CHAPTER 1

PREAMBLE

This chapter covers the details related to the problem, solution, objectives, advantages, literature survey and organization of the report.

1.1 INTRODUCTION

Deaf and mute people around the world communicate using sign language as distinct from spoken language in their everyday a visual language that uses a system of manual, facial and body movements as the means of communication. Sign language is not a universal language, different sign languages are used in different countries, like the many spoken languages all over the world. Some countries such as Belgium, the UK, the USA or India may have more than one sign language. Hundreds of sign languages are in used around the world, for instance, Indian Sign Language, British Sign Language (BSL), Spanish Sign Language, Japanese Sign Language. Sign language is a visual language and consists of 3 major components [1].

Table 1.1: Three major components of visual language

Fingerspelling	**Word level sign vocabulary**	**Non manual features**
Used to spell words letter by letter.	Used for the majority of communication.	Facial expressions and tongue, mouth and body position.

Table 1.1 illustrates the 3 major components of visual language. Fingerspelling is used to spell letters one by one. Word level language sign vocabulary used for major part of communications. Non manual feature includes facial expressions and tongue, mouth and body positions.

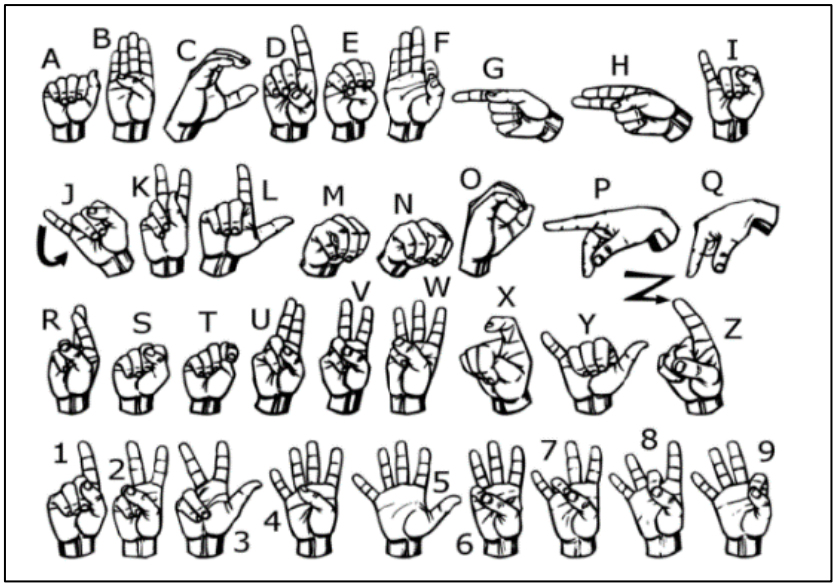

Fig 1.1: Finger Spelling American Sign Language

Fig 1.1 illustrates the finger spellings used in American Sign Language. Accordingly different regions have different finger spellings of their own.

Some of the major problems faced by a person who are unable to speak is they cannot express their emotion as freely in this world. Utilize that voice recognition and voice search systems in smartphone(s).[1] Audio results cannot be retrieved. They are not able to utilize (Artificial Intelligence/personal Butler) like google assistance, or Apple's SIRI etc.[2] because all those apps are based on voice controlling.

There is a need for such platforms for such kind of people. American Sign Language (ASL) is a complete, complex language that employs signs made by moving the hands combined with facial expressions and postures of the body. It is the go-to language of many North Americans who are not able to talk and is one of various communication alternatives used by people who are deaf or hard-of-hearing.[3]

While sign language is very essential for deaf-mute people, to communicate both with normal people and with themselves, is still getting less attention from the normal people.[4] The importance of sign language has been tending to ignored, unless there are areas of concern with individuals who are deaf-mute. One of the solutions to talk with the deaf-mute people is by using the mechanisms of sign language.

