VISUAL SPEECH RECOGNITION

SYNOPSIS

Visual speech Recognition aims at transcribing lip movements into readable text. The system will convert an input video into transcripted text.

The technique uses viseme and phoneme matching, where visemes are particular lip moments and phonemes are corresponding sounds that can be converted into text. In recent years, there have been many advances in automatic speech reading system with the inclusion of audio and visual speech features to recognize words even under noisy conditions. Our model focuses mainly on the visual element, while a good system uses the visual segment as a support for the acoustic segment.

Our system have to match visemes with phonemes. We need to capture exact lip moments such that a system can distinguish between various phonemes. Also, our system needs to handle individual frames that can be passably identical to the next frame.

The system has two modules, the first one will handles extraction of lip features from input while the next is a neural network system trained to process the lip visemes and match it with phonemes.

CONTENTS

CHAPTER **Page No.**

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 VISUAL SPEECH RECOGNITION

Visual Speech Recognition is the process of extracting textual or speech data through image processing techniques. It plays an important role in human-computer interaction mostly in noisy environments. Similar to speech recognition systems, lip reading (LR) systems also face problems due to variances in skin tone, speaking speed, pronunciation, and facial features. To simplify this problem, many systems limit the datasets to contain only certain words and phrases rather than all possible sentences. Here, we extract data from lip movements also called lip features or visemes. Our input is a video file.

The input will have a lot of parameters like height, width, and frame rate and so on. In our case, we place an emphasis on frame rate. The frame rate is the no. of frames per unit time. We extract lip features from each frame and store it. A problem we find here is that there won't be any perceivable difference between the two frames. The changes can be recognized by the system but suitable phoneme match can't be provided by a training dataset. So, fixed number of frames are clumped together so as to get a more consistent result.

The system is made of two segments: one being the feature extraction system that extracts lip features and processes it into a visual feature cube while the other being a Convolutional Neural Network trained on a rich dataset, which matches visemes to corresponding text.

1.2 TYPES OF VISUAL SPEECH RECOGNITION

1.2.1 Speaker Independent (SI)

In this setting, the training and testing data are from different speakers. Adapting a leave-one-out strategy where we train all but one data and then validate it with the left out one. This is followed for each speaker in the dataset.

1.2.2 Speaker Dependent (SD)

In this setting, the training and the testing data are from the same speaker so that it cannot be used for a generalized speech recognition problem.

1.3 MOTIVATION

With an enormous amount of videos available, it is necessary to increase the amount of information that can be extracted from such a file. Details such as data rate, location, and description are metadata and are to be pre-added. Information that can be extracted includes object detection, audio detection and lip reading for transcripts. Consider an educational video in English, so if a non-English speaker needs to make use of it, it is better to transcript audio and visual moments into text. The transcript can be translated into other languages.

This system uses the visual element of videos, the lip features. They provide real-time applications in several areas. Transcript generation in areas where audio transcription is not possible. Improved hearing aids can be augmented with the visual speech recognition system. Speech recognition in a noisy environment can be done using LR systems. Basically, visual element can be augmented to increase the efficiency of Automatic Speech Recognition systems.

1.4 VISEME EXTRACTION

A Viseme is a visual representation of a phoneme or unit of sound in spoken languages. The Viseme for a particular phoneme varies from person to person, language to language, and pronunciation to pronunciation. A Lip Reading system tries to map these visemes to phonemes where visemes and phonemes do not share a one-to-one correspondence. The lip portion (viseme) only carries a high amount of information about the spoken sound than any other facial feature. So visemes are extracted from the input video and are used for classification. There are many methods trying to extract visemes from the frontal face feature. For better output quality of input video and the angle of view makes a great contribution. Video input is broken down into frames and the lip features are matched. They are cropped so that it contains only the lip portion and individual frames are combined together to create a visual cube. The changes between any two frames will be a minute, so a series of frames must be taken together as one. Another issue will be the articulation point, which is the face inclination to the video. Best results will be achieved if the input recording has the lip feature pointed towards the source.

Viseme	Phoneme
V1	/b/ /p/ /m/
V2	/f/ /v/
V3	/t/ /d/ /s/ /z/ /th/ /dh/
V4	/w/ /r/
V5	/k/ /g/ /n/ /l/ /ng/ /hh/ /h/ /y/
V6	/ch/ /jh/ /sh/ /zh/
V7	/ch/ /ey/ /ae/ /aw/ /er/ /ea/
V8	/uh/ /uw/
V9	/iy/ /ih/ /ia/
V10	/ah/ /ax/ /ay/
V11	/ao/ /oy/ /ow/ /ua/
V12	/aa/
V13	/oh/
V14	/sil/

Table 1.1 Fisher mapping of 45 phonemes to 14 visemes including silence

1.5 CNN SYSTEM

Traditional models use conventional Mixture Models where multiple operations can be performed on to extract various phonemes mapped to various positions. Also, ConvNets, which works best with images can be used to extract lip features (viseme) with ease. Visemes and phonemes are mapped and trained, which can be used for classification. Our model focuses mainly on the visual element, while a good system uses the visual part as a support for the acoustic part.

CHAPTER 2

LITERATURE SURVEY

2.1 VISEME EXTRACTION

In the process of visual speech recognition, not all the features of a face are significant. Only lip movements (viseme) tend to be important. In this module, the lip movements are extracted from the input video so that this feature can act as an input to viseme-phoneme mapping module.

2.1.1 EXTRACTION USING YIQ DOMAIN

The video sequences are converted from Red Green Blue (RGB) domain to Luminance (Y), In-phase Quadrature (YIQ) domain.

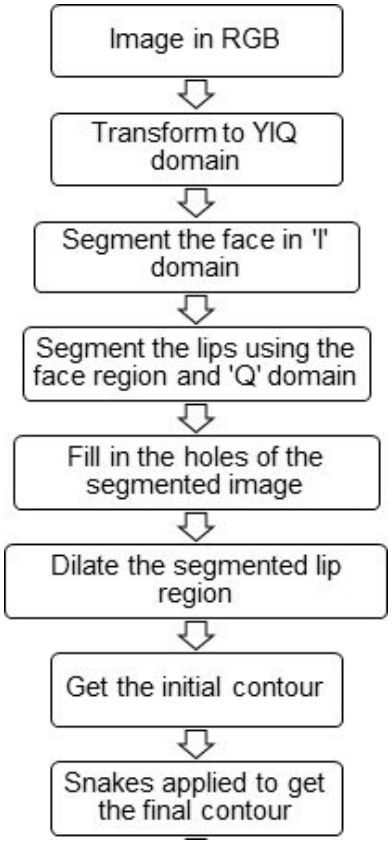

Fig 2.1 Algorithm for lip extraction

The Y component represents the luminance. It is the only component used by black-and-white television receivers. I and Q represent the chrominance information. The reason for using YIQ format is due to the observation which was performed on the face and lips of humans in different domains of colour representation.

It was found that the lips of a human are generally brighter in the 'Q' space and the face of a human is generally brighter in the 'I' space. Thus, face-detection is done in the 'I' space and the final lip-localization is done in the 'Q' domain. The detailed algorithm is shown in the Fig 2.1.

2.1.2 LABIAL SEGMENTATION METHOD

Two levels of approaches have been used in this method depending on the type of considerations made and data given for the algorithm.

The low-level approach (Image-based approaches) controlled by data, use directly the image of the mouth region. The working of this approach is based on the fact pixels on the lips have a different feature compared to that of ones in the skin. They do not allow to carry out a precise detection of the lip edges.

The high level approach (Model-based approaches), which is directed by physical extraction distance, uses a model. For example, consider the active edges, widely used in lip segmentation. These approaches also exploit the pixel information of the image, but they integrate regulatory constraints.

The problem of labial segmentation is to detect some POI on the lips and to track them throughout the speech sequence.

2.1.3 HYBRID METHOD

To overcome the problem faced with the labial segmentation method, a new hybrid method is built. This approach is applied on the first stage of the active contour method to automatically localize the lip feature points in the speaker's face. In the second stage, we propose a spatial-temporal tracking method of these points based on the Freeman coding directions and on voting techniques. This POI tracking will carry out visual information describing the lip movements among the locution of the video sequence.

2.2 VISEME CLASSIFICATION

In English linguistics a term called phoneme is used to describe the pronunciation of each character or letter. The lip movement which is extracted in the previous module tends to

map to a particular phoneme. One or more visemes can have the same phoneme.

2.2.1 ZERNIKE AND MFCC FEATURES

The main objective of this model is to improve audio-visual recognition accuracy. The proposed solution includes extracting visual features using Zernike moments and audio feature using Mel frequency cepstral coefficients on visual vocabulary of independent standard words dataset which contains a collection of an isolated set of city names of ten speakers.

Zernike features for each frame in visual utterance will be calculated and the result will be of the form of 9x1 columns. The visual utterance captured for two seconds results in formation of 52 frames therefore the Zernike features for one visual utterance results into 468x1 for single word. All words from data set are passed for visual feature extraction. This feature set is called as 'Training Set' and the dimension of this data set is to be reduced and to be interpreted using Principal Component Analysis. PCA will be applied on Zernike features to extract the most significant components of feature corresponding to the set of isolated words. PCA converts all of the original variables to be some independent linear set of variables. Those independent linear sets of variables possess the most information in the original data referred as principal components.

Recognition of uttered word is based on information in speech signal contained at the time of pronunciation of word. Mel-Frequency Cepstral Coefficients (MFCC) approach is the most popular because it uses spectral base as parameters for recognition. MFCC's are the coefficients, which represent audio based on perception of human auditory systems. The reason behind selection of MFCC for recognition purpose due to its peculiar difference between the operations of FFT/DCT. In the MFCC, the frequency bands are positioned logarithmically (on the Mel scale) which approximates the human auditory system response more closely than the linearly spaced frequency bands of FFT or DCT

The performance of recognition of isolated words based on visual only and audio only features results in 63.88 and 100% respectively. The advantage of this method is high accuracy. The main drawback is out of 36 samples 23 samples were correctly recognized and 13 samples were misclassified. It can be extended to a wide range of databases. It can be trained and tested with a huge variety of data.

2.2.2 VECTOR QUANTIZATION

This model uses vector quantization. Dynamic time warping, and a new heuristic distance measure. It significantly improved performance compared to the acoustic recognition

system alone. Speaker independent recognition can be done only by choosing a proper code book which can lead to recognition errors.

The visemes extracted are clustered using an iterative approach. With this, a successive approximation algorithm is also used to determine the new cluster distance. This process is continued until 255 clusters are created. Then entries are made in image code book. The mouth image code book is a table in which set of visemes are mapped to an index.

Once the mouth image codebook is created, inter-cluster distance table is created for fast vector quantized template matching. It is usually in the form of a two dimensional array where the index of each row represents the word or character and each row has a set of visemes corresponding to that particular word or character. The utterance of known words or characters are put in the image codebook. The utterance of new words or characters are matched with the closest possible image codebook .To optimize the performance the size of the code book and consistency need to be investigated.

In the future, the issues of arbitrary lighting, range and viewing angle must be investigated to free the speaker from the head-mounted camera. However, if a camera and lights could be mounted inside a pilots oxygen mask.

2.2.3 DEEP NEURAL NETWORKS

This model is built as a Speaker-independent lip-reading by using a combination of Maximum Likelihood Linear Transform (MLLT) followed by Speaker Adaptive Training (SAT). Dataset is trained using Deep Neural Network (DNN).

The visual features are trained using DNN to map to a phoneme. MLLT is used to find the linear transform of the input features. SAT is a technique used to normalize the variation in the acoustic features of different speakers while training the system. DNN is experimented to be promising for speaker independent lip reading despite the limited amount of training data and without the inclusion of a pre training stage .The best known results for speaker independent lip reading system is to use a hybrid system which uses MLLT followed by SAT. The performance of a “hybrid” Context-Dependent Deep Neural Networks (CD-DNN) where Context-Dependent Gaussian mixture model (CD-GMM) likelihoods in HMM are replaced by posterior probabilities of DNN after being converted into quasi-likelihoods is better.

With very few exceptions, performance increases with each stage for every speaker. Lower error rates for speaker independent tasks. The main demerit is the system is trained for smaller dataset. More training data will certainly improve results, but it labelled video data collection is still an expensive exercise—one possible improvement would be to collect video

data from opportunistic sources such as TV and video.

CHAPTER 3

SYSTEM REQUIREMENTS

All computer software to be used efficiently needs certain hardware components or other software resources to be present on a computer. These prerequisites are known as (computer) system requirements and are often used as a guideline as opposed to an absolute rule. Most software defines two sets of system requirements: minimum and recommended. With increasing demand for higher processing power and resources in newer versions of software, system requirements tend to increase over time. Industry analysts suggest that this trend plays a bigger part in driving upgrades to the existing computer systems than technological advancements.

3.1 SOFTWARE REQUIREMENT:

Operating System	: WINDOWS 10
Coding Language	: Python
Dataset	: shape_predictor_68_face_landmarks.dat , MIRACL-VC1
Packages	: dlib, skvideo.io, FFMpeg, cv2, keras

3.2 PACKAGES

3.2.1 CV2

CV2 package is aimed at Computer Vision. OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Being a BSD-licensed product, OpenCV makes it easy for businesses to utilize and modify the code.

The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc. OpenCV has more than 47 thousand people of the user community and the estimated number of downloads exceeding 14 million. The library is used extensively in companies, research groups and by governmental bodies.

