze(syncnet_weights[syncnet_weights_idx]),
                 np.squeeze(syncnet_weights[syncnet_weights_idx + 1]),
                 syncnet_weights[syncnet_weights_idx + 2][0],
                 syncnet_weights[syncnet_weights_idx + 2][1]])
            syncnet_weights_idx += 3

        # ReLU layer
        elif 'relu' in i:
            continue

        # Pooling layer
        elif 'pool' in i:
            continue

        # Dense (fc) layer
        elif 'fc' in i:
            # Skip Flatten layer
            if 'flatten' in syncnet_model.layers[syncnet_layer_idx].name:
                syncnet_layer_idx += 1
            # Set weight to Dense layer
            syncnet_model.layers[syncnet_layer_idx].set_weights(
                [np.reshape(
                    np.transpose(syncnet_weights[syncnet_weights_idx],
                        (2, 3, 1, 0)),
                    (syncnet_weights[syncnet_weights_idx].shape[2]*\
                     syncnet_weights[syncnet_weights_idx].shape[3]*\
                     syncnet_weights[syncnet_weights_idx].shape[1],
                     syncnet_weights[syncnet_weights_idx].shape[0])),
                np.squeeze(syncnet_weights[syncnet_weights_idx + 1])])
            syncnet_weights_idx += 2

def load_pretrained_syncnet_model(mode, verbose):
    ''' final function to call loading functions here and prepare the final model'''

    # mode = {lip, audio, both}
    if mode not in {'lip', 'audio', 'both'}:
        print("\n\nERROR: 'mode' not defined properly! Expected one of {'lip', 'audio', 'both'}, g
ot:", mode, "\n")
        return

    try:

        # Load syncnet model
        syncnet_model = load_syncnet_model(mode=mode, verbose=verbose)

        if verbose:
            print("Loaded syncnet model")

        # Read weights and layer names
        syncnet_weights, syncnet_layer_names, audio_start_idx, lip_start_idx = load_syncnet_weights
(verbose=verbose)

        if verbose:
            print("Loaded syncnet weights.")

        # Set lip weights to syncnet_model
        if mode != 'both':
            set_syncnet_weights_to_syncnet_model(syncnet_model=syncnet_model,
                                                 syncnet_weights=syncnet_weights,
                                                 syncnet_layer_names=syncnet_layer_names,
                                                 mode=mode,
                                                 verbose=verbose)
        else:
            # Audio
            set_syncnet_weights_to_syncnet_model(syncnet_model=syncnet_model[0],
                                                 syncnet_weights=syncnet_weights,
                                                 syncnet_layer_names=syncnet_layer_names,
                                                 mode='audio',
                                                 verbose=verbose)
            # Lip
            set_syncnet_weights_to_syncnet_model(syncnet_model=syncnet_model[1],
                                                 syncnet_weights=syncnet_weights,
```

```
                                                     syncnet_layer_names=syncnet_layer_names,
                                                     mode='lip',
                                                     verbose=verbose)

        if verbose:
            print("Set syncnet weights.")

    except ValueError as err:
        print(err)
        return

    except KeyboardInterrupt:
        print("\n\nCtrl+C was pressed!\n")
        return

    return syncnet_model
```

In [9]:

```
# calling function to load model with weights

mode = 'both'
model=load_pretrained_syncnet_model( mode=mode, verbose=False)
model
```

Out[9]:

```
[<tensorflow.python.keras.engine.sequential.Sequential at 0x7f3d0fac7550>,
 <tensorflow.python.keras.engine.sequential.Sequential at 0x7f3cdf82fa58>]
```

In [10]:

```
# distance functions in tf

def euclidean_distance_loss(y_true, y_pred):
    ''' using tensorflow implementation to calculate distance '''

    dists = tf.linalg.norm(y_pred - y_true, axis=1)
    return dists

def distance_euc_tf(feat1, feat2, vshift=15):
  ''' takes 2 tensors as input and return euclidian distance between those '''

  win_size = vshift*2+1
  paddings = tf.constant([[vshift, vshift+1,], [0, 0]])
  feat2p = tf.pad(feat2, paddings, "CONSTANT")              # padding for feat2

  if len(feat2p) < len(feat1)+win_size:
    # after padding in feat2, if still not getting enough rows to calculate distance in below for
loop, we have to pad 'n' more rows

    n = len(feat1)+win_size-len(feat2p)
    padd = tf.constant([[0, n,], [0, 0]])
    feat2p = tf.pad(feat2p, padd, "CONSTANT")

  dists = []

  # we have to create pairwise distance, so running below for loop so it'll calculate distance of
every row of feat1 with every 31 rows sample of feat2p

  for i in range(0,len(feat1)):
    a=tf.repeat([feat1[i,:]], win_size, axis=0)
    b=feat2p[i:i+win_size,:]
    dists.append(euclidean_distance_loss(a, b))

  mdist = np.mean(np.stack(dists,1),1)      # mdist will be array of 31 values
  mdist = tf.convert_to_tensor(mdist)
  return mdist
```

In [11]:

```
def loss_function(y_true, pred_dist):
  ''' calculates contrastive loss between true and predictive values '''
```

```
  e=0
  for i in range(31):
    e = e + (y_true[i]*(pred_dist[i])**2) + ((1-y_true[i])*max(1-pred_dist[i],0)**2)
  loss = e/(2*31)

  return loss
```

In [24]:

```
def final_fun_1(video):

    MOUTH_H = 112
    MOUTH_W = 112
    FACE_H = 224
    FACE_W = 224
    MOUTH_TO_FACE_RATIO = 0.65
    SYNCNET_VIDEO_FPS = 25
    SYNCNET_VIDEO_CHANNELS = int(0.2 * SYNCNET_VIDEO_FPS)  # 5
    SYNCNET_MFCC_CHANNELS = 12
    AUDIO_TIME_STEPS = 20
    IMAGE_DATA_FORMAT = 'channels_last'

    user=video.split('_')[2]
    user=user.split('/')[1]
    au=video.split('_')[-1]
    au=au.split('.')[0]

    video_fea=video_processing(video)
    audio_fea=audio_processing('/content/gdrive/My Drive/VIDTIMIT/'+user+'/audio/'+au+'.wav',False
)
    #print('video and audio features respectively :',video_fea.shape,audio_fea.shape)

    audio_pred = model[0].predict(audio_fea)
    lip_pred = model[1].predict(video_fea)

    return lip_pred, audio_pred

def final_fun_2(lip_pred, audio_pred, y_true):
    d = distance_euc_tf(lip_pred, audio_pred)
    #print('distance :',d)

    conf = np.median(d)-min(d)
    if conf>3.5:
      print ('video is real')
    else:
      print ('video is fake')

    l=loss_function(y_true,d)
    return l
```

In [25]:

```
# calling final function

video='/content/gdrive/My Drive/vidtimit_videos/tampered/fadg0_sa1.mp4'

lip_pred, audio_pred = final_fun_1(video)       # calling function 1
print('video predicted shape',lip_pred.shape)
print('audio predicted shape',audio_pred.shape)

y_true=[0 for i in range(31)]
loss = final_fun_2(lip_pred, audio_pred, y_true)  # calling function 2
print('loss is :', loss)
```

```
video predicted shape (23, 128)
audio predicted shape (23, 128)
video is fake
loss is : tf.Tensor(72.666695, shape=(), dtype=float32)
```

In [ ]: