B.Tech Major Project Report

COT-414

on

SPEECH DRIVEN FACIAL ANIMATION

BY

MUSKAN MANGLA (11510198)

NAINA GUPTA (11520072)

SHRISTI PATHAK (11510455)

Group No. : 04

Under the Supervision of

Dr. S.K. JAIN, Professor

DEPARTMENT OF COMPUTER ENGINEERING

NATIONAL INSTITUTE OF TECHNOLOGY

KURUKSHETRA-136119, HARYANA (INDIA)

December, 2018 – April, 2019

CERTIFICATE

We hereby certify that the work which is being presented in this B.Tech Major Project (COT-414) report entitled "**Speech Driven Facial Animation**", in partial fulfillment of the requirements for the award of the **Bachelor of Technology in Computer Engineering** is an authentic record of my own work carried out during a period from December 2018 to April 2019 under the supervision of Dr. S.K. Jain, Professor, Computer Engineering Department.

The matter presented in this project report has not been submitted for the award of any other degree elsewhere.

Signature of candidates

MUSKAN MANGLA (11510198)

NAINA GUPTA (11520072)

SHRISTI PATHAK (11510455)

This is to certify that the above statement made by the candidates is correct to the best of my knowledge.

Date:

Signature of Supervisor

Dr. S.K. Jain

Professor

TABLE OF CONTENTS

ABSTRACT

Nonverbal behaviour signals, such as facial expressions, provide key information about what we think, act or react. Expressive facial animation is an essential part of modern computer generated movies and digital games. Furthermore, not only the contextual sound units are carried in the audio recording, but also emotional states of the speaker via speed or intensity of the speech. It is an attractive but also challenging to study the signals because they are always hidden or may vary from different people. In this project, we are trying to use deep learning methods for modelling human facial expressions. Therefore, we can get a framework which enables us to predict the facial expression of a never-seen-person when we only hear that person speak.

I. INTRODUCTION

Face synthesis is essential to many applications, such as computer games, animated movies, teleconferencing, talking agents, among others. We present a deep learning framework for speech-driven 3D facial animation from speech audio. We focus on the entire face, not just the mouth and lips. Intuitively, this work is analogous to visual 3D face tracking however, it is more challenging as we try to map acoustic sequence to visual space, instead of conveniently relying on textural cues from input images. Our deep neural network directly maps an input sequence of speech spectrograms to a series of micro facial action unit intensities to drive a 3D blendshape face model. In particular, our deep model is able to learn the latent representations of time-varying contextual information and affective states within the speech. Hence, our model not only activates appropriate facial action units at inference to depict different utterance generating actions, in the form of lip movements, but also, without any assumption, automatically estimates emotional intensity of the speaker and reproduces their ever-changing affective states by adjusting strength of related facial unit activations. For example, in a happy speech, the mouth opens wider than normal, while other facial units are relaxed; or both eyebrows raise higher in a surprised state.

II. MOTIVATION

Expressive facial animation is an essential part of modern computer generated movies and digital games.While the quality obtainable from capture systems is steadily improving, the cost of producing high-quality facial animation remains high. A second, less obvious issue is that whenever new shots are recorded, the actors need to be on location, and ideally also retain their appearance. There lies another problem with vision-based facial capture approaches, however, where part of the face is occluded. In such cases, other input modalities, such as audio, may be exploited to infer facial actions.

A diverse set of applications ranging from entertainment (movies and games), medicine (facial therapy and prosthetics) and education (language/speech training and cyber-assistants) are all empowered by the ability to realistically model, simulate and animate human faces. Speech, as a natural form of communication among various modes of interactions, is becoming more immersive, evidenced by the increasing popularity of virtual voice assistants, such as Microsoft's Cortana or Amazon's Alexa, in our daily lives. Furthermore, not only the contextual sound units (phonemes) are carried in the audio recording, but also emotional states of the speaker via speed or intensity of her speech. Thus, a lively animated 3D head representing the speaker will certainly enhance the speech perception experience in many applications. One such application is the development of talking agent, either in the form of virtual or physical (i.e. robotic) avatars, for face-to-face human machine interaction, as in computer-assisted voice agent. In this scenario, the recorded speech can easily be manipulated, by changing the speed or pitch, to reflect the artificial emotion of the digital assistant. These changes can be automatically reflected visually on the avatar, and make the interaction more engaging. On the other hand, it can also make inter-person telecommunication more enjoyable by expressing speech via personalized avatars, especially in interactive role-playing games, where the gamers communicate with other characters in the virtual world.

III. LITERATURE

In Research Paper[1] , the authors had created a model which will give a face animation when we input the audio of a speaker. They exploit landmarks to generate faces because landmarks can describe and preserve the information of facial movement. CNN and GRU model can find the relationship between landmarks and the audio. They employ Deep Convolution Generated Adversarial Networks (DCGAN) to synthesize faces in an attempt to accentuate micro changes of facial expressions.

In Research Paper[2] , the authors aim to map acoustic sequence to visual space, instead of conveniently relying on textural cues from input images. Moreover, speech emanating facial movements involve different activations of correlated regions on the geometric surface, thus it is difficult to achieve realistic looking, emotion-aware facial deformation from speech sequence. They extract a wide range of acoustics features to capture contextual and emotional progression of the speech. To tackle the difficulty of avatar generation, they utilize the blendshape model in which is purposely designed with enough constraints to ensure that, the final model would always look realistic given a specific set of control parameters. In addition, it can represent various emotional states, e.g. sadness, happiness, etc., without explicitly specifying them. In order to directly map the input features to face shape parameters, they used deep recurrent neural network with LSTM cells to model the long range context of the sequence.

In Research Paper[3] ,the authors aim to recreate a better 3D talking avatar from speech in real-time. Firstly, they forgo using handcrafted, high-level acoustic features which may cause the loss of important information to identify some specific emotions, e.g. happy. Instead, they directly use spectrogram as input to our neural network. Secondly, they employ convolutional neural networks (CNN) to learn meaningful acoustic feature representations, taking advantage of the locality and shift invariance in the time-frequency domain of audio signal. Lastly, they combine these convolutional layers with recurrent layer in an end-to-end network, which learns both temporal transition of facial movements, as well as spontaneous actions and varying emotional states from only speech sequences.

IV. PROPOSED APPROACH

IV.1 Acoustic feature extraction

The input to our system can be any arbitrary speech of any length. As we only use low-level acoustic features, our model is not tied to any particular language. Specifically, we extract Mel-scaled spectrogram, Mel frequency cepstral coefficients (MFCCs) and chromagram from the audio sequence. Mel-scaled spectrogram and MFCCs are standard 82 acoustic features proven to be very effective in presenting the contextual information, whereas chromagram is necessary to determine the pitch in the speech, which reflects the affective states of the speaker throughout the entire sequence. We assume that every input audio sequence is synchronized to the corresponding video at 30 FPS and the audio sampling rate is at 44.1 kHz. Thus, for every video frame, there are 1,470 corresponding audio samples. In every audio window, values of 128 Mel bands, 13 Mel frequency cepstral coefficients and their delta and delta-delta coefficients, and 12 chroma bins, are extracted.

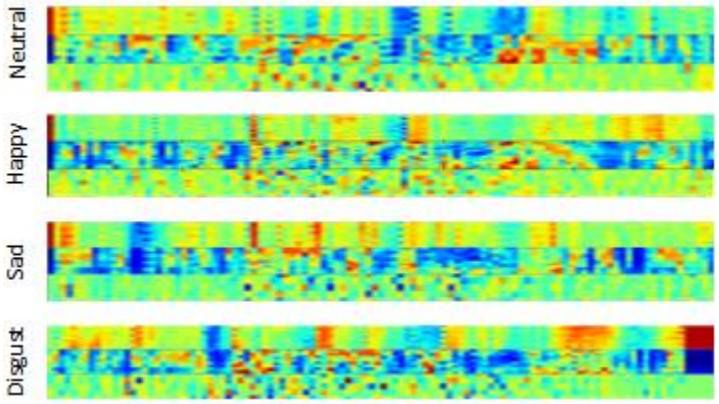

Figure 1: Feature sequences extracted from videos of the same actor speaking the same sentence. ”Kids are talking by the door” under different emotional states. From top row: Neutral, Happy, Sad and Disgust, respectively.

IV.2 Video feature extraction

For each video, we split it into frames. Every video can be split to around 100 frames. Then, we extract landmarks consisted of 68 coordinates from each frame. We exploit landmarks to generate faces because landmarks can describe and preserve the information of facial movement. In order to reduce the influence that not every face is of same size, all landmarks are normalized between 0 to 1. The input audio feature for each video frame is a 128 by 32 matrix.

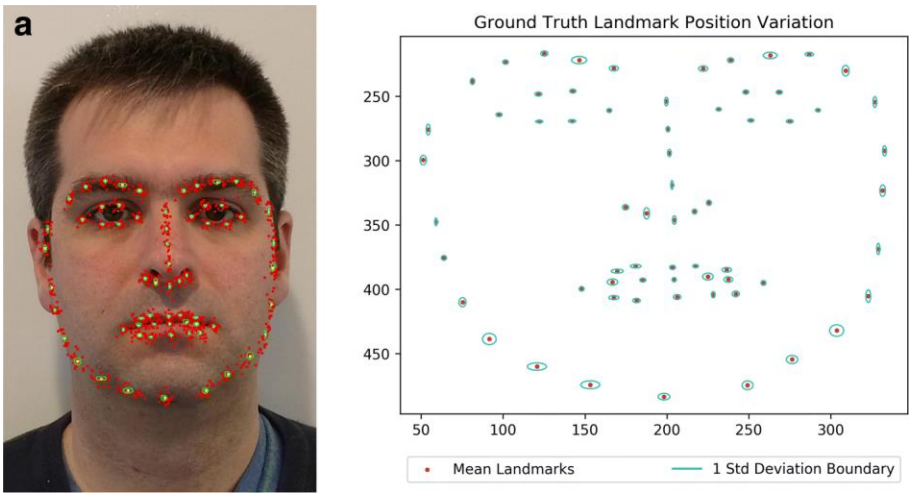

Figure 2. Video to facial landmark generation result. 1st column is real landmark; 2nd column is generated landmark.

IV.3 Data Preprocessing

Generate CNTK text format files to speed the process of training. First 20 actors are assigned for training and the remaining 4 are assigned for testing.

IV.4 Training and mapping

We employ convolutions in the time-frequency domain, and formulate a deep neural network that directly maps input waveforms to blendshape weights. The convolutional neural networks (CNN) learn meaningful acoustic feature representations, taking advantage of the locality and shift invariance in the time-frequency domain of audio signal. Specifically, the time-varying contextual non-linear mapping between audio stream and visual facial movements is realized by training a neural network on a large audio-visual data corpus. Output facial movements are characterized by 3D rotation and blending expression weights of a blendshape model, which can be used directly for animation. The animation phase is very straightforward: given a recorded speech sequence and its features, the model estimates head rotation and deformation parameters, which are then used to animate a 3D face model to visually recreate the facial movements and expression carried in the input speech.

V. IMPLEMENTATION DETAILS

V.1 Architecture

For each video, we split it into frames. Then we extract landmarks consisted of 68 coordinates from each frame. In order to reduce the influence that not every face is of same size, all landmarks are normalized to 0 between 1.

The input to our model is raw time-frequency spectrograms of audio signals. Specifically, each spectrogram contains 128 frequency power bands across 32 time frames, in the form of a 2D (frequency-time) array suitable for CNN. We apply convolutions on frequency and time separately. Specifically, the input spectrogram is first convolved and pooled on the frequency axis with the downsampling factor of two, until the frequency dimension is reduced to one. Then, convolution and pooling is applied on the time axis.The output layer is a fully connected layer of 1,024 units. Our CNN model can find the relationship between landmarks and the audio.

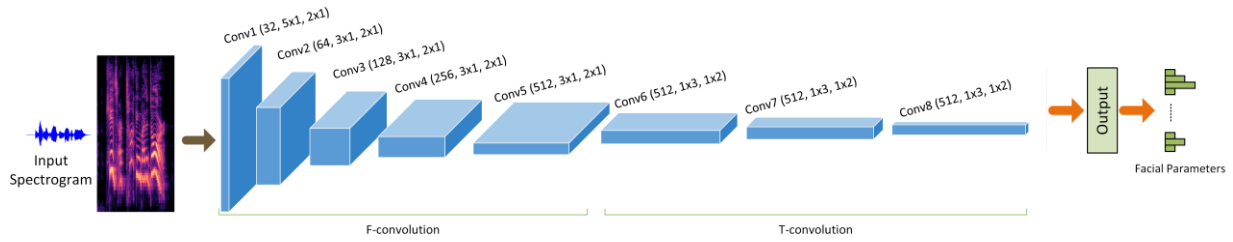

Figure 3: The proposed facial animation framework. The input spectrogram is first convolved over frequency axis (F-convolution), then over time (T-convolution).

V.2 Dataset

Dataset is available at RAVDESS website: https://smartlaboratory.org/ravdess/

We use the Ryerson Audio-Visual Database of Emotional Speech and Song (RAVDESS) for training and evaluation. This dataset contains the complete set of 7356 RAVDESS files (total size: 24.8 GB). Specifically, the database consists of 24 professional actors (12 male and 12 female, respectively) speaking and singing with various emotions. The speech set consists of eight general emotional expressions: neutral, calm, happy, sad, angry, fearful, surprise, and disgust, where each video sequence is associated with one among eight affective states. Similarly, the song set, in which the actors sing short sentences, consists of six general emotional expressions: neutral, calm, happy, sad, angry, and fearful. Both sets are used for training and

testing. We use video sequences of the first 20 actors for training, with around 250,000 frames in total, and evaluate the model on the data of four remaining actors.

Filename Modifiers

- Modality (01 = full-AV)
- Voice channel (01 = speech, 02 = song)
- Emotion (01 = neutral, 02 = calm, 03 = happy, 04 = sad, 05 = angry, 06 = fearful, 07 = disgust, 08 = surprised)
- Emotional intensity (01 = normal, 02 = strong)
- Statement (01 = “Kids are talking by the door”, 02 = “Dogs are sitting by the door”)
- Repetition (01 = 1st repetition, 02 = 2nd repetition)
- Actor (01 to 24. Odd numbered actors are male, even numbered actors are female)

V.3 Hardware/Software Requirements :

- Language Used : Python 3.6
- Libraries Required : OpenCv, TensorFlow, Cntk, Librosa, Pyglet, PIL, Dlib, numpy, Pathlib
- Hardware Required : GPU(Google Colab), SuperComputer (PARAM Shavak)
- Memory Required : Around 80 GB

V.4 Brief desciption about how the model works:

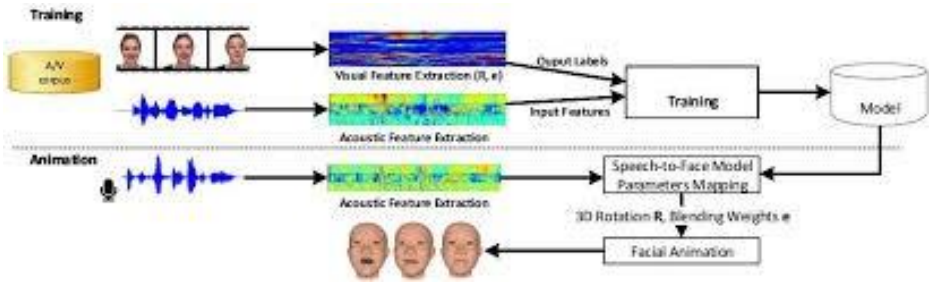

Figure 4: The proposed speech-driven facial animation framework.

1) Frame Extraction : Uses video capture function of Opencv library to divide video into frames and capture those frames. Every video can be split to around 100 frames.
2) Generate Frontal Face Data : For each detected frame, face is detected and cropped.

3) Landmaek Generation : For each frame generate landmarks by using 68 shape landmark point detection file from Dlib library and store them in npy format.
4) Audio Extraction : Extract .wav files from the .mp4 video files using ffmpeg.
5) Acoustic feature Extraction : Extract feature array from audio files using Librosa resulting in mfcc, log-mel, chroma and dbspectrogram files (.csv format).
6) Data Preprocessing : Generate .ctf files by merging and further processing the already generated csv and npy files to recover a power spectrogram of 128 frequency bands and 32 time frames.
7) Training and mapping : Configure cnn-training model and then computational graph and learner are created. The model was trained and stored as .dnn file. Also, the process was logged side by side in a log file.

At any given time t, the deep model estimates Yt = (Rt, Et) from an input spectrogram xt.Yr for rotation and Ye for expression weights. Rotation and deformation parameters (R; e) are the output of our deep model.

$Yrt = (WXt + b)$

$Yet = (WEt + b)$

$Yt = (Yrt, Yet)$

We train the model by minimizing the square error:

$E = \sum_t \| y_t - \hat{y}_t \|^2$

VI. Data Flow Diagram

VI.1 Level 0 DFD

Figure 5: Level 0 DFD

VI.2 Level 1 DFD

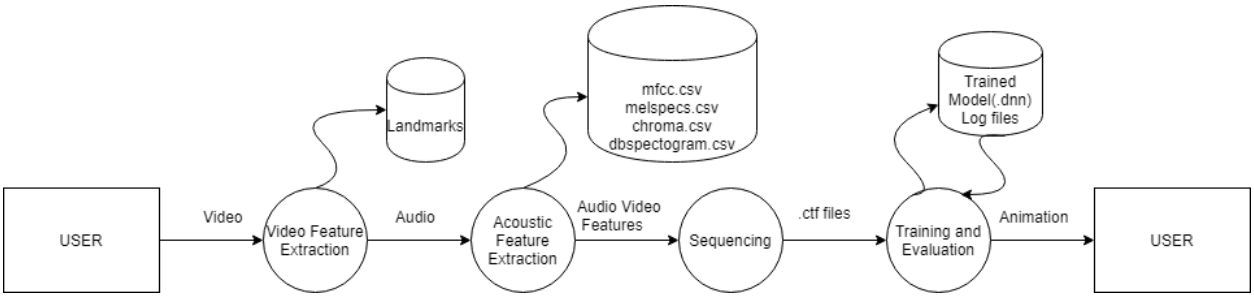

Figure 6: Level 1 DFD

VI.3 Level 2 DFD

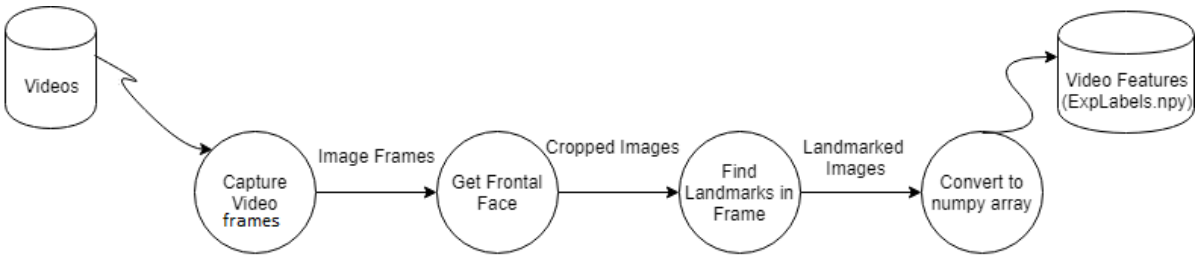

Figure 7: Level 2 Video Feature Extraction DFD

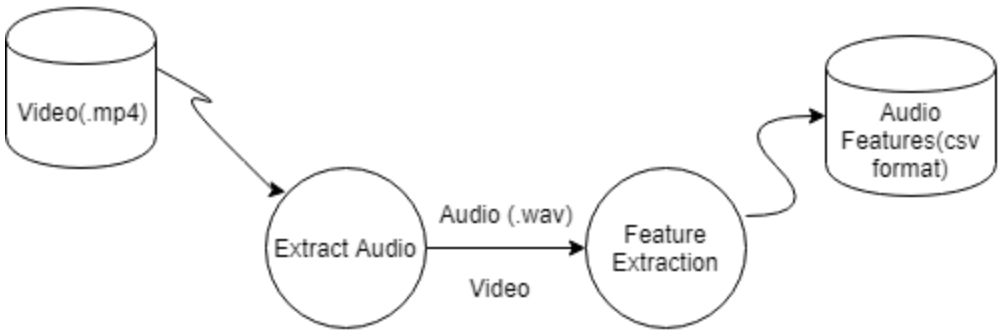

Figure 8: Level 2 Acoustic Feature Extraction DFD

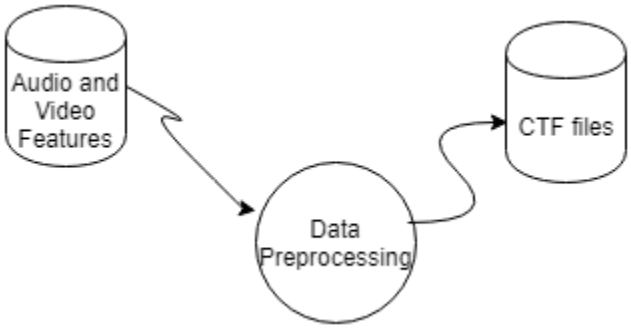

Figure 9: Level 2 Data Preprocessing DFD

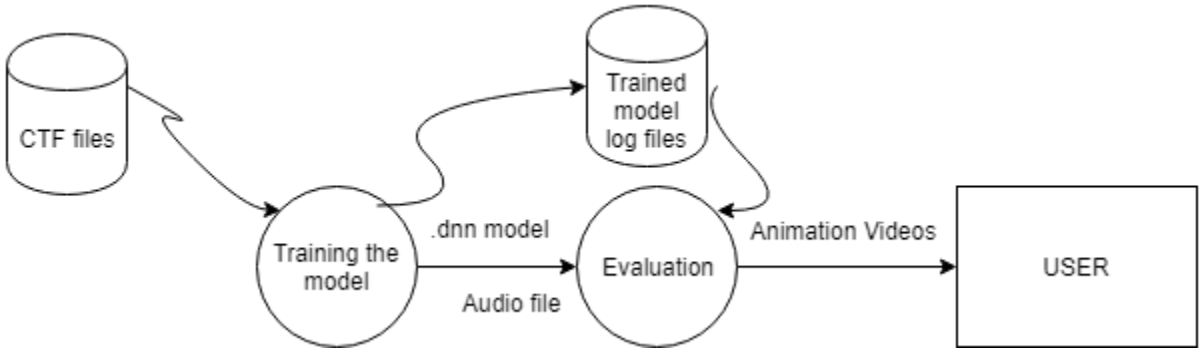

Figure 10: Level 2 Training and Mapping DFD

VII. RESULT

Experiments on a diverse audiovisual corpus of different actors across a wide range of emotional states show interesting and promising results of our approach. Our model yields reasonable results even when driven with audio from various speakers with different gender, accent, or language.The RMSE of CNN-static is about 0.7mm consistently across different categories. These results prove the robustness, generalization and adaptability of the CNN architecture and suggest that we have pursued the right direction in using CNN to model facial action from raw waveforms, but there are limitations and deficiencies in our current approach that need to be addressed.

MSE on AU coefficients (× 1e-2) : 6.53

RMSE (unit : mm) on 3D landmarks : 1.038

Temporal smoothness (× 1e-2) : 1.94

Self smoothness (× 1e-2) : 2.654

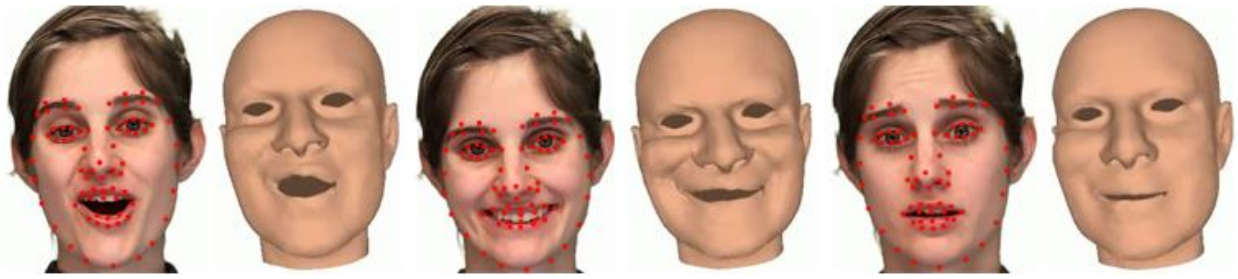

Figure 11 : A few samples from the RAVDESS database, where a 3D facial blendshape (right) is aligned to the face of the actor (left) in the corresponding frame. Red dots indicate 3D landmarks of the model projected to the image plane.

VIII. CONCLUSION AND FUTURE PLAN

This project introduces a deep learning framework for speech-driven 3D facial animation from sequence of input spectrograms. Our proposed deep neural network learns a mapping from audio signal to the temporally varying context of the speech, as well as emotional states of the speaker represented implicitly via blending weights of a 3D face model. Experiments demonstrate that our approach could estimate lip movements with emotional intensity of the speaker reasonably from just her speech.

In future work, we will improve the generalization of our deep neural network, and explore other generative models to increase the quality and realism of facial reconstruction.We hope to see a more principled study of these and related effects in a realistic interactive setting with two or more speakers. We believe our work is a reasonably good baseline for further research in speech-driven facial animation. We will also try to explore the ability to learn features directly from the raw waveform data, and incorporate deep generative model in our framework to improve its facial parameter generation quality.

REFERENCES

[1] Speech-Driven Facial Animation by CNN-RNN-GAN Model, Fengru Li, Xiaoyi Tang, Fei Tao, Ke Xu, Department of Electrical and Computer Engineering, Rutgers University

[2] Pham, H. X., Cheung, S., & Pavlovic, V. (2017, July). Speech-driven 3d facial animation with implicit emotional awareness: a deep learning approach. In 2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW) (pp. 2328-2336). IEEE.

[3] Pham, H. X., Wang, Y., & Pavlovic, V. (2018, October). End-to-end Learning for 3D Facial Animation from Speech. In Proceedings of the 2018 on International Conference on Multimodal Interaction (pp. 361-365). ACM.

[4] Livingstone, S. R., & Russo, F. A. (2018). The Ryerson Audio-Visual Database of Emotional Speech and Song (RAVDESS): A dynamic, multimodal set of facial and vocal expressions in North American English. PloS one, 13(5), e0196391.

[5] Pham, H. X., Wang, Y., & Pavlovic, V. (2017). End-to-end learning for 3d facial animation from raw waveforms of speech. arXiv preprint arXiv:1710.00920.

APPENDIX

A. COMPLETE CONTRIBUTORY SOURCE CODE

A.1 landmark_generation.py

```
import cv2
import numpy as np
import dlib
import os
detector = dlib.get_frontal_face_detector()
predictor= dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")

def find_landmarks_from_frame(frame):
    cv2.imwrite("../ExpLabels/frame.jpg" , frame)
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
    clahe_image = clahe.apply(gray)
    detections = detector(clahe_image, 1)

    for k,d in enumerate(detections):
        landmarks = []
        shape = predictor(clahe_image, d)
        for i in range(0,68):
            landmarks.append((shape.part(i).x, shape.part(i).y))
            cv2.circle(frame, (shape.part(i).x, shape.part(i).y), 1, (0,0,255), thickness=2)
    cv2.imwrite("../ExpLabels/frame.jpg" , frame)
    return landmarks

def get_normalization_standard_points(landmarks):
    landmarks_array = np.array(landmarks)
    xmax = np.max(landmarks_array[:,0])
    xmin = np.min(landmarks_array[:,0])
```

```
    ymax = np.max(landmarks_array[:,1])
    ymin = np.min(landmarks_array[:,1])
    return {"xmax":xmax,"xmin":xmin,"ymax":ymax,"ymin":ymin}

def normalize_landmarks(landmarks,standard_points):
    normalized_landmarks = []
    x_length =  standard_points['xmax'] -  standard_points['xmin']
    y_length =  standard_points['ymax'] -  standard_points['ymin']
    for pair in landmarks:
        normalized_x = (pair[0] - standard_points['xmin']) / float(x_length)
        normalized_y = (pair[1] - standard_points['ymin']) / float(y_length)
        normalized_landmarks.extend((normalized_x,normalized_y))
    return normalized_landmarks

all_landmarks =[]
file_count=1
for filename in os.listdir('../video/'):
    print("Preprocess Video "+str(file_count))
    file_count+=1
    video_landmarks = []
    if filename != ".DS_Store":
        vidcap = cv2.VideoCapture('../video/'+filename)
        success,image = vidcap.read()
        count = 0
        step=30
        success = True
        while success:
            vidcap.set(cv2.CAP_PROP_POS_MSEC,count*step)
            success,image = vidcap.read()
            if success:
                count += 1
                landmarks = find_landmarks_from_frame(image)
```

```
            standard_points = get_normalization_standard_points(landmarks)
            normalized_landmarks = normalize_landmarks(landmarks,standard_points)
            all_landmarks.append(np.array(normalized_landmarks).T.tolist())
np.save('../ExpLabels.npy',np.array(all_landmarks,dtype=np.float32))
```

A.2 frame_extract.py

```
import cv2
import os

def convert_to_frame(test_file):
    vidcap = cv2.VideoCapture(test_file)
    print(vidcap)
    success,image = vidcap.read()
    count = 0
    while success:
        success,image = vidcap.read()
        print("reading frame ",count,success)
        crop_img = image[:, 280:1000]
        cv2.waitKey(0)
        cv2.imwrite("frame%d.jpg" % count, crop_img)     # save frame as JPEG file
        count += 1
convert_to_frame("test.mp4")
```

A.3 extract_wav.py

```
import pathlib as plb
import subprocess as sup
video_root = "../video"
audio_root = "../speech"

def extract_one_video(video_file, audio_file):
    command = "ffmpeg -i " + video_file + " -y -ab 160k -ac 2 -ar 44100 -vn " + audio_file + " -
loglevel quiet"
    sup.call(command, shell=True)

def make_dirs(video_root, audio_root):
    vr_dir = plb.Path(video_root)
    ar_dir = plb.Path(audio_root)
    try:
        ar_dir.mkdir()
    except FileExistsError:
        print ("Audio Root Directory existed")
    except FileNotFoundError:
        print ("Parent directory not found")
    for actor in vr_dir.iterdir():
        if actor.is_dir():
            try:
                plb.Path(audio_root + "/" + actor.name).mkdir()
            except FileExistsError:
                print ("Directory " + actor.name + "  existed")

def convert_all(video_root, audio_root):
    vr_dir = plb.Path(video_root)
    for actor in vr_dir.iterdir():
        for video_file in actor.iterdir():
```

```
            if video_file.name[len(video_file.name)-4:] == '.mp4':
                video_path=video_root+"/"+actor.name+"/" +video_file.stem+".mp4"
                audio_path = audio_root + "/" + actor.name + "/" + video_file.stem + ".wav"
                extract_one_video(video_path, audio_path)

if __name__ == "__main__":
    make_dirs(video_root, audio_root)
    convert_all(video_root, audio_root)
```

A.4 extract_feature.py

```
import cv2
import math
import pathlib as plb
import librosa
import csv
import numpy as np

def write_csv(filename, data):
    with open(filename, 'w', newline="") as csvfile:
        writer = csv.writer(csvfile)
        for arow in data:
            writer.writerow(arow)

def get_fps(videofile):
    cap = cv2.VideoCapture(videofile)
    fps = cap.get(cv2.CAP_PROP_FPS)
    nFrame = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
    cap.release()
    return (nFrame, fps)

def extract_one_frame_data(data, curPosition, nFrameSize, nSamPerFrame):
    frameData = np.zeros(nFrameSize, dtype=np.float32)
    if curPosition < 0:
        startPos = -curPosition
        frameData[startPos:nFrameSize] = data[0:(curPosition+nFrameSize)]
    else:
        frameData[:] = data[curPosition:(curPosition+nFrameSize)]
    nextPos = curPosition + nSamPerFrame
    return (frameData, nextPos)
```

```
def extract_one_file(videofile, audiofile):
    print (" --- " + videofile)
    nFrames, fps = get_fps(videofile)
    data, sr = librosa.load(audiofile, sr=44100)
    nSamPerFrame = int(math.floor(float(sr) / fps))
    n25sSam = int(math.ceil(float(sr) * 0.025))
    nSamPerStep = 512
    nStepsPerFrame = 3
    nFrameSize = (nStepsPerFrame - 1) * nSamPerStep + n25sSam
    curPos = nSamPerFrame - nFrameSize
    mfccs = []
    melspecs = []
    chromas = []
    for f in range(0,nFrames):
        frameData, nextPos = extract_one_frame_data(data, curPos, nFrameSize, nSamPerFrame)
        curPos = nextPos
        S = librosa.feature.melspectrogram(frameData, sr, n_mels=128,
hop_length=nSamPerStep)
        log_S = librosa.amplitude_to_db(S)
        mfcc = librosa.feature.mfcc(y=frameData, sr=sr, hop_length=nSamPerStep, n_mfcc=13)
        delta_mfcc = librosa.feature.delta(mfcc,mode='nearest')
        delta2_mfcc = librosa.feature.delta(delta_mfcc,mode='nearest')
        chroma = librosa.feature.chroma_cqt(frameData, sr, hop_length=nSamPerStep)

        full_mfcc = np.concatenate([mfcc[:,0:3].flatten(), delta_mfcc[:,0:3].flatten(),
delta2_mfcc[:,0:3].flatten()])
        mfccs.append(full_mfcc.tolist())
        melspecs.append(log_S[:,0:3].flatten().tolist())
        chromas.append(chroma[:,0:3].flatten().tolist())
    return (mfccs, melspecs, chromas)

video_root = "../video"
```

```
audio_root = "../speech"
feat_root ="../feat_new"

def process_all():
    video_dir = plb.Path(video_root)
    feat_dir = plb.Path(feat_root)
    feat_dir.mkdir(parents=True, exist_ok=True)
    for actor in video_dir.iterdir():
        for video_file in actor.iterdir():
            if video_file.name[len(video_file.name)-4:] != '.mp4' or video_file.name[0:2] != '01':
                continue
            ar_dir=  plb.Path(feat_root + "/" + actor.name)
            if not ar_dir.exists():
                try:
                    ar_dir.mkdir()
                except FileExistsError:
                    print ("Directory " + actor.name + "  existed")

            seq_dir = plb.Path( feat_root + "/" + actor.name + "/" + video_file.stem )
            if seq_dir.exists():
                continue
            try:
                seq_dir.mkdir()
            except FileExistsError:
                print ("feat Root Directory existed")
            except FileNotFoundError:
                print ("Parent directory not found")
            video_path = str(video_file)
            audio_path = audio_root + "/" + actor.name + "/" + video_file.stem + ".wav"
            mfccs, melspecs, chromas = extract_one_file(video_path, audio_path)
            mfcc_path = feat_root + "/" + actor.name + "/" + video_file.stem + "/mfcc_2.csv"
            mel_path = feat_root + "/" + actor.name + "/" + video_file.stem + "/log_mel.csv"
```

```
            chroma_path = feat_root + "/" + actor.name + "/" + video_file.stem + "/chroma_cqt.csv"
            write_csv(mfcc_path, mfccs)
            write_csv(mel_path, melspecs)
            write_csv(chroma_path, chromas)

if __name__ == "__main__":
    process_all()
```

A.5 extract_spectrogram.py

```
import math
import pathlib as plb
import librosa
import csv
import numpy as np
from extract_feature import write_csv, get_fps, extract_one_frame_data

FREQ_DIM = 128
TIME_DIM = 32
NFFT = FREQ_DIM*2
nFrameSize = (TIME_DIM - 3) * FREQ_DIM + NFFT

def extract_one_file(videofile, audiofile):
    print (" --- " + audiofile)
    nFrames, fps = get_fps(videofile)
    data, sr = librosa.load(audiofile, sr=44100)
    nSamPerFrame = int(math.floor(float(sr) / fps))
    nSamPerFFTWindow = NFFT
    nSamPerStep = FREQ_DIM
    nStepsPerFrame = TIME_DIM
    nFrameSize = (nStepsPerFrame - 1) * nSamPerStep + nSamPerFFTWindow
    curPos = nSamPerFrame - nFrameSize
    dbspecs = []
    for f in range(0,nFrames):
        frameData, nextPos = extract_one_frame_data(data, curPos, nFrameSize, nSamPerFrame)
        curPos = nextPos
        FD = librosa.core.stft(y=frameData, n_fft=NFFT, hop_length=FREQ_DIM)
        FD, phase = librosa.magphase(FD)
        DB = librosa.core.amplitude_to_db(FD, ref=np.max)
        DB = np.divide(np.absolute(DB), 80.0)
```

```
        newDB = DB[0:-1,:]
        dbspecs.append(newDB.flatten().tolist())
    return dbspecs
video_root = "../video"
audio_root = "../speech"
feat_root = "../feat_new"

def process_all():
    video_dir = plb.Path(video_root)
    feat_dir = plb.Path(feat_root)
    feat_dir.mkdir(parents=True, exist_ok=True)
    for actor in video_dir.iterdir():
        for video_file in actor.iterdir():
            seq_dir = plb.Path( feat_root + "/" + actor.name + "/" + video_file.stem )
            video_path = str(video_file)
            audio_path = audio_root + "/" + actor.name + "/" + video_file.stem + ".wav"
            dbspecs = extract_one_file(video_path, audio_path)
            feature_path = feat_root + "/" + actor.name + "/" + video_file.stem +
"/dbspectrogram.csv"
            write_csv(feature_path, dbspecs)

if __name__ == "__main__":
    process_all()
```

A.6 ravdess_processing.py

```
import pathlib as plb
import csv
import sys

def list_all_sequences(root_path):
    root_dir = plb.Path(root_path)
    actors = []
    for actor_dir in root_dir.iterdir():
        if(actor_dir.is_dir()):
            seqs = []
            for seq_dir in actor_dir.iterdir():
                if seq_dir.name[:2] == "01":
                    seqs.append(str(seq_dir))
            actors.append(seqs)
    return actors

def convert_one_sequence(seq_id, seq_dir, output)
    try:
        mfccsinput = csv.reader(open(seq_dir + "/mfcc_normed.csv"), delimiter=',', quotechar='|')
        melsinput = csv.reader(open(seq_dir + "/log_mel_normed.csv"), delimiter=',',
quotechar='|')
        chromasinput = csv.reader(open(seq_dir + "/chroma_normed.csv"), delimiter=',',
quotechar='|')
        labelsinput = csv.reader(open(seq_dir + "/Explabels.csv"), delimiter=',', quotechar='|')

    except IOError as e:
        (errno,strerror) = e.args
        print ("I/O error({0}): {1}".format(errno,strerror))
        sys.exit(1)
    except:
```

```
    print ("Unexpected Error: ",sys.exc_info()[0])
    raise
mfccs = list(mfccsinput)
mels = list(melsinput)
chromas = list(chromasinput)
labels2 = list(labelsinput)

labels = []
i = 0
for label in labels2:
    if len(label) > 49:
        if label[49] == ':
            print(seq_dir)
            label = label[:49]
    if len(label) > 0:
        labels.append(label)
        if len(label) != 49:
            print ("labels at " + str(i) + " of " + seq_dir + " have less than 49 values")
            return False
        i += 1
if len(labels) != len(mfccs):
    print ("Lengths of features and labels of " + seq_dir + " are different")
    return False
seq_len = min([len(labels), len(mfccs)])
for i in range(0,seq_len):
    if len(labels[i]) != 49:
        print ("labels at " + str(i) + " of " + seq_dir + " have less than 49 value")
        return False
    output.write(str(seq_id))
    ft_str = "\t|features "
    for ft in mels[i]:
        ft_str += str(ft) + " "
```

```
        for ft in chromas[i]:
            ft_str += str(ft) + " "
        for ft in mfccs[i]:
            ft_str += str(ft) + " "
        output.write(ft_str)
        lb_str = "\t|labels "
        for lb in labels[i]:
            lb_str += str(lb) + " "
        output.write(lb_str)
        output.write("\n")
    return True

def convert_one_actor(actor, output, last_seq_id):
    seq_id = last_seq_id
    for seq_dir in actor:
        seq_id += 1
        if not convert_one_sequence(seq_id, seq_dir, output):
            seq_id -= 1
    return seq_id

def convert_all(ctf_train, ctf_test, actors):
    train_num = 0
    test_num = 0
    for i in range(0,20):
      train_num += len(actors[i])
    for i in range(20, 24):
      test_num += len(actors[i])
    print ("train num: " , train_num, " test num: ", test_num)
    train_output = open(ctf_train, "w")
    last_id = -1
    for i in range(20):
        print(" --- Converting actor" + str(i+1) + "\n")
```

```
        last_id = convert_one_actor(actors[i], train_output, last_id)
    test_output = open(ctf_test, "w")
    for i in range(20, 24):
        print(" --- Converting actor" + str(i+1) + "\n")
        last_id = convert_one_actor(actors[i], test_output, last_id)

if __name__ == '__main__':
    root_path = "../feat_root"
    ctf_path="../speech_dir"
    actors = list_all_sequences(root_path)
    ctf_train = ctf_path + "/train_1_20_mfcc_0406.ctf"
    ctf_test = ctf_path + "/test_21_24_mfcc_0406.ctf"
    convert_all(ctf_train, ctf_test, actors)
    print ("Done.")
```

A.7 create_spectrogram_ctf.py

```
import numpy as np
import pathlib as plb
from SysUtils import get_items, make_dir, get_path

def convert_vector_to_string(np_vec):
    a = list(np_vec)
    ret = " ".join(["{:.6f}".format(x) for x in a])
    return ret

def write_seq(nSeq, audio, exp, fout):
    n = audio.shape[0]
    # if nSeq < 0: write line without sequence ID
    if nSeq >= 0:
        for i in range(n):
            rowstr = "{:d} |features {:s} |labels {:s}\n".format(nSeq,
convert_vector_to_string(audio[i,:]), convert_vector_to_string(exp[i,:]))
            fout.write(rowstr)
    else:
        for i in range(n):
            rowstr = "|features {:s} |labels {:s}\n".format(convert_vector_to_string(audio[i,:]),
convert_vector_to_string(exp[i,:]))
            fout.write(rowstr)

def create_ctf_file(seq_root, exp_root, output_file, is_training=True):
    databases = ["RAVDESS","VIDTIMIT","SAVEE"]
    nSeq = 0
    fout = open(output_file, "w")
    for i, db in enumerate(databases):
        db_dir = seq_root + "/" + db
        actors = get_items(db_dir)
```

```
if is_training:
    if db == "RAVDESS":
        actors = actors[:20]
    elif db == "SAVEE":
        actors = actors[:3]
    elif db == "VIDTIMIT":
        actors = actors[:40]
    else:
        raise IOError("folder does not exist!")
else:
    if db == "RAVDESS":
        actors = actors[20:]
    elif db == "SAVEE":
        actors = actors[3:]
    elif db == "VIDTIMIT":
        actors = actors[40:]
    else:
        raise IOError("folder does not exist!")
for actor in actors:
    seq_dir = db_dir + "_feat/" + actor
    exp_dir = exp_root + "/" + db + "/" + actor
    items = get_items(seq_dir)
    for seq in items:
        print(seq_dir + "/" + seq)
        seq_path = seq_dir + "/" + seq + "/dbspectrogram.csv"
        exp_path = exp_dir + "/" + seq + ".npy"
        if plb.Path(seq_path).exists() and plb.Path(exp_path).exists():
            audio = np.loadtxt(seq_path, dtype=np.float32, delimiter=',')
            exp = np.load(exp_path)
            if exp.shape[0] != audio.shape[0]:
                print("length not matched")
                continue
```

```
                write_seq(nSeq, audio, exp, fout)
                nSeq += 1
            else:
                print("file not exist")
    fout.close()

def create_ctf_file_noseq(seq_root, exp_root, output_file, is_training=True):
    databases = ["RAVDESS","VIDTIMIT","SAVEE"]
    fout = open(output_file, "w")
    for i, db in enumerate(databases):
        db_dir = seq_root + "/" + db
        actors = get_items(db_dir)
        if is_training:
            if db == "RAVDESS":
                actors = actors[:20]
            elif db == "SAVEE":
                actors = actors[:3]
            elif db == "VIDTIMIT":
                actors = actors[:40]
            else:
                raise IOError("folder does not exist!")
        else:
            if db == "RAVDESS":
                actors = actors[20:]
            elif db == "SAVEE":
                actors = actors[3:]
            elif db == "VIDTIMIT":
                actors = actors[40:]
            else:
                raise IOError("folder does not exist!")
        for actor in actors:
            seq_dir = db_dir + "_feat/" + actor
```

```
            exp_dir = exp_root + "/" + db + "/" + actor
            items = get_items(seq_dir)
            for seq in items:
                print(seq_dir + "/" + seq)
                seq_path = seq_dir + "/" + seq + "/dbspectrogram.csv"
                exp_path = exp_dir + "/" + seq + ".npy"
                if plb.Path(seq_path).exists() and plb.Path(exp_path).exists():
                    audio = np.loadtxt(seq_path, dtype=np.float32, delimiter=',')
                    exp = np.load(exp_path)
                    if exp.shape[0] != audio.shape[0]:
                        print("length not matched")
                        continue
                    write_seq(-1, audio, exp, fout)
                else:
                    print("file not exist")
    fout.close()

def main():
    seq_root = "../speech_dir"
    exp_root = "../ExpLabels"
    ctf_train = seq_root + "/audio_exp_train_noseq.ctf"
    create_ctf_file_noseq(seq_root, exp_root, ctf_train)
    ctf_test = seq_root + "/audio_exp_test_noseq.ctf"
    create_ctf_file_noseq(seq_root, exp_root, ctf_test, False)
    ctf_train = seq_root + "/audio_exp_train.ctf"
    create_ctf_file(seq_root, exp_root, ctf_train)
    ctf_test = seq_root + "/audio_exp_test.ctf"
    create_ctf_file(seq_root, exp_root, ctf_test, False)
if __name__ == "__main__":
    main()
```

A.8 layerUtils.py

```
import cntk as C
import numpy as np

def lrelu(input, leak=0.2, name=""):
    return C.param_relu(C.constant((np.ones(input.shape)*leak).astype(np.float32)), input,
name=name)

def bn(input, activation=None, name=""):
    if activation is not None:
        x = C.layers.BatchNormalization(map_rank=1, name=name+"_bn" if name else "")(input)
        x = activation(x, name=name)
    else:
        x = C.layers.BatchNormalization(map_rank=1, name=name)(input)
    return x

def bn_relu(input, name=""):
    return bn(input, activation=C.relu, name=name)

def bn_lrelu(input, name=""):
    return bn(input, activation=C.leaky_relu, name=name)

def conv(input, filter_shape, num_filters, strides=(1,1), init=C.he_normal(), activation=None,
pad=True, name=""):
    return C.layers.Convolution(filter_shape, num_filters, strides=strides, pad=pad,
activation=activation, init=init, bias=False, name=name)(input)

def conv_bn(input, filter_shape, num_filters, strides=(1,1), init=C.he_normal(),
activation=None, name=""):
    x = conv(input, filter_shape, num_filters, strides, init, name=name+"_conv" if name else "")
    x = bn(x, activation, name=name)
```

```
    return x

def conv_bn_nopad(input, filter_shape, num_filters, strides=(1,1), init=C.he_normal(),
activation=None, name=""):
    x = conv(input, filter_shape, num_filters, strides, init, pad=False, name=name+"_conv" if
name else "")
    x = bn(x, activation, name=name)
    return x

def conv_bn_relu(input, filter_shape, num_filters, strides=(1,1), init=C.he_normal(), name=""):
    return conv_bn(input, filter_shape, num_filters, strides, init, activation=C.relu, name=name)

def conv_bn_lrelu(input, filter_shape, num_filters, strides=(1,1), init=C.he_normal(), name=""):
    return conv_bn(input, filter_shape, num_filters, strides, init, activation=C.leaky_relu,
name=name)

def conv_bn_relu_nopad(input, filter_shape, num_filters, strides=(1,1), init=C.he_normal(),
name=""):
    return conv_bn_nopad(input, filter_shape, num_filters, strides, init, activation=C.relu,
name=name)

def conv_bn_lrelu_nopad(input, filter_shape, num_filters, strides=(1,1), init=C.he_normal(),
name=""):
    return conv_bn_nopad(input, filter_shape, num_filters, strides, init, activation=C.leaky_relu,
name=name)

def flatten(input, name=""):
    assert (len(input.shape) == 3)
    return C.reshape(input, input.shape[0]*input.shape[1]* input.shape[2], name=name)

def flatten_2D(input, name):
    assert (len(input.shape) >= 3)
```

```
    return C.reshape(input, (input.shape[-3], input.shape[-2]* input.shape[-1]), name=name)

def broadcast_xy(input_vec, h, w):
    """ broadcast input vector of length d to tensor (d x h x w) """
    assert(h > 0 and w > 0)
    d = input_vec.shape[0]
    x = C.reshape(input_vec, (d, 1, 1))
    t = np.zeros((d, h, w), dtype=np.float32)
    y = C.constant(t)
    z = C.reconcile_dynamic_axes(y, x)
    z = z + x
    return z

def conv_from_weights(x, weights, bias=None, padding=True, name=""):
    k = C.parameter(shape=weights.shape, init=weights)
    y = C.convolution(k, x, auto_padding=[False, padding, padding])
    if bias:
        b = C.parameter(shape=bias.shape, init=bias)
        y = y + bias
    y = C.alias(y, name=name)
    return y

def bi_recurrence(input, fwd, bwd, name=""):
    F = C.layers.Recurrence(fwd, go_backwards=False, name='fwd_rnn')(input)
    B = C.layers.Recurrence(bwd, go_backwards=True, name='bwd_rnn')(input)
    h = C.splice(F, B, name=name)
    return h
```

A.9 SysUtils.py

```
import pathlib as plb
import datetime
import sys
import argparse

def is_Win32():
    return sys.platform.startswith("win")

def make_dir(dir_path, exist_ok=True):
    plb.Path(dir_path).mkdir(parents=True, exist_ok=exist_ok)

def get_current_time_string():
    return datetime.datetime.now().strftime('%Y-%m-%d-%H-%M')

def get_items(folder, what_to_get=None):
    if what_to_get is None:
        what_to_get = "stem"
    if what_to_get not in ["stem", "name", "full"]:
        raise ValueError("Incorrect flag")
    path = plb.Path(folder)
    if what_to_get == "stem":
        return [item.stem for item in path.iterdir()]
    elif what_to_get == "name":
        return [item.name for item in path.iterdir()]
    elif what_to_get == "full":
        return [str(item) for item in path.iterdir()]

def get_path(args):
    return "/".join(args)
```

```
def get_filename(path):
    return plb.Path(path).stem

def get_extension(path):
    return plb.Path(path).suffix

def get_parent_dir(path):
    return plb.Path(path).parent

def read_learning_rate(filename):
    if not plb.Path(filename).exists():
        return None
    lr = []
    with open(filename, "r") as f:
        for line in f:
            tokens = line.split()
            if len(tokens) > 2:
                raise ValueError("there are more than 2 entries")
            if len(tokens) == 0:
                continue
            elif len(tokens) == 1:
                lr += [float(tokens[0])]
            elif len(tokens) == 2:
                lr += [float(tokens[0])]*int(tokens[1])
    return lr

class ArgParser(object):
    def __init__(self):
        self.config = {}
        self.config["num_epochs"] = 300
        self.config["minibatch_size"] = 128
        self.config["epoch_size"] = 50000
```

```
        self.config["constlr"] = True
        self.config["lr"] = 0.0001
        self.config["lr_list"] = None
        self.config["momentum"] = 0.9

    def prepare(self):
        self.parser = argparse.ArgumentParser(description="Process input arguments for training")
        self.parser.add_argument("--epoch", type=int, default=self.config["num_epochs"])
        self.parser.add_argument("--minibatch", type=int, default=self.config["minibatch_size"])
        self.parser.add_argument("--epoch_size", type=int, default=self.config["epoch_size"])
        self.parser.add_argument("--constlr", action="store_true")
        self.parser.add_argument("--lr", type=float, default=self.config["lr"])
        self.parser.add_argument("--momentum", type=float, default=self.config["momentum"])
        self.parser.add_argument("--lr_file", type=str)

    def parse(self):
        self.args = self.parser.parse_args()
        self.config["num_epochs"] = self.args.epoch
        self.config["minibatch_size"] = self.args.minibatch
        self.config["epoch_size"] = self.args.epoch_size
        self.config["constlr"] = self.args.constlr
        self.config["lr"] = self.args.lr
        self.config["momentum"] = self.args.momentum

        if self.config["num_epochs"] < 1 or self.config["minibatch_size"] < 1 or
self.config["epoch_size"] < 1:
            raise ValueError("number of minibatches, epochs or minibatch size must be positive")
        if (not self.config["constlr"]) and self.args.lr_file:
            self.config["lr_list"] = read_learning_rate(self.args.lr_file)
```

A.10 train_end2end.py

```
import numpy as np
import cntk as C
import sys
import argparse
from LayerUtils import conv_bn_lrelu, bi_recurrence, flatten
from SysUtils import get_current_time_string, is_Win32, make_dir, ArgParser
C.cntk_py.set_fixed_random_seed(1)
F_DIM = 128
T_DIM = 32
input_dim_model = (1, F_DIM, T_DIM)
input_dim = F_DIM*T_DIM
label_dim = 46

def audio_encoder(input):
    h = conv_bn_lrelu(input, filter_shape=(5,1), num_filters=32, strides=(2,1), name="conv1")
    h = conv_bn_lrelu(h, filter_shape=(3,1), num_filters=64, strides=(2,1), name="conv2")
    h = conv_bn_lrelu(h, filter_shape=(3,1), num_filters=128, strides=(2,1), name="conv3")
    h = conv_bn_lrelu(h, filter_shape=(3,1), num_filters=256, strides=(2,1), name="conv4")
    h = conv_bn_lrelu(h, filter_shape=(3,1), num_filters=512, strides=(2,1), name="conv5")

    h = conv_bn_lrelu(h, filter_shape=(1,3), num_filters=512, strides=(1,2), name="t_conv1")
    h = conv_bn_lrelu(h, filter_shape=(1,3), num_filters=512, strides=(1,2), name="t_conv2")
    h = conv_bn_lrelu(h, filter_shape=(1,3), num_filters=512, strides=(1,2), name="t_conv3")
    return h

def audio_encoder_2(input):
    h = conv_bn_lrelu(input, filter_shape=(5,1), num_filters=64, strides=(2,1), name="conv1")
    h = conv_bn_lrelu(h, filter_shape=(3,1), num_filters=128, strides=(2,1), name="conv2")
    h = conv_bn_lrelu(h, filter_shape=(3,1), num_filters=256, strides=(2,1), name="conv3")
    h = conv_bn_lrelu(h, filter_shape=(3,1), num_filters=512, strides=(2,1), name="conv4")
```

```
    h = conv_bn_lrelu(h, filter_shape=(3,1), num_filters=1024, strides=(2,1), name="conv5")

    h = conv_bn_lrelu(h, filter_shape=(1,3), num_filters=1024, strides=(1,2), name="t_conv1")
    h = conv_bn_lrelu(h, filter_shape=(1,3), num_filters=1024, strides=(1,2), name="t_conv2")
    h = conv_bn_lrelu(h, filter_shape=(1,3), num_filters=1024, strides=(1,2), name="t_conv3")
    return h

def audio_encoder_3(input, model_file, cloning=False):
    last_layer_name = "t_conv3"
    model = C.load_model(model_file)
    input_node = model.find_by_name("input")
    last_conv = model.find_by_name(last_layer_name)
    if not last_conv:
        raise ValueError("the layer does not exist")
    h = C.combine([last_conv.owner]).clone(C.CloneMethod.clone if cloning else
C.CloneMethod.freeze, {input_node: input})
    return h

def create_model(input, net_type="gru", encoder_type=1, model_file=None, e3cloning=False):
    if encoder_type == 1:
        h = audio_encoder(input)
        if net_type.lower() is not "cnn":
            h = flatten(h)
    elif encoder_type == 2:
        h = audio_encoder_2(input)
        h = C.layers.GlobalAveragePooling(name="avgpool")(h)
        h = C.squeeze(h)
    elif encoder_type == 3:
        h = audio_encoder_3(input, model_file, e3cloning)
        if net_type.lower() is not "cnn":
            h = flatten(h)
    else:
```

```
        raise ValueError("encoder type {:d} not supported".format(encoder_type))
    if net_type.lower() == "cnn":
        h = C.layers.Dense(1024, init=C.he_normal(), activation=C.tanh)(h)
    elif net_type.lower() == "gru":
        h = C.layers.Recurrence(step_function=C.layers.GRU(256), go_backwards=False,
name="rnn")(h)
    elif net_type.lower() == "lstm":
        h = C.layers.Recurrence(step_function=C.layers.LSTM(256), go_backwards=False,
name="rnn")(h)
    elif net_type.lower() == "bigru":
        h = bi_recurrence(h, C.layers.GRU(128), C.layers.GRU(128), name="bigru")
    elif net_type.lower() == "bilstm":
        h = bi_recurrence(h, C.layers.LSTM(128), C.layers.LSTM(128), name="bilstm")
    h = C.layers.Dropout(0.2)(h)
    # output
    y = C.layers.Dense(label_dim, activation=C.sigmoid, init=C.he_normal(), name="output")(h)
    return y

def l2_loss(output, target):
    return C.reduce_mean(C.square(output - target))

def std_normalized_l2_loss(output, target):
    std_inv = np.array([6.6864805402, 5.2904440280, 3.7165409939, 4.1421640454,
8.1537399389, 7.0312877415, 2.6712380967,
                        2.6372177876, 8.4253649884, 6.7482162880, 9.0849960354, 10.2624412692,
3.1325531319, 3.1091179819,
                        2.7337937590, 2.7336441031, 4.3542467871, 5.4896293687, 6.2003761588,
3.1290341469, 5.7677042738,
                        11.5460919611, 9.9926451700, 5.4259818848, 20.5060642486, 4.7692101480,
3.1681517575, 3.8582905289,
                        3.4222250436, 4.6828286809, 3.0070785113, 2.8936539301, 4.0649030157,
25.3068458731, 6.0030623160,
```

```
                3.1151977458, 7.7773542649, 6.2057372469, 9.9494258692, 4.6865422850,
5.3300697628, 2.7722027974,
                4.0658663003, 18.1101618617, 3.5390113731, 2.7794520068],
dtype=np.float32)
    weights = C.constant(value=std_inv) #.reshape((1, label_dim)))
    dif = output - target
    ret = C.reduce_mean(C.square(C.element_times(dif, weights)))
    return ret

def l1_reg_loss(output):
    return C.reduce_mean(output)

def build_graph(config):
    assert(config['type'] in ["cnn", "lstm", "gru", "bilstm", "bigru"])
    if config["type"] == "cnn":
        features = C.input_variable(input_dim_model, name="input")
        labels = C.input_variable(label_dim, name="label")
    else:
        features = C.sequence.input_variable(input_dim_model, name="input")
        labels = C.sequence.input_variable(label_dim, name="label")
    netoutput = create_model(features, config["type"], config["encoder"],
config["pretrained_model"], config["e3_clone"])

    if config["l2_loss_type"] == 1:
        print("Use standard l2 loss")
        ce = l2_loss(netoutput, labels)
    elif config["l2_loss_type"] == 2:
        print("Use variance normalized l2 loss")
        ce = std_normalized_l2_loss(netoutput, labels)
    else:
        raise ValueError("Unsupported loss type")
```

```
    if config["l1_reg"] > sys.float_info.epsilon:
        ce = ce + config["l1_reg"] * l1_reg_loss(netoutput)
    pe = C.squared_error(netoutput, labels
    if config["constlr"]:
        lr_schedule = config["lr"]
    else:
        if config["lr_list"] is not None:
            print("use learning rate schedule from file")
            lr_schedule = config["lr_list"]
        else:
            if config["type"] != "cnn": # default learning rate for recurrent model
                lr_schedule = [0.005] + [0.0025]*2 + [0.001]*4 + [0.0005]*8 + [0.00025]*16 +
[0.0001]*1000 + [0.00005]*1000 + [0.000025]
            elif config["lr_schedule"] == 1: # learning rate for CNN
                lr_schedule = [0.005] + [0.0025]*2 + [0.00125]*3 + [0.0005]*4 + [0.00025]*5 +
[0.0001]
            elif config["lr_schedule"] == 2:
                lr_schedule = [0.005] + [0.0025]*2 + [0.00125]*3 + [0.0005]*4 + [0.00025]*5 +
[0.0001]*100 + [0.00005]*50 + [0.000025]*50 + [0.00001]
            else:
                raise ValueError("unknown learning rate")
    learning_rate = C.learning_parameter_schedule_per_sample(lr_schedule,
epoch_size=config["epoch_size"])
    momentum_schedule = C.momentum_schedule(0.9, minibatch_size=300)

    learner = C.adam(netoutput.parameters, lr=learning_rate, momentum=momentum_schedule,
                     l2_regularization_weight=0.0001,
                     gradient_clipping_threshold_per_sample=3.0,
gradient_clipping_with_truncation=True)
    trainer = C.Trainer(netoutput, (ce, pe), [learner])

  return features, labels, netoutput, trainer
```

```
def create_reader(path, is_training=True):
    return C.io.MinibatchSource(C.io.CTFDeserializer(path, C.io.StreamDefs(
        features = C.io.StreamDef(field='features', shape=input_dim, is_sparse=False),
        labels = C.io.StreamDef(field='labels', shape=label_dim, is_sparse=False)
    )), randomize=is_training, max_sweeps = C.io.INFINITELY_REPEAT if is_training else 1)

def train(config):
    features, labels, netoutput, trainer = build_graph(config)
    C.logging.log_number_of_parameters(netoutput)
    epoch_size = config["epoch_size"]
    progress_printer = C.logging.ProgressPrinter(freq=200, tag='Training') # more detailed
logging
    minibatch_size = config["minibatch_size"]
    max_epochs = config["num_epochs"]
    model_file = config["modelfile"]
    log_file = config["logfile"]
    reader = create_reader(config["datafile"])
    input_map = {features: reader.streams.features, labels: reader.streams.labels}
    t = 0
    for epoch in range(max_epochs):
        epoch_end = (epoch+1) * epoch_size
        while t < epoch_end:
            data = reader.next_minibatch(min(minibatch_size, epoch_end-t),
input_map=input_map)
            trainer.train_minibatch(data)
            t += trainer.previous_minibatch_sample_count
            progress_printer.update_with_trainer(trainer, with_metric=True) # log progress
        loss, metric, actual_samples = progress_printer.epoch_summary(with_metric=True)
        with open(log_file, 'a') as csvfile:
            csvfile.write("Epoch:  "+ str(epoch) + " Loss: " + str(loss) + " Metric: " + str(metric) +
"\n")
     netoutput.save(model_file)
```

```
    return features, labels, netoutput, trainer

def evaluate(datafile, features, labels, trainer, log_file):
    progress_printer = C.logging.ProgressPrinter(tag="Evaluation", num_epochs=0)
    minibatch_size = 200
    reader = create_reader(datafile, is_training=False)
    input_map = {features: reader.streams.features, labels: reader.streams.labels}
    while True:
        data = reader.next_minibatch(minibatch_size, input_map=input_map)
        if not data:
            break
        metric = trainer.test_minibatch(data)
        progress_printer.update(0, data[labels].num_samples, metric) # log progress
    loss, metric, actual_samples = progress_printer.epoch_summary(with_metric=True)
    with open(log_file, 'a') as csvfile:
        csvfile.write("\n\n --- Test error: " + str(metric) + "\n")

def process_args():
    class ThisArgParser(ArgParser):
        def __init__(self):
            super(ThisArgParser, self).__init__()
            self.config["minibatch_size"] = 300
            self.config["lr_schedule"] = 1
            self.config["type"] = "gru"
            self.config["encoder"] = 1
            self.config["pretrained_model"] = None
            self.config["e3_clone"] = False

        def prepare(self):
            super(ThisArgParser, self).prepare()
            self.parser.add_argument("--gru", action="store_true")
            self.parser.add_argument("--lstm", action="store_true")
```

```
        self.parser.add_argument("--cnn", action="store_true")
        self.parser.add_argument("--bigru", action="store_true")
        self.parser.add_argument("--bilstm", action="store_true")
        self.parser.add_argument("--l2type", type=int, default=2)
        self.parser.add_argument("--l1reg", type=float, default=0.1)
        self.parser.add_argument("--lrschd", type=int, default=self.config["lr_schedule"])
        self.parser.add_argument("--encoder", type=int, default=self.config["encoder"])
        self.parser.add_argument("--pretrained_model", type=str)
        self.parser.add_argument("--e3clone", action="store_true")

    def parse(self):
        super(ThisArgParser, self).parse()
        self.config["lr_schedule"] = self.args.lrschd
        self.config["l2_loss_type"] = self.args.l2type
        self.config["l1_reg"] = self.args.l1reg
        self.config["lr_schedule"] = self.args.lrschd
        self.config["encoder"] = self.args.encoder
        if self.args.pretrained_model:
            self.config["pretrained_model"] = self.args.pretrained_model
        if self.args.cnn:
            self.config["type"] = "cnn"
        if self.args.lstm:
            self.config["type"] = "lstm"
        if self.args.gru:
            self.config["type"] = "gru"
        if self.args.bigru:
            self.config["type"] = "bigru"
        if self.args.bilstm:
            self.config["type"] = "bilstm”
        if self.args.e3clone:
            self.config["e3_clone"] = True
parser = ThisArgParser()
```

```
    parser.prepare()
    parser.parse()
    config = parser.config
    return config

def main():
    config = process_args()
    print("training type: {:s}".format(config["type"]))
    print("max epoch: {:d}".format(config["num_epochs"]))
    current_time = get_current_time_string()
    data_dir = "../speech_dir”
    if config["type"] == "cnn":
        train_file = data_dir + "/audio_exp_train_noseq.ctf"
        test_file = data_dir + "/audio_exp_test_noseq.ctf"
    else:
        train_file = data_dir + "/audio_exp_train.ctf"
        test_file = data_dir + "/audio_exp_test.ctf"
    model_dir = data_dir + "/model_audio2exp_" + current_time
    make_dir(model_dir)
    print("directory created")
    model_filename = model_dir + "/model_audio2exp_" + current_time
    model_file = model_filename + ".dnn"
    log_file = model_dir + "/training_log.txt"
    if config["encoder"] == 3:
        if not config["pretrained_model"]:
            config["pretrained_model"] = data_dir + "/model_audio2exp_2019-03-24-21-03.dnn"
    else:
        config["pretrained_model"] = None
    config["datafile"] = train_file
    config["modelfile"] = model_file
    config["logfile"] = log_file
```

```
    features, labels, netoutput, trainer = train(config)
    print ("Training done!")
    evaluate(test_file, features, labels, trainer, log_file)
    print ("Testing done!")

if __name__=='__main__':
    main()
```

A.11 ShapeUtils.py

```
import numpy as np
import math
import pyglet
from pyglet.gl import *
import ctypes
from PIL import Image
import cv2

def load_processed_baseshapes(filename=""):
    if filename:
        baseshapes = np.load(filename)
    else:
        baseshapes = np.load("../shapes/baseshapes.npy")
    return baseshapes

def load_triangles(filename=""):
    if filename:
        triangles = np.load(filename)
    else:
        triangles = np.load("../shapes/triangles.npy")
    return triangles

def calc_shape(allshapes, e):
    assert(e.size == 46)
    e = np.array(e).astype(np.float32)
    full_e = np.ones(e.size+1, dtype=np.float32)
    full_e[1:] = e
    shape = full_e @ allshapes
    numVert = int(shape.size / 3)
    ret_shape = np.reshape(shape, (numVert,3))
```

```
    return ret_shape

def calc_all_shapes(allshapes, Es):
    shapes = []
    for e in Es:
        shape = cal_shape(allshapes, e)
        shapes.append(shape)
    return shapes

def transform_shape(shape, R=None, T=None):
    ret_shape = np.copy(shape)
    if R is not None:
        ret_shape = ret_shape @ R.transpose()
    if T is not None:
        ret_shape = np.add(ret_shape, T)
    return ret_shape

def calc_vertex_normals(vertices, triangles):

    def normalize_v3(arr):
        lens = np.sqrt( arr[:,0]**2 + arr[:,1]**2 + arr[:,2]**2 )
        arr[:,0] /= lens
        arr[:,1] /= lens
        arr[:,2] /= lens
        return arr

    norm = np.zeros(vertices.shape, dtype=vertices.dtype)
    tris = vertices[triangles]
    n = np.cross(tris[: :,1] - tris[: :,0], tris[: :,2] - tris[: :,0])
    normalize_v3(n)
    norm[triangles[:,0]] += n
    norm[triangles[:,1]] += n
```

```
    norm[triangles[:,2]] += n
    normalize_v3(norm)
    return norm

class Renderer(object):
    def __init__(self, width=640, height=480, caption="Renderer"):
        self.win = pyglet.window.Window(width=width, height=height, caption=caption,
visible=False)
        self.width = width
        self.height = height
        self.initGL()
        self.buffer = (GLubyte * (3*width*height))(0)

    def initGL(self):
        glClearColor(1.0, 1.0, 1.0, 1.0)
        glClearDepth(1.0)
        glShadeModel(GL_SMOOTH)
        glPolygonMode(GL_FRONT_AND_BACK, GL_FILL)

    def render(self, shape, triangles, normals=None, text=""):
        self.win.dispatch_events()
        if normals is None:
            normals = calc_vertex_normals(shape, triangles)
        glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT)
        glClearColor(1.0, 1.0, 1.0, 1.0)
        glEnable(GL_DEPTH_TEST)
        glEnable(GL_LIGHTING)
        glEnable(GL_LIGHT0)
        ambient_light = (GLfloat*4)(*[0.25, 0.25, 0.25, 1.0])
        diffuse_light = (GLfloat*4)(*[0.7, 0.7, 0.7, 1.0])
        light_pos = (GLfloat*3)(*[-1000.0, 0.0, 200000.0])
        glLightfv(GL_LIGHT0, GL_AMBIENT, ambient_light)
```

```
glLightfv(GL_LIGHT0, GL_DIFFUSE, diffuse_light)
glLightfv(GL_LIGHT0, GL_POSITION, light_pos)
diffuse_material = (GLfloat*4)(*[0.5, 0.5, 0.5, 1.0])
glEnable(GL_COLOR_MATERIAL)
glColorMaterial(GL_FRONT, GL_AMBIENT_AND_DIFFUSE)
glMaterialfv(GL_FRONT, GL_DIFFUSE, diffuse_material)
glMatrixMode(GL_PROJECTION)
glLoadIdentity()
gluPerspective(45.0, float(self.width)/float(self.height), 0.1, 100.0)
glMatrixMode(GL_MODELVIEW)
glLoadIdentity()
glTranslatef(0.0, 0.0, -4.0)
glEnableClientState(GL_VERTEX_ARRAY)
glEnableClientState(GL_NORMAL_ARRAY)
va = shape[triangles].flatten().astype(ctypes.c_float)
no = normals[triangles].flatten().astype(ctypes.c_float)
nTri = triangles.shape[0]
glVertexPointer(3, GL_FLOAT, 0, va.ctypes.data)
glNormalPointer(GL_FLOAT, 0, no.ctypes.data)
glColor3f(0.81, 0.59, 0.49)
glDrawArrays(GL_TRIANGLES, 0, nTri*3)
glColor3f(0.67, 0.34, 0.22)
eyebrow_tris = np.array([    [9427, 7187, 9414],
                             [9427, 9414, 7170],
                             [9427, 7170, 9429],
                             [9429, 7170, 2139],
                             [714, 4271, 10870],
                             [10870, 4271, 4451],
                             [4451, 4271, 4275],
                             [4451, 4275, 4293] ], dtype=np.int32)
ebv = shape[eyebrow_tris].flatten().astype(ctypes.c_float)
ebn = normals[eyebrow_tris].flatten().astype(ctypes.c_float)
```

```
        glVertexPointer(3, GL_FLOAT, 0, ebv.ctypes.data)
        glNormalPointer(GL_FLOAT, 0, ebn.ctypes.data)
        glDisable(GL_DEPTH_TEST)
        glDrawArrays(GL_TRIANGLES, 0, 24)
        if text:
            glDisable(GL_DEPTH_TEST)
            glDisable(GL_LIGHT0)
            glDisable(GL_LIGHTING)
            glDisable(GL_COLOR_MATERIAL)
            glMatrixMode(GL_PROJECTION)
            glLoadIdentity()
            gluOrtho2D(0, self.width, 0, self.height)
            glMatrixMode(GL_MODELVIEW)
            glLoadIdentity()
            pyglet.text.Label(text, font_name='Times New Roman', font_size=24, x=5, y=5,
anchor_x='left',anchor_y='baseline', color=(255, 0, 255, 128)).draw()
        self.win.flip()

    def capture_screen(self):
        glReadBuffer(GL_FRONT)
        glReadPixels(0, 0, self.width, self.height, GL_RGB, GL_UNSIGNED_BYTE, self.buffer)
        image = Image.frombytes(mode="RGB", size=(self.width, self.height), data=self.buffer)
        image = image.transpose(Image.FLIP_TOP_BOTTOM)
        image = np.array(image)
        new_image = np.copy(image)
        new_image[:,:,0] = image[:,:,2]
        new_image[:,:,2] = image[:,:,0]
        return new_image

    def get_3D_render(self, shape, triangles):
        self.render(shape, triangles)
        return self.capture_screen()
```

```
    def exit(self):
        self.win.close()

class Visualizer(object):
    def __init__(self, draw_error=True):
        self.baseshapes = load_processed_baseshapes()
        self.triangles = load_triangles()
        self.renderer = Renderer()
        self.draw_error = draw_error

    def visualize(self, image, e_real, e_fake):
        shape_fake = calc_shape(self.baseshapes, e_fake)
        shape_real = calc_shape(self.baseshapes, e_real)
        self.renderer.render(shape_real, self.triangles)
        img_real = self.renderer.capture_screen()
        self.renderer.render(shape_fake, self.triangles)
        img_fake = self.renderer.capture_screen()
        new_img = np.zeros((300,900,3), dtype=np.uint8)
        if image is not None:
            new_img[:,0:300,:] = cv2.resize(image, (300,300), interpolation=cv2.INTER_CUBIC)
        new_img[:,300:600,:] = img_real[52:352,170:470,:]
        new_img[:,600:900,:] = img_fake[52:352,170:470,:]
        if self.draw_error:
            error = np.sum(np.square(e_real-e_fake))
            txt = "error: {:.4f}".format(error)
            cv2.putText(new_img, txt, (10,280), cv2.FONT_HERSHEY_COMPLEX, 1.0, (255, 0,
255), 1)
        return new_img

    def exit(self):
        self.renderer.exit()
```

```
    def restart(self):
        self.renderer = Renderer()


def draw_error_bar_plot(e_real, e_fake, final_size=(900,100)):
    error = np.round(np.abs(e_real - e_fake) * 100.0).astype(np.int32)
    eg = np.round(e_real * 100.0).astype(np.int32)
    ef = np.round(e_fake * 100.0).astype(np.int32)
    # draw 46 bars
    img = np.zeros((460, 220, 3), dtype=np.uint8) + 255
    for i in range(46):
        # draw the error bars
        y1 = 2 + i*10
        y2 = y1 + 6
        x1 = 120
        x2 = 120 + error[i]
        img = cv2.rectangle(img, (x1,y1), (x2,y2), (255, 0, 0), cv2.FILLED)
        img = cv2.putText(img, "{:d}".format(i+1), (105, y1+5), cv2.FONT_HERSHEY_PLAIN,
0.5, (0, 0, 255), 1)
        x1 = 0
        x2 = ef[i]
        img = cv2.rectangle(img, (x1,y1), (x2,y1+3), (0, 255, 0), cv2.FILLED)
        # draw e_real bars
        x2 = eg[i]
        img = cv2.rectangle(img, (x1,y1+3), (x2,y2), (0, 0, 255), cv2.FILLED)
    img = cv2.transpose(img)
    img = cv2.flip(img, 0)
    ret = cv2.resize(img, final_size)
    return ret
```

A.12 eval_speech.py

```
import numpy as np
import cntk as C
import cv2
from scipy.signal import medfilt
import ShapeUtils as SU
from SysUtils import make_dir, get_items

def load_image(path):
    img = cv2.imread(path, 1)
    if img.shape[0] != 100 or img.shape[1] != 100:
        img = cv2.resize(img, (100, 100), interpolation=cv2.INTER_CUBIC)
    return img

def load_image_stack(paths):
    imgs = [load_image(path) for path in paths]
    return np.stack(imgs)

def load_exp_sequence(path, use_medfilt=False, ksize=3):
    exp = np.load(path).astype(np.float32)
    if use_medfilt:
        exp = medfilt(exp, kernel_size=(ksize,1)).astype(np.float32)
    return exp

def is_recurrent(model):
    names = ["RNN", "rnn", "LSTM", "LSTM1", "GRU", "gru", "fwd_rnn", "bwd_rnn"]
    isrnn = False
    for name in names:
        if model.find_by_name(name) is not None:
            isrnn = True
    return isrnn
```

```
def estimate_one_audio_seq(model, audio_seq, small_mem=False):
    if isinstance(model, str):
        model = C.load_model(model)
    # set up 2 cases: if the model is recurrent or static
    if is_recurrent(model):
        n = audio_seq.shape[0]
        NNN = 125
        if n > NNN and small_mem:
            nseqs = n//NNN + 1
            indices = []
            for i in range(nseqs-1):
                indices.append(NNN*i + NNN)
            input_seqs = np.vsplit(audio_seq, indices)
            outputs = []
            for seq in input_seqs:
                output = model.eval({model.arguments[0]:[seq]})[0]
                outputs.append(output)
            output = np.concatenate(outputs)
        else:
            output = model.eval({model.arguments[0]:[audio_seq]})[0]
    else:
        output = model.eval({model.arguments[0]: audio_seq})
    return output

def visualize_one_audio_seq(model, video_frame_list, audio_csv_file, exp_npy_file, visualizer,
save_dir):
    if isinstance(model, str):
        model = C.load_model(model)
    audio = np.loadtxt(audio_csv_file, dtype=np.float32, delimiter=",")
    audio_seq = np.reshape(audio, (audio.shape[0], 1, 128, 32))
    e_fake = estimate_one_audio_seq(model, audio_seq)
    if e_fake.shape[1] != 46:
```

```
        if e_fake.shape[1] == 49:
            e_fake = e_fake[:,3:]
        else:
            raise ValueError("unsupported output of audio model")
    e_real = load_exp_sequence(exp_npy_file, use_medfilt=True)
    if e_real.shape[0] != e_fake.shape[0]:
        raise ValueError("number of true labels and number of outputs do not match")
    if video_frame_list:
        video = load_image_stack(video_frame_list)
        if video.shape[0] != e_real.shape[0]:
            print("number of frames and number of labels do not match. Not using video")
            video = None
    else:
        video = None
    make_dir(save_dir)
    n = e_real.shape[0]
    for i in range(n):
        if video is not None:
            img = video[i,:,:,:]
        else:
            img = None
        ef = e_fake[i,:]
        er = e_real[i,:]
        ret = visualizer.visualize(img, er, ef)
        plot = SU.draw_error_bar_plot(er, ef, (ret.shape[1],200))
        ret = np.concatenate([ret, plot], axis=0)
        save_path = save_dir + "/result{:06d}.jpg".format(i)
        cv2.imwrite(save_path, ret)


def test_one_seq(visualizer):
    save_dir = "../speech_dir/test_output_single"
    model_file = "../speech_dir/model_audio2exp_2019-03-24-21-03/model_audio2exp_2019-
```

```
03-24-21-03.dnn"
    video_dir = "../FrontalFaceData/RAVDESS/Actor_21/01-01-07-02-01-01-21"
    audio_file = "../feat_new/Video_Speech_Actor_21/01-01-07-02-01-01-
21/dbspectrogram.csv"
    exp_file = "../ExpLabels/RAVDESS/Actor_21/01-01-07-02-01-01-21.npy”
    video_list = get_items(video_dir, "full")
    model = C.load_model(model_file)
    visualize_one_audio_seq(model, video_list, audio_file, exp_file, visualizer, save_dir)

if __name__ == "__main__":
    visualizer = SU.Visualizer()
    test_one_seq(visualizer)
```