This system uses OpenCV for lip feature extraction and marking. It is also used for grayscale conversion of images.

3.2.2 DLIB

Dlib is a modern C++ toolkit containing machine learning algorithms and tools for creating complex software in C++ to solve real-world problems. Python version of Dlib is available as Dlib.py. It is used in both industry and academia in a wide range of domains including robotics, embedded devices, mobile phones, and large high-performance computing environments. Dlib's open source licensing allows you to use it in any application, free of charge. Dlib’s image processing provides a means for matching our lip features from a particular frame.

Common Image Processing Functionalities in Dlib are

- Routines for reading and writing common image formats.
- Automatic color space conversion between various pixel types.
- Common image operations such as edge finding and morphological operations.
- Implementations of the feature extraction algorithms.
- Tools for detecting objects in images including frontal face detection and object pose estimation.
- High-quality face recognition.

Viseme Extraction module uses a frontal face detection algorithm from dlib to match lip features.

3.2.3 Skvideo.io and FFMpeg

Skvideo.io or scikit video is designed for easy video processing using Python. It is modeled in the spirit of other successful scikits such as scikit-learn and scikit-image. The developers of scikit-video know libraries exist for manipulating videos, such as FFmpeg, MoviePy, PyAV, imageIO, and OpenCV. However, no libraries have been found to provide an all-in-one solution for research-level video processing tools. This is used along with FFmpeg to manage videos files, also for parsing metadata from a video. It provides Reader and Writer functions to load the video and process them as individual frames, rather than entire video chunks. Reader parses the video as frames.

3.2.4 Keras

Keras is a high-level neural networks API, written in Python and capable of running on top of Tensorflow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result with the least possible delay is key to doing good research.

Keras is an API designed for human beings, not machines. It puts user experience front and center. Keras follows best practices for reducing cognitive load: it offers consistent & simple APIs, it minimizes the number of user actions required for common use cases, and it provides clear and actionable feedback upon user error.

A model is understood as a sequence or a graph of standalone, fully-configurable modules that can be plugged together with as few restrictions as possible. In particular, neural layers, cost functions, optimizers, initialization schemes, activation functions, regularization schemes are all standalone modules that can be combined to create new models.

New modules are simple to add (as new classes and functions), and existing modules provide ample examples. To be able to easily create new modules allows for total expressiveness, making Keras suitable for advanced research.

No separate models configuration files in a declarative format. Models are described in Python code, which is compact, easier to debug, and allows for ease of extensibility.

Keras can be used if a deep learning library is needed that:

- Allows for easy and fast prototyping (through user friendliness, modularity, and extensibility).
- Supports both convolutional networks and recurrent networks, as well as combinations of the two.

- Runs seamlessly on CPU and GPU.

3.2.5 CNN

A convolutional neural network (CNN) is very much related to the standard NN. CNN is old. They were developed in the late 1980's. The lack of processing power then meant that the implementation of such systems was not possible. In 1990's Yann LeCun in his paper "Gradient-based learning applied to document recognition" provided the first implementation of CNN.

In a deep learning network, each layer of a neural network has more layers on its own. Simply put, an artificial neural network (ANN) that uses the perceptron algorithm for supervised learning to analyze data is a CNN. CNN have applications ranging from natural language processing, image processing and even a lot of cognitive tasks.

ConvNets or CNN showed the most efficacy in image recognition. Thus ConvNets assures promising applications in deep learning systems. Computer vision through ConvNets has several applications like self-driving cars, robotics and so on.

CNN architecture is shaped into a hierarchical structure for fast feature extraction and classification. The main objective is to extract the input image volume and convert it into an output volume that holds class scores. A differential function is used for image processing. CNN consists of a stack of convolutional and subsampling layers which are followed by a series of neural networks.

3.3 DATASETS

3.3.1 Shape predictor

Dubbed as “shape_predictor_68_face_landmarks”, this is a trained dataset for dlib that is used for matching visemes. This provides means to match facial features. The interface is provided through predictor and detector classes from dlib package. The face detector used is made using the classic Histogram of Oriented Gradients (HOG) feature combined with a linear classifier, an image pyramid, and sliding window detection scheme.

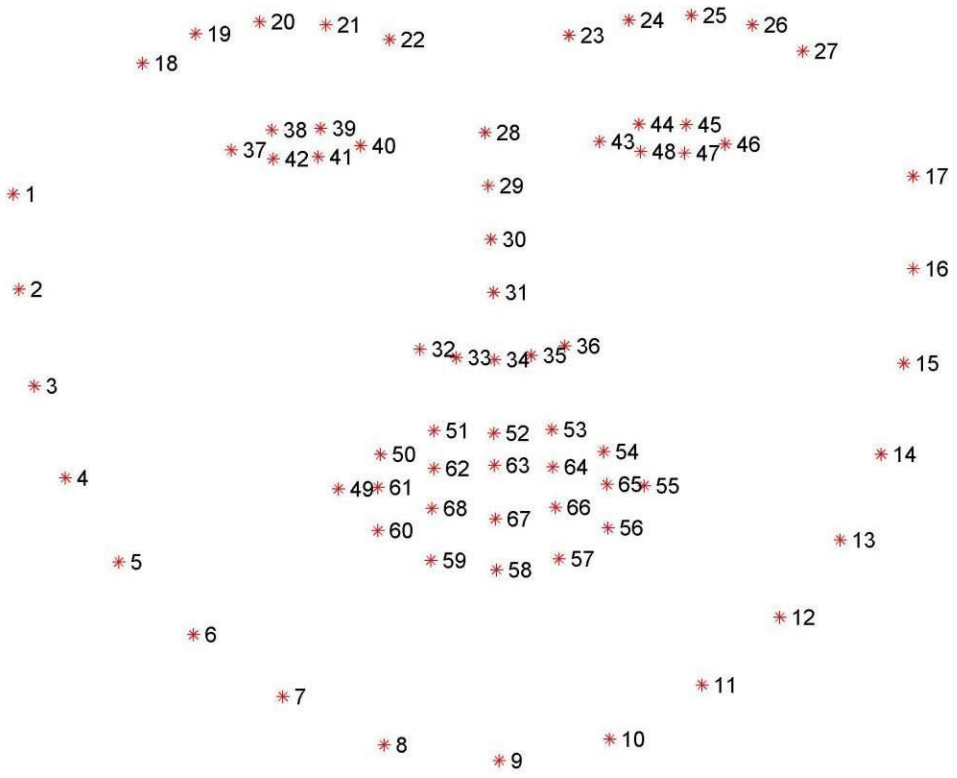

Fig. 3.1 68-point facial features

The landmark points from 48 to 68 are assumed to approximate the lip portion. So, those landmarks are considered as edge points while cropping.

3.3.2 MIRACL-VC1

MIRACL-VC1 is a lip-reading dataset including both depth and color images. It can be used for diverse research fields like visual speech recognition, face detection, and biometrics. Fifteen speakers (five men and ten women) positioned in the frustum of an MS Kinect sensor and utter ten times a set of ten words and ten phrases. Each instance of the dataset consists of a synchronized sequence of color and depth images (both of 640x480 pixels). The MIRACL-VC1 dataset contains a total number of 3000 instances (15 x 10 x 10). For this project, only the colour images are utilized.

ID	Word	ID	Phrase
01	Begin	01	Stop navigation
02	Choose	02	Excuse me
03	Connection	03	I am sorry
04	Navigation	04	Thankyou
05	Next	05	Good bye
06	Previous	06	I love this game
07	Start	07	Nice to meet you
08	Stop	08	You are welcome
09	Hello	09	How are you
10	web	10	Have a good time

Table 3.1 Words and Phrases available in MIRACL-VC1 dataset

Fig. 3.2 Sample colour and depth image frames

3.3 HARDWARE REQUIREMENTS

RAM: 16GB DDR4

PROCESSOR: INTEL CORE i7

GRAPHICS CARD: NVIDIA GTX 980M

CHAPTER 4

PROJECT DESCRIPTION

4.1 PROJECT AIM

Recently, there have been numerous advances in visual speech recognition (without neural nets) to perceive words under noisy conditions. The aim of the visual speech recognition system is to improve recognition accuracy. The visual component of speech is used for support of acoustic speech recognition. Most vital conditions for good lip-reading are quality of visual speech of a speaker (legitimate verbalization) and point of view.

4.2 PROPOSED SYSTEM

Our proposed system can be divided into two modules:

- Viseme extraction
- Viseme classification.

4.3 VISEME EXTRACTION

The algorithm takes video input. Facial features are matched using the dlib dataset. A predictor object is created from this dataset. Dlib provides a detector with a facial features class. This is called “frontal face detector”. A detector object is created.

The video is loaded. An FFMpeg reader object and FFMpeg writer object is created separately for the video. Packages required for parsing video is skvideo.io and the reader/writer objects are created using FFMpeg. The reader is capable of accessing the video as frames. Each frame can be processed separately.

Each frame is matched with the detector created earlier. Facial features existent in the frame are stored into detections. Detection is an array that holds the coordinates of all facial features. The coordinates of the lip lie between 48 and 68. So we parse those point and extract the endpoints. With endpoints provided, we can create a border box with a width say 30 and write it on the frame using the writer object and cv2 package. The lip features are converted to grayscale and stored as images. Thus we will have a lip image per frame. If there is no visible lip feature, the frame is marked as a null point.

4.4 VISEME CLASSIFICATION

Once the visemes are extracted, they are classified into respective text classes using a CNN trained with a rich dataset. The viseme extracted by DLIB is not ready for processing by the CNN. The dataset used for training (MIRACL-VC1) already has frames so that visemes are extracted from those frames. Features are to be generated out of these visemes and class labels (text IDs) so that CNN can process them. Feature generator generates instances which are the sequence of visemes and class labels.

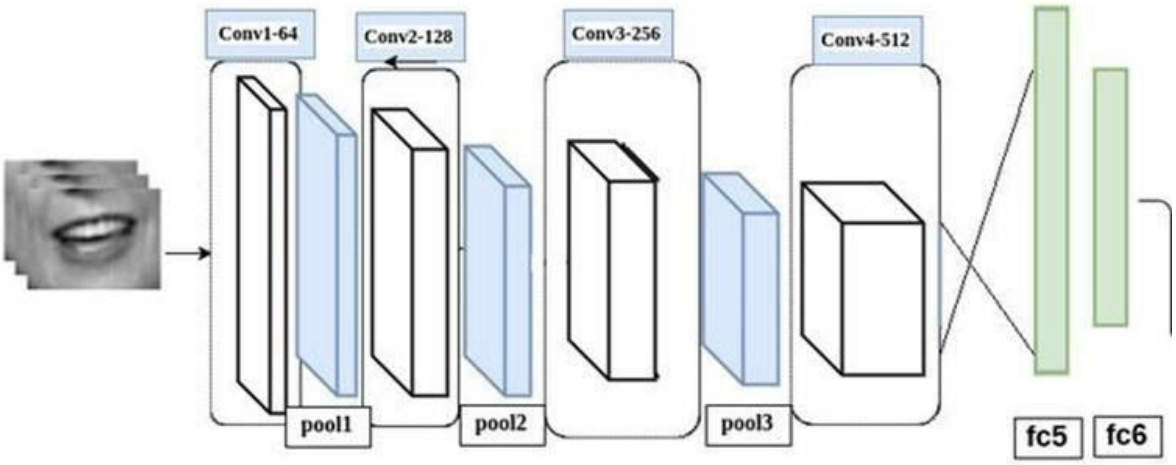

Fig. 4.1 3D-CNN architecture

The CNN is built using the 3D-CNN architecture referred to the one shown in Fig. 4.1, with some changes in the dimensions of the input and intermediate layers. The CNN processes the features, convolves, and finally arrives the weights that fit the model better. On testing, the CNN predicts the class labels of the visemes stacked up together.

CHAPTER 5

DESIGN AND IMPLEMENTATION

The first step is extracting lip features from the video. This is further given to a CNN that can classify visemes to corresponding phonemes. These two functionalities are separated into two modules, the extraction and processing stage.

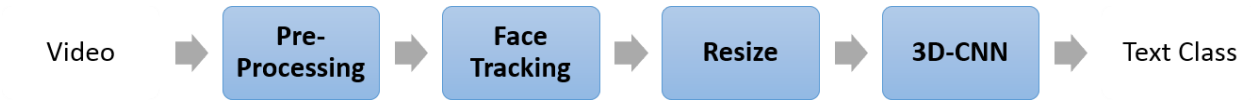

Fig. 5.1 Flow chart of proposed design

The flowchart shown in Fig. 5.1 is explained as follows:

5.1 PRE-PROCESSING

The visemes are extracted from each frame. So the video is broken into individual frames first. It is known that frame rate (number of frames per second) differs for each video. All videos before processing, are equalised to have a same frame rate say 30fps to make further processing easier. Now all the frames of the input video are extracted.

5.2 FACE TRACKING

Facial tracking obtains data about still images and video sequences by automatically tracking the facial landmarks. There are certain facial landmarks such as 48 face landmarks, 68 face landmarks etc. It involves two steps: Localisation of face, and Detection of key facial structures. Since we do not need all the points in the image, only the facial region is tracked and detected using 68 facial landmarks and use it for further processing. It is done using the *dlib*

package's frontal face detector and shape predictor functionalities. Using APIs saves time for training the CNN.

5.3 LOCALISATION OF FACE

A pre-trained HOG + Linear SVM Object Detector or deep learning based algorithms can be applied to localise face. The algorithm does not matter but obtaining the (x, y) coordinates of the face (bounding box) through some method.

5.4 DETECTION OF KEY FACIAL STRUCTURES

A variety of facial landmark detectors are available that try to effectively localise and label the following facial regions:

- Mouth
- Right eyebrow
- Left eyebrow
- Right eye
- Left eye
- Nose
- Jaw

The dlib library uses *One Millisecond Face Alignment with an Ensemble of Regression Trees* face detector from the paper by Kazemi and Sullivan (2014).
The method works as follows:

1. A set of labelled facial landmarks on an image are trained. These images are manually labelled, specifying specific (x, y) coordinates of regions surrounding each facial structure listed above.
2. Priors, of more specifically, the probability on distance between pairs of input pixels.

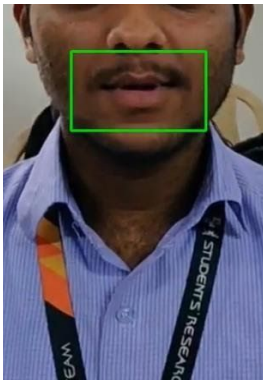

Fig. 5.2 Lip region bordered input video

Given this training data, an ensemble of regression trees are trained to estimate the facial landmark positions directly from the pixel intensities themselves (no feature extraction happens here).

5.5 RESIZING

The tracked facial images can be in any angle of view. The speaker would have spoken the phrases either by looking straight into the camera or by looking somewhere. This poses a difficulty in detecting facial features and some features may be lost. To resolve this problem, we can either restrict the speakers where to look or cropping and resizing the image so that it contains the lip region only. The face images may be in different sizes. So, the detected faces are clipped and resized into the same size which helps the CNN process them easily. It can be done by finding out the edge points of lip region and cropping the desired portion off the image.

Fig. 5.3 Cropped and resized visemes

5.6 CNN

The cropped images (visemes) are used as input for the CNN. But these visemes are not ready for processing. They are processed to have generated feature vectors (instances) from the visemes and class labels. From the hierarchical structure of the dataset, these visemes are easily converted to feature vectors. Since the dataset contains both words and phrases, the CNN is trained and tested only for words. Each utterance of the same word is made as an individual instance with the class label appended to the vector. The class label is the numerical ID corresponding to a word or phrase which can be mapped from the table provided (Table 3.1). The architecture of the CNN used is shown in Fig. 5.4.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1 (Conv3D)               (None, 13, 28, 46, 16)    448
_________________________________________________________________
pool1 (MaxPooling3D)         (None, 13, 13, 22, 16)    0
_________________________________________________________________
conv2 (Conv3D)               (None, 11, 11, 20, 32)    13856
_________________________________________________________________
pool2 (MaxPooling3D)         (None, 11, 5, 9, 32)      0
_________________________________________________________________
conv3 (Conv3D)               (None, 9, 3, 7, 64)       55360
_________________________________________________________________
pool3 (MaxPooling3D)         (None, 9, 1, 3, 64)       0
_________________________________________________________________
flatten (Flatten)            (None, 1728)              0
_________________________________________________________________
fc5 (Dense)                  (None, 250)               432250
_________________________________________________________________
fc6 (Dense)                  (None, 60)                15060
_________________________________________________________________
fc7 (Dense)                  (None, 10)                610
_________________________________________________________________
softmax (Activation)         (None, 10)                0
=================================================================
Total params: 517,584
Trainable params: 517,584
Non-trainable params: 0
_________________________________________________________________
None
```

Fig. 5.4 Architecture of CNN used

The architecture shown in Fig. 5.4 shown, contains three convolution layers each followed by a max pooling layer, with kernel size 3. The feature map is given to three fully connected layers of 250 units, 60 units and 10 units respectively.

CHAPTER 6
RESULT AND ANALYSIS

6.1 RESULT

Since the dataset has 1500 instances (10 words and 10 phrases each uttered 10 times by 10 speakers), a subset of the dataset is taken for the experiment. The subset contains the first person's utterances of words alone (150 instances). The frames are processed to have only the visemes which are resized to 48x30 to have equal width for all the visemes. Each utterance contains different number of frames (Fig 6.1) so that a dummy viseme of closed mouth image is appended to make all the utterances have same number of visemes.

Fig. 6.1 7 visemes of the second utterance of *Choose*

Now, all the instances have a sequence of 15 images of size 48x30. This is fed into the CNN built. 20% split is used for validation so that 120 out of 150 instances are used for training and the rest 30 instances are used for validation. Since this is a multi-class classification problem, categorical cross entropy loss function us used. The model is run for
10 epochs without batches. The train labels are one-hot encoded to make it easy for processing by CNN.

The dimension of train data and train labels are (150, 15, 30, 48, 1) and (150, 10) where 1 in train data dimension indicates the channel as grayscale. The model achieves an accuracy of 40.83% which is good compared to other visual speech recognition systems. This system, when coupled with an Audio Speech Recognition system will give a high accuracy for the text transcription.

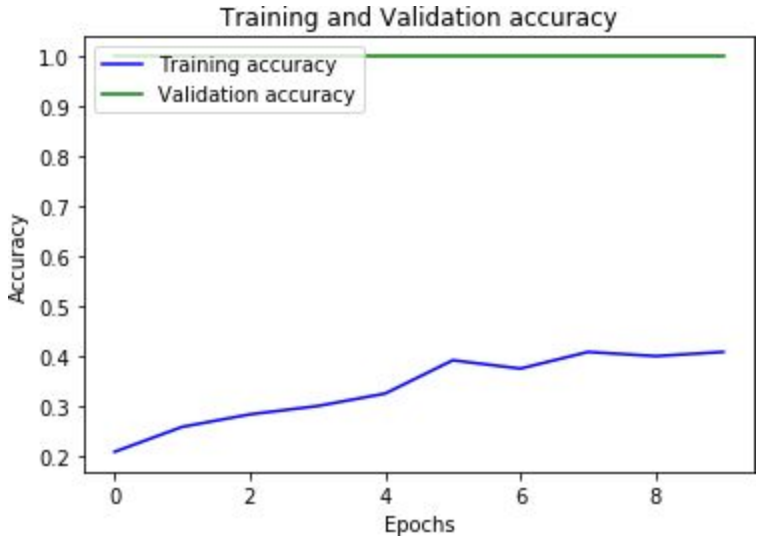

Fig. 6.2 Train and Validation accuracy

From the graph shown in Fig. 6.2, it is seen that the train accuracy increases over the epochs and validation accuracy increases gradually too.

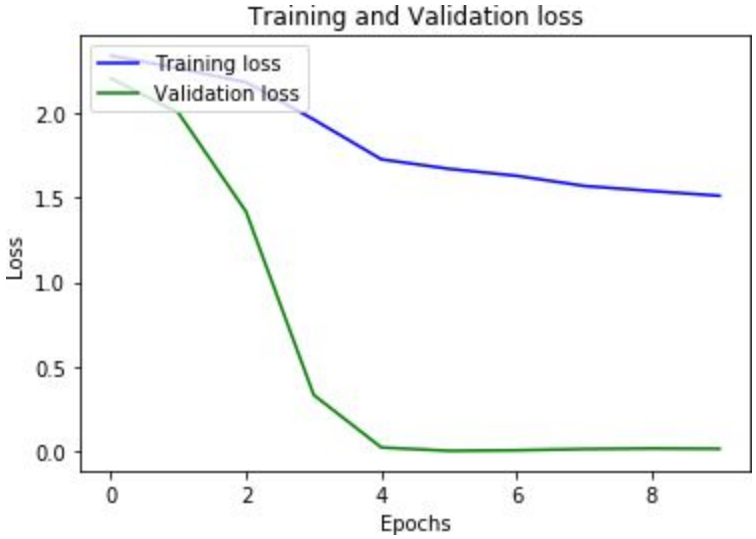

Fig. 6.3 Train and Validation loss

The loss graph (Fig. 6.3) shows how the loss is reduced for training data and validation data. The model is now speaker dependent since only one speaker's utterances are trained. So, it can be made speaker independent by training utterances of all the speakers in the dataset.

For testing, a part of training data itself is given which achieves a test accuracy of 64%. A sample predicted vs ground truth labels is shown below in Fig 6.4.

```
Predicted vs Ground truth labels
hello      =>    stop
hello      =>    stop
next       =>    stop
hello      =>    stop
hello      =>    stop
next       =>    stop
hello      =>    stop
hello      =>    stop
web        =>    stop
web        =>    hello
hello      =>    hello
choose     =>    hello
next       =>    hello
hello      =>    hello
hello      =>    hello
next       =>    hello
hello      =>    hello
```

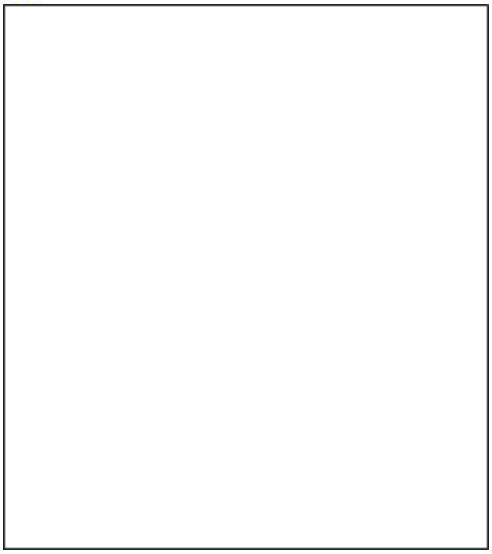

Fig. 6.4 Predicted vs ground truth labels

CONCLUSION AND FUTURE ENHANCEMENTS

Visual Speech Recognition paves for new possibilities such as speech recognition in loud areas. Captioning for the video content with video features can be done. But in a real-time scenario, there are several limitations for such a system. A system trained with one speaker may not work well with another due to several distinctions in facial features. Moreover the system will work only for visemes and phonemes pre-existing in the dataset.

A practical lip-reading system may not that efficient. Several factors limit this efficiency. Also, training a system provides a lot of complications. A multi-layer CNN architecture is used that may need a lot of memory and time to be trained.

To improve this efficiency, the system can be trained with multiple iterations of multiple speakers. Moving from CPU to GPU can improve the speed by several times. A richer dataset can provide a more accurate transcription. Typical dataset exist with utterances as words or as sentences. So for such a dataset, the system can be expected to match only that particular word or sentence grouped together.

Despite of all these limitations, a visual system can extract more data from video files. Also, this can be augmented with audio speech recognition systems, thus paving way for better Automatic Text Recognition systems. A system that can extract more information for a source is always preferred over another. With improved efficiency, transcriptions can be made a lot more reliable.

BIBLIOGRAPHY

REFERENCES

[1] Prashant Borde ,Amarsinh Varpe, Ramesh Manza , Pravin Yannawar, “Recognition of isolated words using Zernike and MFCC features for audio visual speech recognition”, *International Journal of Speech Technology (IJST)*, Volume 18 issue 2, 2015 [DOI 10.1007/s10772-014-9257-1].

[2] Eric Petajan, Bradford Bischoff and David Bodoff , N. Michael , “An Improved Automatic Lipreading System to Enhance Speech Recognition ”, *ACM SIGCHI*, Washington D.C ., United States, 15 - 19 May, 1998.

[3] Ibrahim Almajai, Stephen Cox, Richard Harvey, Yuxuan Lan, “Improved Speaker Independent Lip Reading Using Speaker Adaptive Training and Deep Neural Networks”, *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2016.

[4] Amirsina Torfi , Seyed Mehdi Iranmanesh, Nasser Nasrabadi, (FELLOW, IEEE), and Jeremy Dawson.,“3D Convolutional Neural Networks for Cross Audio-Visual Matching Recognition”, *IEEE*, October 2017 [DOI 0.1109/ACCESS.2017.2761539].

[5] Nazeeh Alothmany, Robert Boston, Ching Li, Susan Shaiman, John Durrant, “Classification of Visemes Using Visual Cues”, *IEEE*, 15 – 17 September, 2010.

[6] Ahmed Rekik, Achraf Ben-Hamadou, Walid Mahdi, “A New Visual Speech Recognition Approach For RGB-D Cameras”, *International Conference Image Analysis and Recognition (ICIAR)*, 2014.

APPENDIX I

LIP READING

lip_reading.py

```
from models.cnn3d import CNN
from preprocess.makefeatures import FeatureGen

#Dictionary mapping Word IDs to words
labels = {'01': 'begin',
        '02': 'choose',
        '03': 'connection',
        '04': 'navigation',
        '05': 'next',
        '06': 'previous',
        '07': 'start',
        '08': 'stop',
        '09': 'hello',
        '10': 'web'}

#Inputting and extracting features
input_path = 'path/to/speaker/words'
feat = FeatureGen(type_ = 'words')
(train_X, train_Y) = feat.make_features(input_path)
train_X = train_X.reshape(train_X.shape + (1, ))
test_X, test_Y = train_X[70:120], train_Y[70:120]
train_Y, label_encoder = feat.make_one_hot_encoding(train_Y)

#Training CNN
cnn = CNN()
cnn.build_net()
print(cnn.summary())
cnn.train(train_X, train_Y, epochs = 10, validation_split = 0.2)

#Testing CNN
pred = cnn.test(test_X)
print(pred)
print(pred.shape)
dec_pred = feat.decode_one_hot(pred, label_encoder).reshape(len(test_X))
dec_truth = feat.decode_one_hot(test_Y, label_encoder).reshape(len(test_X))
cnn.evaluate(test_X, test_Y) #Evaluating the performance of the result
print('Predicted vs Ground truth labels')
for t, g in zip(dec_pred, dec_truth):
    print('%-10s %-5s %-10s'%(labels[t], '=>', labels[g]))
```

FEATURE GENERATION (PRE-PROCESSING)

makefeatures.py from re
import match from scipy
import misc
from sklearn.preprocessing import LabelEncoder as sk_LabelEncoder, OneHotEncoder as sk_OneHotEncoder
from numpy import array as np_array
import numpy as np
import os

```
class FeatureGen:
    def init (self, type_):
        if type_ not in ('words', 'phrases'): raise
            ValueError('unsupported type')
        self.type = type_
        self.X = np_array(eval('[[[[0.0]*48]*30]*15]*150'))
        self.Y = np_array(eval('["01"]*150'))

    def get_color_images(self, images):
        return list(filter(lambda file: bool(match('^color_\d{3}\.jpg$', file)), images))

    def make_features(self, in_path, verbose = True): i =
        0
        cwd = os.getcwd()
        in_path += '/words' if self.type == 'words' else '/phrases'
        self.in_path = in_path
        os.chdir(in_path)
        contents = os.listdir(os.getcwd())
        print('Words' if self.type == 'words' else 'Phrases') for
        content in contents:
            j = 0
            if verbose == True:
                print('Word' if self.type == 'words' else 'Phrase', content)
            utterances = os.listdir('./' + content)
            for utterance in utterances: if
                verbose == True:
                    print('Reading utterance', utterance)
                counter = 0
                for image in self.get_color_images(os.listdir('./' + content + '/' + utterance)): self.X[i][j] =
                    misc.imresize(misc.imread('./' + content + '/' + utterance + '/' + image,
mode = 'F', flatten = True), size = (30, 48))/255
                    counter += 1
                if counter < 15:
                    self.X[i] = self.append_frames(self.X[i], 15, counter) j
                    += 1
                self.Y[i] = content i
                += 1
        os.chdir(cwd)
        return (self.X, self.Y)
```

```
    def append_frames(self, im_seq, max_frame, counter):
        dummy_viseme = misc.imresize(misc.imread(self.in_path + '/../dummy.jpg', mode = 'F', flatten
= True), size= (30, 48))/255
```

```
        for i in range(counter, max_frame):
            im_seq[i] = dummy_viseme
        return im_seq

    def make_one_hot_encoding(self, Y): label_encoder =
        sk_LabelEncoder() integer_encoded =
        label_encoder.fit_transform(Y)
        onehot_encoder = sk_OneHotEncoder(sparse = False) integer_encoded
        = integer_encoded.reshape(len(integer_encoded), 1) onehot_encoded =
        onehot_encoder.fit_transform(integer_encoded)
        """
        print('Label', label_encoder.inverse_transform([np.argmax(onehot_encoded[0, :])])) """
        return onehot_encoded, label_encoder

    def decode_one_hot(self, predictions, label_encoder):
        dec = []
        for i in range(len(predictions)):
            dec.append(label_encoder.inverse_transform([np.argmax(predictions[i, :])]))
        return np.array(dec)
```

3D-CNN

cnn3d.py

```
import keras
from keras.models import Sequential, load_model
from keras.layers import Dense, Activation, Flatten
from keras.layers import Conv3D, MaxPooling3D from
sklearn.metrics import classification_report import
matplotlib.pyplot as plt
from IPython import get_ipython
get_ipython().run_line_magic('matplotlib', 'inline')

class CNN:
    def init (self):
        print('---3D CNN for Visual Speech Recognition---')
        self.model = Sequential()
        self.history = None

    def build_net(self, load_name = None): if
        load_name:
            self.model = load_model('{}.h5'.format(load_name))
        else:
            self.model.add(Conv3D(name = 'conv1',
                                  filters = 16,
                                  kernel_size = 3,
                                  strides = 1,
                                  input_shape = (15, 30, 48, 1),
                                  data_format = 'channels_last',
                                  activation = 'relu'))
            self.model.add(MaxPooling3D(name = 'pool1',
```

```
                        pool_size = (1, 3, 3),
                        strides    =    (1,    2,    2)))
self.model.add(Conv3D(name = 'conv2',
```

```
                    filters = 32,
                    kernel_size = 3,
                    strides = 1,
                    data_format = 'channels_last',
                    activation = 'relu'))
        self.model.add(MaxPooling3D(name = 'pool2',
                        pool_size = (1, 3, 3),
                        strides = (1, 2, 2)))
        self.model.add(Conv3D(name = 'conv3',
                    filters = 64,
                    kernel_size = 3,
                    strides = 1,
                    data_format = 'channels_last',
                    activation = 'relu'))
        self.model.add(MaxPooling3D(name = 'pool3',
                        pool_size = (1, 3, 3),
                        strides = (1, 2, 2)))
        """self.model.add(Conv3D(name = 'conv4',
                    filters = 128,
                    kernel_size = 3,
                    strides = 1, activation
                    = 'relu'))
        """
        self.model.add(Flatten(name = 'flatten'))
        self.model.add(Dense(250, name = 'fc5'))
        self.model.add(Dense(60, name = 'fc6'))
        self.model.add(Dense(7, name = 'fc7'))
        self.model.add(Activation('softmax', name = 'softmax'))


    self.model.compile(loss = keras.losses.categorical_crossentropy,
                optimizer = keras.optimizers.Adam(),
                metrics = ['accuracy'])

def train(self, X, Y, epochs, validation_split, save = False, name = None):
    print('---Training CNN3D---')
    self.history = self.model.fit(x = X,
                        y = Y,
                        epochs = epochs,
                        validation_split = validation_split).history
    if save:
        self.model.save('{}.h5'.format(name))


    acc =  self.history['acc'] val_acc =
    self.history['val_acc'] loss =
    self.history['loss']
    val_loss = self.history['val_loss']
    epochs = list(range(len(acc)))
    plt.plot(epochs, acc, 'b', label = 'Training accuracy')
```

```
plt.plot(epochs, val_acc, 'g', label = 'Validation accuracy')
plt.title('Training and Validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend(loc = 'upper left')
```

```
        plt.figure()
        plt.plot(epochs, loss, 'b', label = 'Training loss')
        plt.plot(epochs, val_loss, 'g', label = 'Validation loss')
        plt.title('Training and Validation loss')
        plt.xlabel('Epochs')
        plt.ylabel('Loss') plt.legend(loc
        = 'upper left') plt.show()

    def test(self, X, steps = None):
        print('---Testing CNN3D---')
        predictions = self.model.predict(x = X,
                          steps = steps)
        print('Predictions: ', predictions.tolist())
        return predictions

    def evaluate(self, X, Y):
        print('---Evaluating CNN3D---')
        eval_ = self.model.evaluate(x = X, y = Y)
        print('Test loss:', eval_[0])
        print('Test accuracy:', eval_[1])

    def save_report(self, name, test_y, predictions, num_classes):
        target_names = ['Class {}:'.format(i) for i in range(1, num_classes)]
        with open(name, 'w') as report:
            report_ = classification_report(test_y, predictions, target_names = target_names)
            report.write(report_)
        print('Report saved successfully')

    def summary(self):
        return self.model.summary()
```

LIP FEATURE EXTRACTION

visualizelip.py import
numpy as np import
cv2
import dlib import
argparse import os
import skvideo.io

