THE FULFILLMENT OF THE TWO-WEEK INTERNSHIP PROGRAM DEGREE

ON

Sign Language Translation

OF

Bachelor of Technology

IN

COMPUTER SCIENCE AND ENGINEERING

(DATA SCIENCE)

By

K.KAVYA SREE - 20951A6719

Under the guidance of

Chief Mentor: Dr. C V R Padmaja

Co-Ordinator Name: Dr. Indu

Career Development Centre

INSTITUTE OF AERONAUTICAL ENGINEERING

DUNDIGAL

MAY 2023

Career Development Centre

INSTITUTE OF AERONAUTICAL ENGINEERING

DUNDIGAL MAY 2023

DECLARATION BY THE CANDIDATE

KYANAM. KAVYA SREE bearing **Roll number: 20951A6719** hereby declare that the project report entitled **“Sign Language Translation”**, is a record of bonafide work carried out by me and the results embodied in this project have not been reproduced or copied from any source. The results of this project report have not been submitted to any other University or Institute for the award of any other Degree or Diploma.

K.KAVYA SREE

20951A6719

Institute of Aeronautical Engineering

affiliated to Jawaharlal Nehru Technological University (JNTUH), Hyderabad

Dundigal Road, Dundigal, Hyderabad, Telangana 500043

CERTIFICATE

This is to certify that the project report entitled **"Sign Language Translation"**, submitted by **K.KAVYA SREE** bearing **Roll. No.: 20951A6719**, in the partial fulfillment of the requirements for the award of the degree of **Bachelor of Computer Science and Engineering** is a record of bonafide work carried out by her for the course **Project**.

<signature>	<signature>
Dr. C V R Padmaja	Dr. Indu
Mentor	Co-Ordinator
Department of CSE	Department of CSE

Countersignature of HOD with seal May 2023

INDEX

CONTENTS

CHAPTER 1
INTRODUCTION

1.1 INTRODUCTION

Individuals who are unable to speak face several significant challenges, including the limited ability to express their emotions freely. They are unable to benefit from voice TRANSLATION and voice search systems available on smartphones[1]. Consequently, they are unable to utilize voice-controlled personal assistants like Google Assistant or Apple's Siri[2]. Platforms that address the requirements of these individuals are urgently required. American Sign Language (ASL) is a complex and detailed form of communication that incorporates hand movements, facial expressions, and body postures. It serves as the primary means of communication for many North Americans who are unable to speak and is one of several alternatives used by the deaf or hard-of-hearing community[3].

While sign language is crucial for effective communication among the deaf and mute population, it often receives inadequate attention from the general public. The significance of sign language tends to be overlooked unless there are specific concerns related to individuals who are deaf or mute. One viable solution for communicating with the deaf and mute community is by employing sign language techniques.

Hand gestures are commonly employed in sign language as a form of non-verbal communication. They are particularly used by individuals who are deaf and mute and have hearing or speech impairments to interact with others. Numerous sign language systems have been developed by various manufacturers worldwide, but they often lack flexibility and affordability for end users.

1.2 SCOPE

Instead of relying on technologies like gloves or Kinect,I aim to solve this problem using state-of-the-art computer vision and machine learning algorithms. The application will consist of two core modules: one that detects the gesture and displays the corresponding alphabet, and another that stores scanned frames in a buffer, enabling the formation of meaningful words after a certain interval.

One approach to facilitate communication with individuals who are deaf-mute is by utilizing the services of a sign language interpreter. However, this option can be costly[3]. Therefore, there is a need for a cost-effective solution that enables seamless communication between deaf-mute individuals and those who can speak normally[3]

Furthermore,I will provide an add-on feature that allows users to create their own custom gestures for special characters

MY strategy involves the development of an application that can recognize pre-defined American Sign Language (ASL) gestures. To detect these gestures,I will utilize a basic hardware component such as a camera, along with the necessary interfacing. The application will be built on the PyQt5 module, offering a comprehensive and user-friendly system.

1.3 PROBLEM STATEMENT

Given a hand gesture, implementing such an application which detects pre-defined American sign language (ASL) in a real time through hand gestures and providing facility for the user to be able to store the result of the character detected in a txt file, so that the problems faced by people who aren't able to talk vocally can be accommodated with technological assistance and the barrier of expressing can be overshadowed.

CHAPTER 2

SYSTEM SPECIFICATION

2.1 SYSTEM REQUIREMENT

2.1.1 HARDWARE REQUIREMENTS

- Intel Core i4 4th gen processor or later.
- 512 MB RAM.
- 512 MB disk space.
- Any external or inbuild camera with minimum pixel resolution 200 x 200 (300ppi or 150lpi) 4-megapixel cameras and up.

2.1.2 SOFTWARE REQUIREMENTS

- Microsoft Windows XP or later / Ubuntu 12.0 LTS or later /MAC OS 10.1 or later.
- Python Interpreter (3.6).
- TensorFlow framework, Keras API.
- PyQT5, Tkinter module.
- Python OpenCV2, scipy, qimage2ndarray, winGuiAuto, pypiwin32, sys, keyboard, pyttsx3, pillow libraries.

2.2 SYSTEM FEATURES

- User-friendly based GUI built using industrial standard PyQT5.
- Real time American standard character detection based on gesture made by user.
- Customized gesture generation.
- Forming a stream of sentences based on the gesture made after a certain interval of time.