Hand gesture is one of the methods used in sign language for non-verbal communication. It is most commonly used by deaf & dumb people who have hearing or talking disorders to communicate among themselves or with normal people.[3] Various sign language systems have been developed by many manufacturers around the world but they are neither flexible nor cost-effective for the end users.

1.2 PROBLEM STATEMENT

Speech impaired people use hand signs and gestures to communicate. Normal people face difficulty in understanding their language. Hence there is a need of a system which recognizes the different signs, gestures and conveys the information to the normal people. It bridges the gap between physically challenged people and normal people.

1.3 THE SOLUTION

The process of converting the signs and gestures shown by the user into text is called sign language recognition. It bridges the communication gap between people who cannot speak and the general public. Image processing algorithms along with neural networks is used to map the gesture to appropriate text in the training data and hence raw images/videos are converted into respective text that can be read and understood.

1.4 OBJECTIVES OF PROJECT

Taking the above aspects in the consideration and providing the knowledge we have in this field; we have been motivated in this thesis to address the following issue accurately. The objectives are,

- To read an analyse the hand gestures.
- Extract different features from hand gesture.
- To segment the extracted hand image using thresholding.
- To train the machine using convolution neural network.
- To provide the accurate output label.

1.5 ADVANTAGES

The advantages are as follows

- The output of the sign language will be displayed in the text form in real time. This makes the system more efficient and hence communication of the hearing and speech impaired people is easier.
- The images captured through web cam are compared and the result of comparison is displayed at the same time.
- It enriches and enhances person’s cognitive processes, leading to higher abstract and creative thinking, better problem-solving skills, greater cognitive flexibility, better listening skills, greater academic achievement, and much more.

1.6 LITERATURE SURVEY

In 2017 Suharjito et al [2] conducted a survey of hand gesture recognition methods in sign language recognition. Sign Language Recognition (SLR) system, which is required to recognize sign languages, has been widely studied for years. The studies are based on various input sensors, gesture segmentation, extraction of features and classification methods. This aims to analyse and compare the methods employed in the SLR systems, classification methods that have been used, and suggests the most promising method for future research. Due to recent advancement in classification methods, many of the recent proposed works mainly contribute on the classification methods, such as hybrid method and Deep Learning. It focuses on the classification methods used in prior Sign Language Recognition system. Based on our review, HMM- based approaches have been explored extensively in prior research, including its modifications.

This study is based on various input sensors, gesture segmentation, extraction of features and classification methods. This aims to analyse and compare the methods employed in the SLR systems, classifications methods that have been used, and suggests the most reliable method for future research. Hybrid CNN-HMM and fully Deep Learning approaches have shown promising results and offer opportunities for further exploration.

In two-way smart communication between deaf-dumb people and normal people by Areesha Gul et al, 2021 [3], the chat applications have become a powerful media that assist people to communicate in different languages with each other. There are lots of chat applications that are used different people in different languages but there is not such a chat application that has facilitate to communicate with sign languages. The developed system is based on Sinhala Sign language. The system has included four main components as text messages are converted to sign messages, voice messages are converted to sign messages. Sign messages are converted to text messages and sign messages are converted to voice messages. Google voice recognition API has used to develop speech character recognition for voice messages. The system has been trained for the speech and text patterns by using these text parameters and signs of Sinhala Sign language is displayed by emoji. Those emoji and signs that are included in this system will bring the normal people closer to the disabled people. This is a 2-way communication system but it uses pattern of gesture recognition which is not very reliable in getting appropriate output [2].

Omkar Vedak et al in 2019 [4] has proposed some methods in a system for recognition of Indian sign language for deaf people using Otsu’s algorithm, through which

the recognition of the signs becomes easy for people while communication. And the result of those symbols signs will be converted into the text. In this project, it is capturing hand gestures through webcam and convert this image into grey scale image. The segmentation of grey scale image of a hand gesture is performed using Otsu thresholding algorithm. Total image level is divided into two classes one is hand and other is background. The optimal threshold value is determined by computing the ratio between class variance and total class variance. To find the boundary of hand gesture in image Canny edge detection technique is used. Canny edge detection uses edge-based segmentation and threshold-based segmentation. Then Otsu's algorithm is used because of its simple calculation and stability. This algorithm fails, when the global distribution of the target and background vary widely.

Speech impairment is a disability which affects one's ability to speak and hear. Such individuals use sign language to communicate with other people. Although it is an effective form of communication, there remains a challenge for people who do not understand sign language to communicate with speech impaired people, Ms. Manisha D. Raut 2015 [5] proposed Sign Language Interpreter using Image Processing and Machine Learning. The aim of this is to develop an application which will translate sign language to English in the form of text and audio, thus aiding communication with sign language. The application acquires image data using the webcam of the computer, then it is pre-processed using a combinational algorithm and recognition is done using template matching. The translation in the form of text is then converted to audio. The database used for this system includes 6000 images of English alphabets. It uses 4800 images for training and 1200 images for testing. The system produces 88% accuracy.

Prerna Sharma and Naman Sharma in 2019 [6] has proposed the communication plays a crucial part in human life as gesture recognition system. It encourages a man to pass on his sentiments, feelings and messages by talking, composing or by utilizing some other medium. Gesture based communication is the main method for Communication for the discourse and hearing weakened individuals. Communication via gestures is a dialect that utilizations outwardly transmitted motions that consolidates hand signs and development of the hands, arms, lip designs, body developments and outward appearances, rather than utilizing discourse or content, to express the individual's musings. Gestures are the expressive and important body developments that speaks to some message or data. Gestures are the requirement for hearing and discourse hindered, they pass on their message to others just with the assistance of motions. Gesture Recognition System is the capacity of the computer interface to catch, track and perceive the motions and deliver the yield in light of

the caught signals. It enables the clients to interface with machines without the any need of mechanical gadgets. There are two sorts of sign recognition methods: image- based and sensor- based strategies. Image based approach is utilized as a part of this project that manages communication via gestures motions to distinguish and track the signs and change over them into the relating discourse and content.

1.7 ORGANIZATION OF REPORT

Chapter 1 deals with the problem statement, solution, project objective, advantages, literature survey and organization of the project report.

Chapter 2 deals with the software requirements specification under that functional overview, functional requirements, software requirements, hardware requirements and project cycle.

Chapter 3 deals with the system design under that project architecture, sequence diagram, dataflow diagram and flowchart.

Chapter 4 deals with the implementation it has codes used in the project.

Chapter 5 deals with the testing part of the project under that it has scope of testing, unit testing, integration testing, functional testing and system testing.

Chapter 6 deals with the results it has snapshots of the project.

Chapter 7 deals with conclusion and future scope.

CHAPTER 2

SOFTWARE REQUIREMENTS SPECIFICATION

In this chapter we discuss about the Software requirements specification User has to provide input and dataset collected from dataset the we review and extract the data.

2.1 FUNCTIONAL OVERVIEW

The visual based assistance for differently abled by recognizing sign language system hand pictures of a person through the camera and compares them with the several signs existing in the dataset, in order to detect the sign.

- Firstly, takes picture as camera input.
- Secondly, hand detection and creation of Region of Interest (ROI).
- Thirdly, detect hand in ROI and feed them into the classification device.
- Lastly, for the sign, classification is determined.

2.2 USER CHARCTERSTICS

User has to provide the input. Then we extract the data that is feature extraction is done by using the matrix form and applying the CNN model. Later system provides the appropriate output.

2.3 FUNCTIONAL REQUIREMENTS

Our system's functional requirements are as follows

- Recognise the Hand gestures.
- Segmentation of the documentation.
- Pre-processing documentation to enhance the quality.
- Extraction of features using training models such as CNN algorithm.
- Display the translated text and voice to the user.

2.4 NON-FUNCTIONAL REQUIREMENTS

Our system's functional requirements are as follows

- Sign language recognition system should produce accurate results.
- It can be adapted using any python platforms.
- This system should be flexible.
- The software system can be moved from one machine to another without incurring considerable overhead.

2.5 SOFTWARE REQUIREMNTS

- Operating System : Microsoft Windows 7
- Programming Language : Python 3.6
- Software : Anaconda

PROGRAMMING LANGUAGE USED

PYTHON

Python is an interpreted, high-level and a general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace. It is the most used programming language presently. It provides constructs that enable clear programming on both small and large scales. The logistic regression in the system is implemented in jupyter and the algorithm is written in python language.

Python functions saves you having to re-type everything every time you run. The standard Unix implementation of Python provides an integrated development environment called idle, which bundles a Python interpreter window with a Pythonaware text editor. To start up idle, log in to the server from an xterm and type IDLE. Then will get a Python shell window, which is an ordinary Python interpreter except that it allows some limited editing capabilities. The real power of idle comes from the use of the integrated editor. To get an editor window for a new file, just choose New Window from the File menu on the Python Shell window. If work is related with an existing file instead, just choose Open from the File menu, and pick the file you want from the resulting dialog box. Then can type text into the editor window, and cut and paste in a fashion that will probably be familiar to most computer users.

HTML, CSS, JSCRIPT

These are the most commonly used webpage development languages. The user interface designed for the outputbased system is been developed using these languages. The website acts as an interface which takes the input those are the various constraints from the users and passes to the program to work upon.

2.6 HARDWARE REQUIREMENTS

- Camera : 3MP
- RAM : 4GB
- Processor : Intel dual core i3
- Processor Speed : 1.0GHz
- Hard disk : 20 GB

2.7 PROJECT CYCLE

Table 2.1: Project cycle

Activity	2021			2022					
	Oct	Nov	Dec	Jan	Feb	Mar	Apr	May	Jun
Collecting Requirements									
Synopsis preparation									
Web interface									
Data collection									
Data Validation									
Processing of collected data and applying algorithm									
Testing									
Report Writing									
Submission of project									

The above table 2.1 represents the project cycle of our project. Collecting requirements is done in October. Synopsis preparation was done in November. Web interface was done in December. Data collection was done in January. Data validation was done in February. Processing of collected data and applying CNN algorithm was done March and April. Testing done in May. Report writing was done June. Submission of project will be done in July.

CHAPTER 3

SYSTEM DESIGN

In this chapter the discussion is held on the project architecture and different UML diagrams corresponding to the project.