```
"""
PART1: Construct the argument parse and parse the arguments """
for word in range(1, 11): for
    i in range(1, 11):
        ap = argparse.ArgumentParser()
        ap.add_argument("-i", "--input", default = "path/to/speaker/F01/words/%02d/%02d/
words%02dutterance%02d.mp4"%(word, i, word, i),
                        help="path to input video file")
        ap.add_argument("-o", "--output", default = " path/to/speaker /F01/ words /%02d/%02d/
```

```
word%02dutterance%02dbordered.mp4"%(word, i, word, i),
                help="path to output video file")
    ap.add_argument("-f", "--fps", type=int, default=10,
                help="FPS of output video")
    ap.add_argument("-c", "--codec", type=str, default="MJPG",
                help="codec of output video")
    args = vars(ap.parse_args())

    """
    PART2: Calling and defining required parameters for:

        1 - Processing video for extracting each frame. 2 -
        Lip extraction from frames.
    """

    # Dlib requirements.
    predictor_path =
r'C:\Users\SandeepArockia\Desktop\Code\data2\shape_predictor_68_face_landmarks.dat'
    detector = dlib.get_frontal_face_detector()
    predictor = dlib.shape_predictor(predictor_path) mouth_destination_path =
    os.path.dirname(args["output"]) + '/' + 'mouth' if not
    os.path.exists(mouth_destination_path):
      os.makedirs(mouth_destination_path)

    inputparameters = {}
    outputparameters = {}
    reader = skvideo.io.FFmpegReader(args["input"],
                inputdict=inputparameters,
                outputdict=outputparameters)
    video_shape = reader.getShape()
    (num_frames, h, w, c) = video_shape
    print(num_frames, h, w, c)

    activation = []
    max_counter = 300
    total_num_frames = int(video_shape[0]) num_frames =
    min(total_num_frames,max_counter) counter = 0
    font = cv2.FONT_HERSHEY_SIMPLEX

    writer = skvideo.io.FFmpegWriter(args["output"])


    width_crop_max = 0
    height_crop_max = 0

    for frame in reader.nextFrame():
       print('frame_shape:', frame.shape)

       if counter > num_frames:
          break
```

```
detections = detector(frame, 1)
marks = np.zeros((2, 20))
Features_Abnormal = np.zeros((190, 1))
```

```
print(len(detections)) if
len(detections) > 0:
    for k, d in enumerate(detections):
        print("Location:" , d)
        shape = predictor(frame, d)

        co = 0
        for ii in range(48, 68): X
            = shape.part(ii) A =
            (X.x, X.y)
            marks[0,   co]   =   X.x
            marks[1, co] = X.y co
            += 1
```

```
X_left, Y_left, X_right, Y_right =
[int(np.amin(marks, axis=1)[0]), int(np.amin(marks,
axis=1)[1]),
                                                    int(np.amax(marks, axis=1)[0]),
area).
                                                    int(
```

```
np.amax(marks,axis=1)[1])]
```

```
X_center=(X_left+X_right)/2.0
```

```
Y_center=(Y_left+Y_right)/2.0
border=10
X_left_new=X_left
```

-
b
o
r
d
e
r
Y
_
l
e
f
t
_
n
e
w
=
Y
_
l
e
f
t
-
b
o
r
d
e
r
X
_
r
i
g
h
t
_
n
e
w
=
X
_
r
i
g
h
t
+
b
o
r
d
e
r
Y
_
r
i
g
h
t
_
n
e
w
=
Y
_
r
i
g
h
t
+
b
o
r
d
e
r

Width and height for cropping(before and after considering the border). width_new = X_right_new - X_left_new
hei
ght
_n
ew
=
Y_r
ight
_n
ew
-
Y_l
eft
_n
ew
wid
th_
cur
ren
t =
X_r
ight
-

```
X_l
eft
hei
ght
_cu
rre
nt
=
Y_r
ight
-
Y_l
eft
# Determine the cropping rectangle
dimensions(the main purpose is to have a
fixed if width_crop_max == 0 and
height_crop_max == 0:
    w
    i
    d
    t
    h
    _
    c
    r
    o
    p
    _
    m
    a
    x
    =
    w
    i
    d
    t
    h
    _
    n
    e
    w
    h
    e
    i
    g
    h
    t
    _
mouth_gray)
    c
    r
    o
    p
    _
    m
    a
    x
    =
    h
    e
    i
    g
    h
    t
    _
    n
    e
    w
else:
    width_crop_max += 1.5 *
    np.maximum(width_current -
    width_crop_max, 0) height_crop_max +=
    1.5 * np.maximum(height_current -
    height_crop_max, 0)

X_left_crop =
int(X_center -
width_crop_max
/ 2.0)
X_right_crop =
int(X_center +
width_crop_max
/ 2.0)
Y_left_crop =
int(Y_center -
height_crop_ma
x / 2.0)
Y_right_crop =
int(Y_center +
height_crop_ma
x / 2.0)

if X_left_crop >= 0 and Y_left_crop >= 0 and
    X_right_crop < w and Y_right_crop < h:
    mouth = frame[Y_left_crop:Y_right_crop,
    X_left_crop:X_right_crop, :]


mouth_gray = cv2.cvtColor(mouth,
cv2.COLOR_RGB2GRAY)
cv2.imwrite(mouth_destination_path + '/' +
'color_' + '_' + '%03d.jpg'%(counter+1),
```

```
print("The cropped mouth is detected ...")
activation.append(1)
```

```
                                                    else:
                                                        cv2.putText(frame, 'The full mouth is not
                                                        detectable. ', (30, 30), font, 1, (0, 255,
255), 2)

                                                        print("The full mouth is not detectable.    ")
                                                        activation.append(0)


            else:
                cv2.putText(frame, 'Mouth is not detectable. ', (30, 30), font, 1, (0, 0, 255), 2)
                print("Mouth is not detectable     ")
                activation.append(0)


            if activation[counter] == 1:
                cv2.rectangle(frame, (X_left_crop, Y_left_crop), (X_right_crop, Y_right_crop), (0, 255, 0),
2)


            print('frame number %d of %d' % (counter, num_frames))
            print("writing frame %d with activation %d" % (counter + 1, activation[counter]))
            writer.writeFrame(frame)
            counter += 1
        writer.close()
```