CHAPTER 3

SYSTEM DESIGN

3.1 SYSTEM ARCHITECTURE

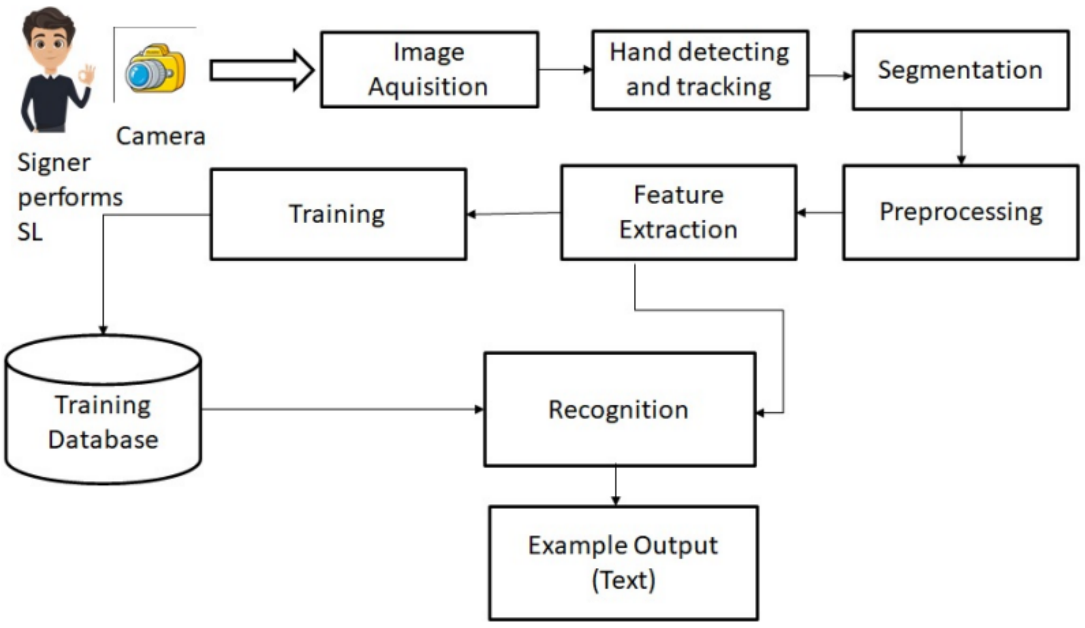

Fig 1: System Architecture for Sign Language Translation Using Hand Gestures.

3.2 MODULES IN THE SYSTEM

- **Data Pre-Processing** - This module involves generating binary images based on the object detected in front of the camera. The object is filled with solid white, while the background is filled with solid black. The pixel regions are assigned numerical values of either 0 or 1 for further processing in subsequent modules.

- **Scan Single Gesture** - A gesture scanner is positioned in front of the user, requiring them to perform a hand gesture. Based on the output of the pre-processed module, the user can view the assigned label associated with each hand gesture. These labels follow the predefined American Sign Language (ASL) standard and are displayed on the output window screen.

- **Formation of a Sentence** - Users have the ability to select a delimiter. As long as the selected delimiter is not encountered, each scanned gesture character is appended to the previous results, creating a meaningful stream of words and sentences.

- **Exporting** - Users can export the scanned character results into an ASCII standard textual file format.

3.3 USE CASE DIAGRAM

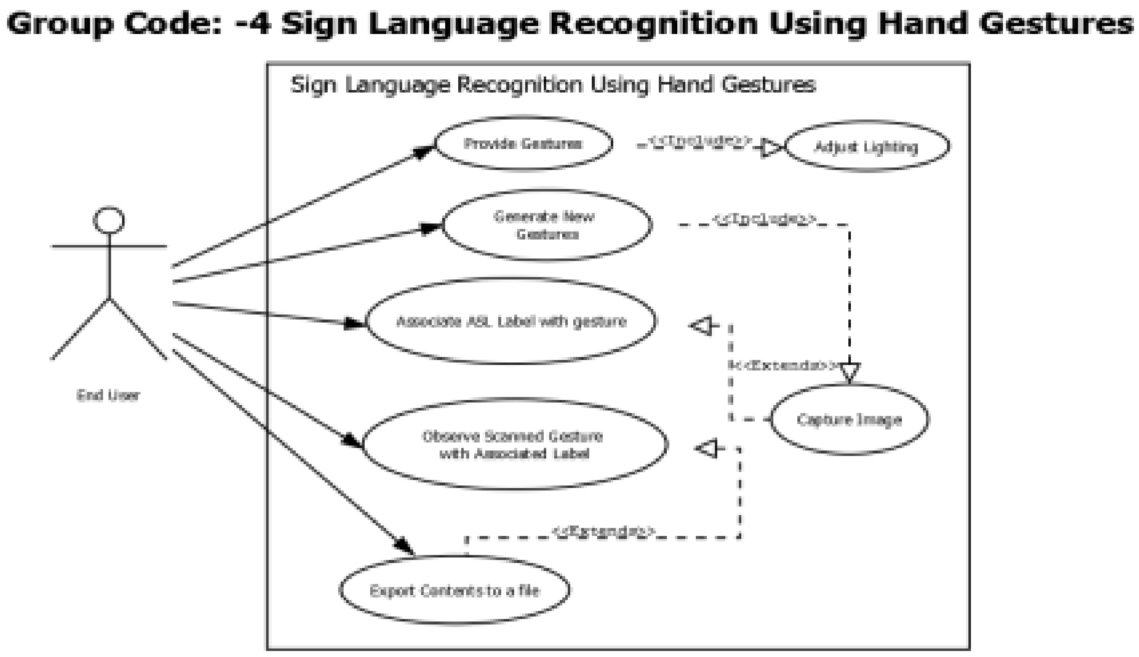
Group Code: -4 Sign Language Recognition Using Hand Gestures
Sign Language Recognition Using Hand Gestures
Provide Gestures
<<Include>>
Adjust Lighting
Generate New Gestures
<<Include>>
Associate ASL Label with gesture
<<Extends>>
Capture Image
Observe Scanned Gesture with Associated Label
<<Extends>>
Export Contents to a file
End User

Fig 2: Use Case Diagram for Sign Language TRANSLATION Using Hand Gestures.