3.1 PROJECT ARCHITECTURE AND DESCRIPTION

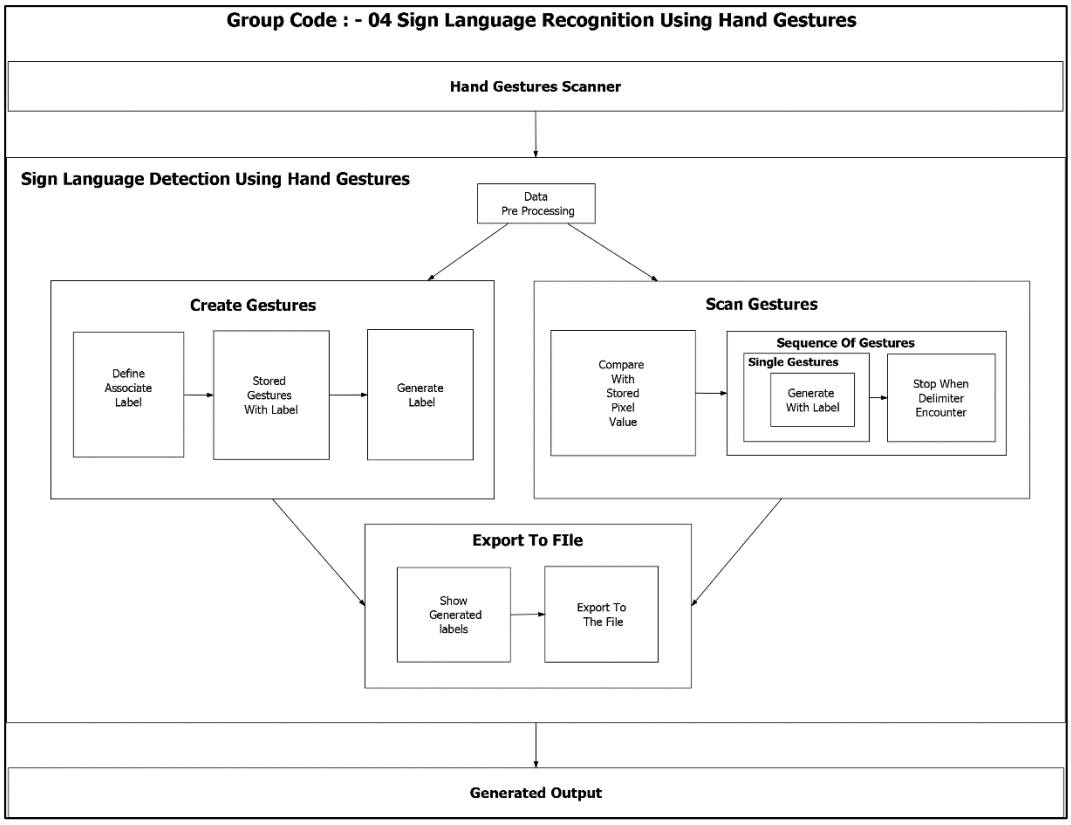

Fig 3.1: Architecture of Sign Language Recognition System

Every project has its own project architecture. The project architecture includes all components, how they interact with each other, the environment in which they operate, and the principles used to design the software. In many cases, it also includes the evolution of the software into the future.

Fig 3.1 illustrates the Architecture of Sign Language recognition System**.** Firstly, the scene is captured through the device camera. It will be in the form of video stream which is converted into single frame of video. Later segmentation is performed using threshold, and segmented image with hand extracted is obtained. The segmented hands are trained with Convolution Neural Network with ASL dataset. In the last step the gained output is labelled respectively.

3.2 METHODOLOGY

DATASET GENERATION

Steps we followed to create our data set are as follows:

1. We use Open computer vision (OpenCV) library in order to produce our dataset. Firstly we capture around 800 images of each of the symbol in ASL for training purposes and around 200 images per symbol for testing purpose.
2. First we capture each frame shown by the webcam of our machine. In the each frame we define a region of interest (ROI) which is denoted by a green bounded square as shown in the image.
3. From this whole image we extract our ROI which is RGB and convert it into grey scale image.
4. Finally we apply our gaussian blur filter to our image which helps us extracting various features of our image.

GESTURE CLASSIFICATION

ALGORITHM LAYER 1

1. Apply gaussian blur filter and threshold to the frame taken with OpenCV to get the processed image after feature extraction.
2. This processed image is passed to the CNN model for prediction and if a letter is detected for more than 50 frames then the letter is printed and taken into consideration for forming the word.
3. Space between the words are considered using the blank symbol.

ALGORITHM LAYER 2

1. We detect various sets of symbols which show similar results on getting detected.
2. We then classify between those sets using classifiers made for those sets only.