3.4 ACTIVITY DIAGRAM

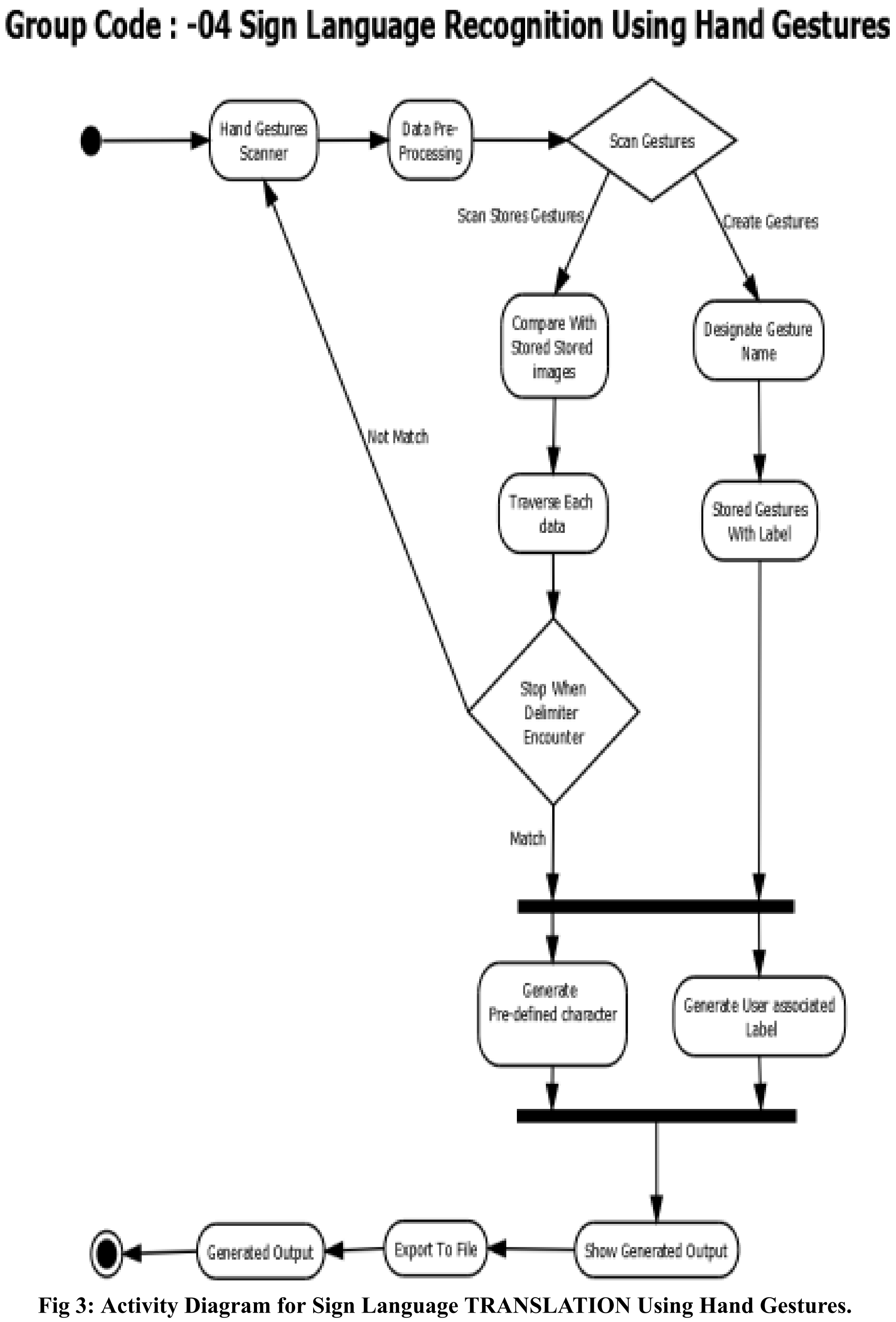
Group Code : -04 Sign Language Recognition Using Hand Gestures
Hand Gestures Scanner
Data Pre-Processing
Scan Gestures
Scan Stores Gestures
Create Gestures
Compare With Stored Stored images
Designate Gesture Name
Not Match
Traverse Each data
Stored Gestures With Label
Stop When Delimiter Encounter
Match
Generate Pre-defined character
Generate User associated Label
Show Generated Output
Export To File
Generated Output

Fig 3: Activity Diagram for Sign Language TRANSLATION Using Hand Gestures.

CHAPTER 4
IMPLEMENTATION

4.1 CODE SNIPPETS

Dashboard.py

```
from PyQt5 import QtWidgets, uic
from PyQt5.QtWidgets import QMessageBox
from PyQt5.QtCore import QUrl
from PyQt5.QtGui import QImage
from PyQt5.QtGui import QPixmap
from PyQt5 import QtCore                                         #importing pyqt5
librarie
from PyQt5.QtCore import QTimer,Qt
from PyQt5 import QtGui
from tkinter import filedialog                                   #for file export
module
from tkinter import *
import tkinter as tk
from matplotlib import pyplot as plt                     #for gesture viewer
from matplotlib.widgets import Button
import sys                                                                     #for
pyqt
import os                                                                      #for
removal of files
import cv2                                                                     #for
the camera operations
import numpy as np                                                     #proceesing
on images
import qimage2ndarray                                                  #convers
images into matrix
import win32api
import winGuiAuto
import win32gui
import win32con                                                                #for
removing title cv2 window and always on top
import keyboard                                                                #for
pressing keys
import pyttsx3                                                         #for tts
assistance
import shutil
import imageio                                    #will help in reading the images
                                    #for removal of directories
index = 0                                                                      #index
used for gesture viewer
engine = pyttsx3.init()                                                #engine
initialization for audio tts assistance

def nothing(x):
       pass

image_x, image_y = 64,64                                         #image resolution

from keras.models import load_model
classifier = load_model('ASLModel.h5')                   #loading the model

def fileSearch():
    """Searches each file ending with .png in SampleGestures directory so that custom
gesture could be passed to predictor() function"""
    fileEntry = []
    for file in os.listdir("SampleGestures"):
        if file.endswith(".png"):
            fileEntry.append(file)
    return fileEntry

def load_images_from_folder(folder):
    """Searches each images in a specified directory"""
    images = []
    for filename in os.listdir(folder):
        img = imageio.imread(os.path.join(folder,filename))
        if img is not None:
            images.append(img)
    return images

def toggle_imagesfwd(event):
       """displays next images act as a gesutre viewer"""
       img=load_images_from_folder('TempGest/')
       global index

       index += 1

       try:
              if index < len(img):
                     plt.axes()
                     plt.imshow(img[index])
                     plt.draw()
       except:
              pass

def toggle_imagesrev(event):
       """displays previous images act as a gesutre viewer"""
       img=load_images_from_folder('TempGest/')
       global index

       index -= 1

       try:
              if index < len(img) and index>=0:
                     plt.axes()
                     plt.imshow(img[index])
                     plt.draw()
       except:
              pass

def openimg():
       """displays predefined gesture images at right most window"""
       cv2.namedWindow("Image", cv2.WINDOW_NORMAL )
       image = imageio.imread('template.png')
       cv2.imshow("Image",image)
       cv2.setWindowProperty("Image",cv2.WND_PROP_FULLSCREEN,cv2.WINDOW_FULLSCREEN)
       cv2.resizeWindow("Image",298,430)
       cv2.moveWindow("Image", 1052,214)


def removeFile():
       """Removes the temp.txt and tempgest directory if any stop button is pressed
oor application is closed"""
       try:
              os.remove("temp.txt")
       except:
              pass
       try:
              shutil.rmtree("TempGest")
       except:
              pass

def clearfunc(cam):
       """shut downs the opened camera and calls removeFile() Func"""
       cam.release()
       cv2.destroyAllWindows()
       removeFile()

def clearfunc2(cam):
       """shut downs the opened camera"""
       cam.release()
       cv2.destroyAllWindows()

def saveBuff(self,cam,finalBuffer):
       """Save the file as temp.txt if save button is pressed in sentence formation
through gui"""
       cam.release()
       cv2.destroyAllWindows()
```

```
        if(len(finalBuffer)>=1):
                f=open("temp.txt","w")
                for i in finalBuffer:
                        f.write(i)
                f.close()


def capture_images(self,cam,saveimg,mask):
        """Saves the images for custom gestures if button is pressed in custom gesture
generationn through gui"""
        cam.release()
        cv2.destroyAllWindows()
        if not os.path.exists('./SampleGestures'):
                os.mkdir('./SampleGestures')

        gesname=saveimg[-1]
        if(len(gesname)>=1):
                img_name = "./SampleGestures/"+"{}.png".format(str(gesname))
                save_img = cv2.resize(mask, (image_x, image_y))
                cv2.imwrite(img_name, save_img)


def controlTimer(self):
        # if timer is stopped
        self.timer.isActive()
        # create video capture
        self.cam = cv2.VideoCapture(0)
        # start timer
        self.timer.start(20)


def predictor():
        """ Depending on model loaded and custom gesture saved prediction is made by
checking array or through SiFt algo"""
        import numpy as np
        from keras.preprocessing import image
        from keras.utils import load_img, img_to_array
        test_image = load_img('1.png', target_size=(64, 64))
        test_image = img_to_array(test_image)
        test_image = np.expand_dims(test_image, axis=0)
        result = classifier.predict(test_image)
        gesname = ''
        fileEntry = fileSearch()
        for i in range(len(fileEntry)):
                image_to_compare = imageio.imread("./SampleGestures/" + fileEntry[i])
                original = imageio.imread("1.png")
                sift = cv2.xfeatures2d.SIFT_create()
                kp_1, desc_1 = sift.detectAndCompute(original, None)
                kp_2, desc_2 = sift.detectAndCompute(image_to_compare, None)

                index_params = dict(algorithm=0, trees=5)
                search_params = dict()
                flann = cv2.FlannBasedMatcher(index_params, search_params)

                matches = flann.knnMatch(desc_1, desc_2, k=2)

                good_points = []
                ratio = 0.6
                for m, n in matches:
                        if m.distance < ratio * n.distance:
                                good_points.append(m)
                if abs(len(good_points) + len(matches)) > 20:
                        gesname = fileEntry[i]
                        gesname = gesname.replace('.png', '')
                        if gesname == 'sp':
                                gesname = ' '
                        return gesname

        if result[0][0] == 1:
                return 'A'
        elif result[0][1] == 1:
                return 'B'
        elif result[0][2] == 1:
                return 'C'
        elif result[0][3] == 1:
                return 'D'
        elif result[0][4] == 1:
                return 'E'
        elif result[0][5] == 1:
                return 'F'
        elif result[0][6] == 1:
                return 'G'
        elif result[0][7] == 1:
                return 'H'
        elif result[0][8] == 1:
                return 'I'
        elif result[0][9] == 1:
                return 'J'
        elif result[0][10] == 1:
                return 'K'
        elif result[0][11] == 1:
                return 'L'
        elif result[0][12] == 1:
                return 'M'
        elif result[0][13] == 1:
                return 'N'
        elif result[0][14] == 1:
                return 'O'
        elif result[0][15] == 1:
                return 'P'
        elif result[0][16] == 1:
                return 'Q'
        elif result[0][17] == 1:
                return 'R'
        elif result[0][18] == 1:
                return 'S'
        elif result[0][19] == 1:
                return 'T'
        elif result[0][20] == 1:
                return 'U'
        elif result[0][21] == 1:
                return 'V'
        elif result[0][22] == 1:
                return 'W'
        elif result[0][23] == 1:
                return 'X'
        elif result[0][24] == 1:
                return 'Y'
        elif result[0][25] == 1:
                return 'Z'


def checkFile():
        """retrieve the content of temp.txt for export module """
        checkfile=os.path.isfile('temp.txt')
        if(checkfile==True):
                fr=open("temp.txt","r")
                content=fr.read()
                fr.close()
        else:
                content="No Content Available"
        return content

class Dashboard(QtWidgets.QMainWindow):
        def __init__(self):
                super(Dashboard, self).__init__()
                self.setWindowFlags(QtCore.Qt.WindowMinimizeButtonHint)
                cap = cv2.VideoCapture('gestfinal2.min.mp4')

                # Read until video is completed
                while(cap.isOpened()):
                        ret, frame = cap.read()
                        if ret == True:
                # Capture frame-by-frame
                                ret, frame = cap.read()
                                cv2.namedWindow("mask", cv2.WINDOW_NORMAL)
                                cv2.imshow("mask", frame)

cv2.setWindowProperty("mask",cv2.WND_PROP_FULLSCREEN,cv2.WINDOW_FULLSCREEN)
                                cv2.resizeWindow("mask",720,400)
                                cv2.moveWindow("mask", 320,220)
                                if cv2.waitKey(25) & 0xFF == ord('q'):
                                        break

                        else:
                                break

                # When everything done, release
                cap.release()

                # Closes all the frames
                cv2.destroyAllWindows()
                self.setWindowIcon(QtGui.QIcon('icons/windowLogo.png'))
                self.title = 'Sign language Recognition'
                uic.loadUi('UI_Files/dash.ui', self)
                self.setWindowTitle(self.title)
                self.timer = QTimer()
                self.create.clicked.connect(self.createGest)
                self.exp2.clicked.connect(self.exportFile)
                self.scan_sen.clicked.connect(self.scanSent)
                if(self.scan_sinlge.clicked.connect(self.scanSingle)==True):
                        self.timer.timeout.connect(self.scanSingle)
                self.create.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
                self.scan_sen.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
                self.scan_sinlge.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
                self.exp2.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
                self.exit_button.clicked.connect(self.quitApplication)
                self._layout = self.layout()
                self.label_3 = QtWidgets.QLabel()
                movie = QtGui.QMovie("icons/dashAnimation.gif")
                self.label_3.setMovie(movie)
                self.label_3.setGeometry(0,160,780,441)
                movie.start()
                self._layout.addWidget(self.label_3)
                self.setObjectName('Message_Window')

        def quitApplication(self):
                """shutsdown the GUI window along with removal of files"""
                userReply = QMessageBox.question(self, 'Quit Application', "Are you
sure you want to quit this app?", QMessageBox.Yes | QMessageBox.No, QMessageBox.No)
                if userReply == QMessageBox.Yes:
                        removeFile()
                        keyboard.press_and_release('alt+F4')

        def createGest(self):
                """ Custom gesture generation module"""
                try:
                        clearfunc(self.cam)
                except:
                        pass
                gesname=""
                uic.loadUi('UI_Files/create_gest.ui', self)
                self.setWindowTitle(self.title)
                self.create.clicked.connect(self.createGest)
                self.exp2.clicked.connect(self.exportFile)
                if(self.scan_sen.clicked.connect(self.scanSent)):
                        controlTimer(self)
                self.scan_sinlge.clicked.connect(self.scanSingle)
                self.linkButton.clicked.connect(openimg)
                self.create.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
                self.scan_sen.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
                self.scan_sinlge.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
                self.exp2.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
                self.pushButton_2.clicked.connect(lambda:clearfunc(self.cam))
                try:
                        self.exit_button.clicked.connect(lambda:clearfunc(self.cam))
                except:
                        pass
                self.exit_button.clicked.connect(self.quitApplication)
                self.plainTextEdit.setPlaceholderText("Enter Gesture Name Here")
                img_text = ''
                saveimg=[]
                while True:
                        ret, frame = self.cam.read()
                        frame = cv2.flip(frame,1)
                        try:
                                frame=cv2.resize(frame,(321,270))
                                frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
                                img2 = cv2.rectangle(frame, (150,50),(300,200),
(0,255,0), thickness=2, lineType=8, shift=0)
                        except:
                                keyboard.press_and_release('esc')

                        height2, width2, channel2 = img2.shape
                        step2 = channel2 * width2
                # create QImage from image
                        qImg2 = QImage(img2.data, width2, height2, step2,
QImage.Format_RGB888)
                # show image in img_label
                        try:
                                self.label_3.setPixmap(QPixmap.fromImage(qImg2))
                                slider2=self.trackbar.value()
                        except:
                                pass

                        lower_blue = np.array([0, 0, 0])
                        upper_blue = np.array([179, 255, slider2])
                        imcrop = img2[52:198, 152:298]
                        hsv = cv2.cvtColor(imcrop, cv2.COLOR_BGR2HSV)
                        mask = cv2.inRange(hsv, lower_blue, upper_blue)

                        cv2.namedWindow("mask", cv2.WINDOW_NORMAL )
                        cv2.imshow("mask", mask)

cv2.setWindowProperty("mask",cv2.WND_PROP_FULLSCREEN,cv2.WINDOW_FULLSCREEN)
                        cv2.resizeWindow("mask",170,160)
                        cv2.moveWindow("mask", 766,271)

                        hwnd = winGuiAuto.findTopWindow("mask")
                        win32gui.SetWindowPos(hwnd, win32con.HWND_TOP,
0,0,0,0,win32con.SWP_NOMOVE | win32con.SWP_NOSIZE | win32con.SWP_NOACTIVATE)

                        try:
                                ges_name = self.plainTextEdit.toPlainText()
                        except:
                                pass
                        if(len(ges_name)>=1):
                                saveimg.append(ges_name)
                        else:
                                saveimg.append(ges_name)
                                ges_name=''

                        try:

self.pushButton.clicked.connect(lambda:capture_images(self,self.cam,saveimg,mask))
                        except:
                                pass

                        gesname=saveimg[-1]

                        if keyboard.is_pressed('shift+s'):
                                if not os.path.exists('./SampleGestures'):
                                        os.mkdir('./SampleGestures')
                                if(len(gesname)>=1):
                                        img_name =
"./SampleGestures/"+"{}.png".format(str(gesname))
                                        save_img = cv2.resize(mask, (image_x, image_y))
                                        cv2.imwrite(img_name, save_img)
                                break

                        if cv2.waitKey(1) == 27:
                                break

                self.cam.release()
                cv2.destroyAllWindows()

                if os.path.exists("./SampleGestures/"+str(gesname)+".png"):
                        QtWidgets.QMessageBox.about(self, "Success", "Gesture Saved
Successfully!")

        def exportFile(self):
                """export file module with tts assistance and gesturre viewer"""
                try:
                        clearfunc2(self.cam)
                except:
                        pass
                uic.loadUi('UI_Files/export.ui', self)
                self.setWindowTitle(self.title)
                self.create.clicked.connect(self.createGest)
                self.exp2.clicked.connect(self.exportFile)
                self.scan_sen.clicked.connect(self.scanSent)
                self.scan_sinlge.clicked.connect(self.scanSingle)
                self.create.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
                self.scan_sen.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
                self.scan_sinlge.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
                self.exp2.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
                self.exit_button.clicked.connect(self.quitApplication)
                content=checkFile()
                self.textBrowser_98.setText("         "+content)
                engine.say(str(content).lower())
                try:
                        engine.runAndWait()
                except:
                        pass
                if(content=="File Not Found"):
                        self.pushButton_2.setEnabled(False)
                        self.pushButton_3.setEnabled(False)
                else:
                        self.pushButton_2.clicked.connect(self.on_click)
                        try:
                                self.pushButton_3.clicked.connect(self.gestureViewer)
                        except:
                                pass

        def on_click(self):
                """Opens tkinter window to save file at desired location """
                content=checkFile()
                root=Tk()
                root.withdraw()
                root.filename =  filedialog.asksaveasfilename(initialdir = "/",title =
"Select file",filetypes = (("Text files","*.txt"),("all files","*.*")))
                name=root.filename
                #fr.close()
                fw=open(name+".txt","w")
                if(content=='No Content Available'):
                        content=" "
                fw.write(content)
                try:
                        os.remove("temp.txt")
                        shutil.rmtree("TempGest")
                except:
                        QtWidgets.QMessageBox.about(self, "Information", "Nothing to
export")
                fw.close()
                root.destroy()

                if not os.path.exists('temp.txt'):
                        if os.path.exists('.txt'):
                                os.remove('.txt')
                        else:
                                QtWidgets.QMessageBox.about(self, "Information", "File
saved successfully!")
                                self.textBrowser_98.setText("         ")

        def gestureViewer(self):
                """gesture viewer through matplotlib """
                try:
                        img=load_images_from_folder('TempGest/')
                        plt.imshow(img[index])
                except:
                        plt.text(0.5, 0.5, 'No new Gesture Available',
horizontalalignment='center',verticalalignment='center')
                axcut = plt.axes([0.9, 0.0, 0.1, 0.075])
                axcut1 = plt.axes([0.0, 0.0, 0.1, 0.075])
                bcut = Button(axcut, 'Next', color='dodgerblue',
hovercolor='lightgreen')
                bcut1 = Button(axcut1, 'Previous', color='dodgerblue',
hovercolor='lightgreen')
```

```
        #plt.connect('button_press_event', toggle_imagesfwd)
        bcut.on_clicked(toggle_imagesfwd)
        bcut1.on_clicked(toggle_imagesrev)
        plt.show()
        axcut._button = bcut         #creating a reference for that element
        axcut1._button1 = bcut1
    #buttonaxe._button = bcut


    def scanSent(self):
        """sentence formation module """
        try:
            clearfunc(self.cam)
        except:
            pass
        uic.loadUi('UI_Files/scan_sent.ui', self)
        self.setWindowTitle(self.title)
        self.create.clicked.connect(self.createGest)
        self.exp2.clicked.connect(self.exportFile)
        if(self.scan_sen.clicked.connect(self.scanSent)):
            controlTimer(self)
        self.scan_sinlge.clicked.connect(self.scanSingle)
        try:
            self.pushButton_2.clicked.connect(lambda:clearfunc(self.cam))
        except:
            pass
        self.linkButton.clicked.connect(openimg)
        self.create.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
        self.scan_sen.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
        self.scan_sinlge.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
        self.exp2.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
        try:
            self.exit_button.clicked.connect(lambda:clearfunc(self.cam))
        except:
            pass
        self.exit_button.clicked.connect(self.quitApplication)
        img_text = ''
        append_text=''
        new_text=''
        finalBuffer=[]
        counts=0
        while True:
            ret, frame =self.cam.read()
            frame = cv2.flip(frame,1)
            try:
                frame=cv2.resize(frame,(331,310))


                frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
                img = cv2.rectangle(frame, (150,50),(300,200), (0,255,0),
thickness=2, lineType=8, shift=0)
            except:
                keyboard.press_and_release('esc')
                keyboard.press_and_release('esc')

            height, width, channel = img.shape
            step = channel * width
        # create QImage from image
            qImg = QImage(img.data, width, height, step,
QImage.Format_RGB888)
        # show image in img_label
            try:
                self.label_3.setPixmap(QPixmap.fromImage(qImg))
                slider=self.trackbar.value()
            except:
                pass

            lower_blue = np.array([0, 0, 0])
            upper_blue = np.array([179, 255, slider])
            imcrop = img[52:198, 152:298]
            hsv = cv2.cvtColor(imcrop, cv2.COLOR_BGR2HSV)
            mask1 = cv2.inRange(hsv, lower_blue, upper_blue)


            cv2.namedWindow("mask", cv2.WINDOW_NORMAL )
            cv2.imshow("mask", mask1)
```

```
cv2.setWindowProperty("mask",cv2.WND_PROP_FULLSCREEN,cv2.WINDOW_FULLSCREEN)
                      cv2.resizeWindow("mask",118,108)
                      cv2.moveWindow("mask", 905,271)

                      hwnd = winGuiAuto.findTopWindow("mask")
                      win32gui.SetWindowPos(hwnd, win32con.HWND_TOP,
0,0,0,0,win32con.SWP_NOMOVE | win32con.SWP_NOSIZE | win32con.SWP_NOACTIVATE)

                      try:
                              self.textBrowser.setText("\n         "+str(img_text))
                      except:
                              pass
                      img_name = "1.png"
                      save_img = cv2.resize(mask1, (image_x, image_y))
                      cv2.imwrite(img_name, save_img)
                      img_text=predictor()
                      if cv2.waitKey(1) == ord('c'):
                                      try:
                                              counts+=1
                                              append_text+=img_text
                                              new_text+=img_text
                                              if not os.path.exists('./TempGest'):
                                                      os.mkdir('./TempGest')
                                              img_names =
"./TempGest/"+"{}{}.png".format(str(counts),str(img_text))
                                              save_imgs = cv2.resize(mask1, (image_x,
image_y))

                                              cv2.imwrite(img_names, save_imgs)
                                              self.textBrowser_4.setText(new_text)
                                      except:
                                              append_text+=''

                                      if(len(append_text)>1):
                                              finalBuffer.append(append_text)
                                              append_text=''
                                      else:
                                              finalBuffer.append(append_text)
                                              append_text=''

                      try:

self.pushButton.clicked.connect(lambda:saveBuff(self,self.cam,finalBuffer))
                      except:
                              pass
                      if cv2.waitKey(1) == 27:
                              break

                      if keyboard.is_pressed('shift+s'):
                              if(len(finalBuffer)>=1):
                                      f=open("temp.txt","w")
                                      for i in finalBuffer:
                                              f.write(i)
                                      f.close()
                              break


              self.cam.release()
              cv2.destroyAllWindows()

              if os.path.exists('temp.txt'):
                      QtWidgets.QMessageBox.about(self, "Information", "File is
temporarily saved ... you can now proceed to export")
              try:
                      self.textBrowser.setText("           ")
              except:
                      pass

      def scanSingle(self):
              """Single gesture scanner"""
              try:
                      clearfunc(self.cam)
              except:
                      pass
              uic.loadUi('UI_Files/scan_single.ui', self)
              self.setWindowTitle(self.title)
```

```
            self.create.clicked.connect(self.createGest)
            self.exp2.clicked.connect(self.exportFile)
            self.scan_sen.clicked.connect(self.scanSent)
            if (self.scan_sinlge.clicked.connect(self.scanSingle)):
                  controlTimer(self)
            self.pushButton_2.clicked.connect(lambda: clearfunc(self.cam))
            self.linkButton.clicked.connect(openimg)
            self.create.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
            self.scan_sen.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
            self.scan_sinlge.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
            self.exp2.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
            try:
                  self.exit_button.clicked.connect(lambda: clearfunc(self.cam))
            except:
                  pass
            self.exit_button.clicked.connect(self.quitApplication)
            img_text = ''
            while True:
                  ret, frame = self.cam.read()
                  frame = cv2.flip(frame, 1)
                  try:
                        frame = cv2.resize(frame, (321, 270))
                        frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
                        img1 = cv2.rectangle(frame, (150, 50), (300, 200), (0, 
255, 0), thickness=2, lineType=8, shift=0)
                  except:
                        keyboard.press_and_release('esc')

                  height1, width1, channel1 = img1.shape
                  step1 = channel1 * width1
                  # create QImage from image
                  qImg1 = QImage(img1.data, width1, height1, step1, 
QImage.Format_RGB888)
                  # show image in img_label
                  try:
                        self.label_3.setPixmap(QPixmap.fromImage(qImg1))
                        slider1 = self.trackbar.value()
                  except:
                        pass

                  lower_blue = np.array([0, 0, 0])
                  upper_blue = np.array([179, 255, slider1])

                  imcrop = img1[52:198, 152:298]
                  hsv = cv2.cvtColor(imcrop, cv2.COLOR_BGR2HSV)
                  mask = cv2.inRange(hsv, lower_blue, upper_blue)

                  cv2.namedWindow("mask", cv2.WINDOW_NORMAL)
                  cv2.imshow("mask", mask)
                  cv2.setWindowProperty("mask", cv2.WND_PROP_FULLSCREEN, 
cv2.WINDOW_FULLSCREEN)
                  cv2.resizeWindow("mask", 118, 108)
                  cv2.moveWindow("mask", 894, 271)

                  hwnd = winGuiAuto.findTopWindow("mask")
                  win32gui.SetWindowPos(hwnd, win32con.HWND_TOP, 0, 0, 0, 0,
                                                      win32con.SWP_NOMOVE | 
win32con.SWP_NOSIZE | win32con.SWP_NOACTIVATE)

                  try:
                        self.textBrowser.setText("\n\n\t" + str(img_text))
                  except:
                        pass

                  img_name = "1.png"
                  save_img = cv2.resize(mask, (image_x, image_y))
                  cv2.imwrite(img_name, save_img)
                  img_text = predictor()

                  if cv2.waitKey(1) == 27:
                        break

            self.cam.release()
            cv2.destroyAllWindows()

app = QtWidgets.QApplication([])
```

```
win = Dashboard()
win.show()
sys.exit(app.exec())
```

4.2 SCREENSHOTS

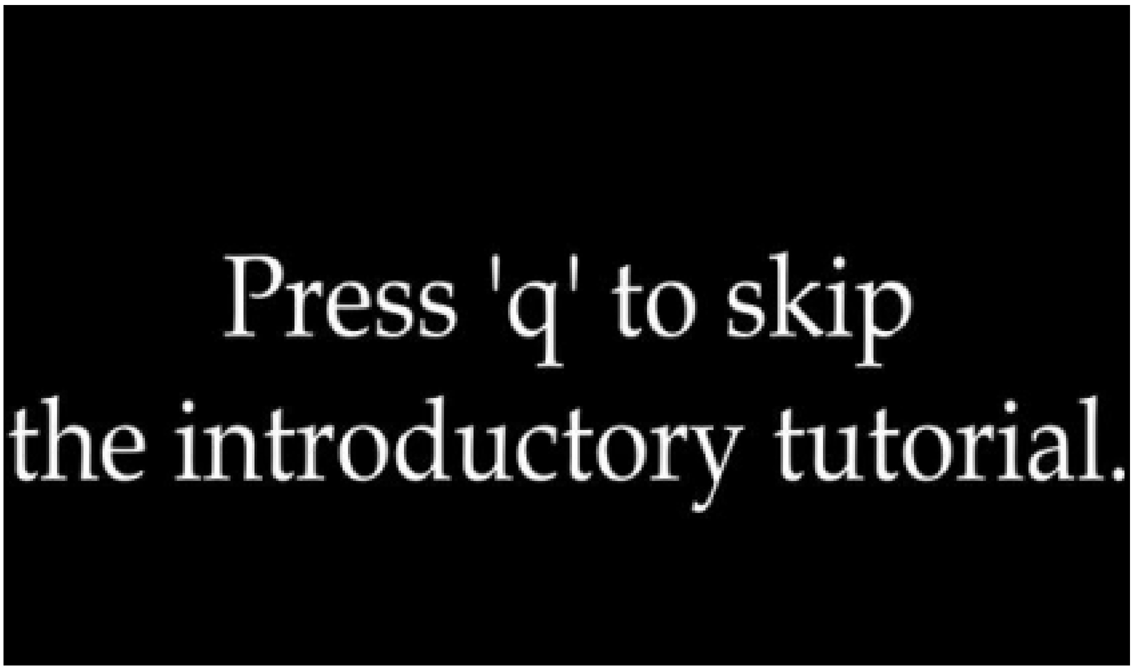
Press 'q' to skip
the introductory tutorial.

Fig 4.2.1: Skip Video.

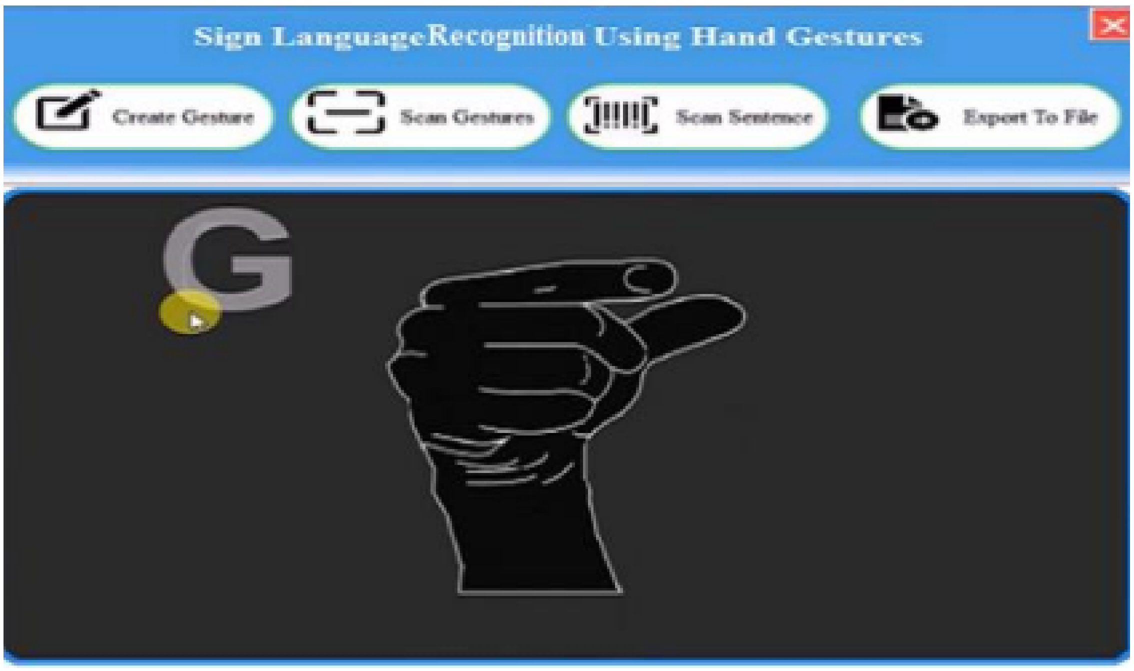

Fig 4.2.2: Dashboard with Sample Gesture Animation.

CHAPTER 5
CONCLUSION

5.1 CONCLUSION

We created a project/application to solve the basic problems faced by disabled people in communication. My research identified the following reasons why these individuals have difficulty expressing themselves independently. The main problem I see is that the viewer on the receiving end often has a hard time interpreting their message well.

Therefore, My app aims to help people learn and communicate using language. It allows users to learn hand gestures and their corresponding meanings according to American Sign Language (ASL) standards.
Users can easily associate each dot with the corresponding letter. Also, the app provides special gesture functions and helps to build sentences. Even illiterate people can use the application by understanding the movement of gestures, allowing them to create gestures corresponding to the desired characters, which will appear on the screen as a test.

We are using the TensorFlow framework with Keras API for the application. I created a complete user interface using PyQT5 to ensure the relationship between customers.

The app provides alerts and notifications based on user actions, showing consistent content and related icons. Additionally, export files are included that allow users to export their sentences and also support text-to-speech (TTS). This means that users can listen and send their sentences, as well as monitor the gestures they make during the sentence.

5.2 FUTURE SCOPE

- It can be integrated with various search engines and texting applications such as Google, and WhatsApp. So that even the illiterate people could be able to chat with other people, or query something from the web just with the help of a gesture.
- This project is working on images currently, further development can lead to detecting the motion of the video sequence and assigning it to a meaningful sentence with TTS assistance.

REFERENCES

[1] Shobhit Agarwal, "What are some problems faced by deaf and dumb people while using todays common tech like phones and PCs", 2017 [Online]. Available: https://www.quora.com/What-are-some-problems-faced-by-deaf-and-dumb-people-while using-todays-common-tech-like-phones-and-PCs, [Accessed April 06, 2019].

[2] NIDCD, "american sign language", 2017 [Online]. Available: https://www.nidcd.nih.gov/health/american-sign-language, [Accessed April 06, 2019].

[3] Suharjito MT, "Sign Language TRANSLATION Application Systems for Deaf-Mute People A Review Based on Input-Process-Output", 2017 [Online]. Available: https://www.academia.edu/35314119/Sign_Language_TRANSLATION_Application_S yste ms_for_Deaf-Mute_People_A_Review_Based_on_Input-Process-Output [Accessed April 06, 2019].