LAYER 1

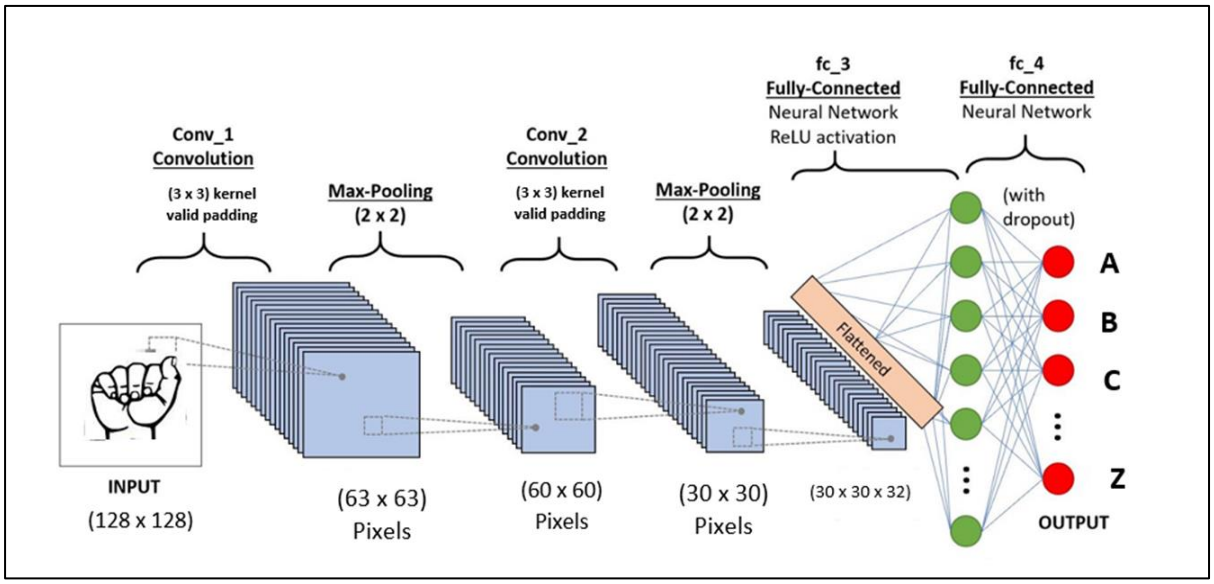

Fig 3.2: CNN Model

The fig 3.2 illustrates the CNN Model and its structure adopted in this project.

1) **1st Convolution Layer :** The input picture has resolution of 128x128 pixels. It is first processed in the first convolutional layer using 32 filter weights (3x3 pixels each). This will result in a 126X126 pixel image, one for each Filter-weights.
2) **1st Pooling Layer :** The pictures are downsampled using max pooling of 2x2 i.e. we keep the highest value in the 2x2 square of array. Therefore, our picture is downsampled to 63x63 pixels.
3) **2nd Convolution Layer :** Now, these 63 x 63 from the output of the first pooling layer is served as an input to the second convolutional layer. It is processed in the second convolutional layer using 32 filter weights (3x3 pixels each).This will result in a 60 x 60 pixel image.
4) **2nd Pooling Layer :** The resulting images are downsampled again using max pool of 2x2 and is reduced to 30 x 30 resolution of images.
5) **1st Fully Connected Layer :** Now these images are used as an input to a fully connected layer with 128 neurons and the output from the second convolutional layer is reshaped to an array of 30x30x32 =28800 values. The input to this layer is an array of 28800 values. The output of these layer is fed to the 2nd Fully Connected Layer. We are using a dropout layer of value 0.5 to avoid overfitting.
6) **2nd Fully Connected Layer :** Now the output from the 1st Fully Connected Layer are used as an input to a fully connected layer with 96 neurons.

7) **Final layer:** The output of the 2nd Fully Connected Layer serves as an input for the final layer which will have the number of neurons as the number of classes we are classifying (alphabets + blank symbol).

LAYER 2

We are using two layers of algorithms to verify and predict symbols which are more similar to each other so that we can get us close as we can get to detect the symbol shown. In our testing we found that following symbols were not showing properly and were giving other symbols also :

1. For D : R and U
2. For I : T, D, K and I
3. For S : M and N

So to handle above cases we made three different classifiers for classifying these sets:

1. {D,R,U}
2. {T,K,D,I}
3. {S,M,N}

3.3 ALGORITHM

SIFT algorithm is used for prediction of the signs.

Step 1: Loads image.

Step 2: Converts Python Imaging Library Image instance to a Numpy array.

Step 3: Keypoints and computing descriptors are extracted using the Scale Invariant Feature Transform.

Step 4: Sift object is created, and comparison of original image and image to compare is done.

Step 5: FlannBasedMatcher is loaded to find the matches.

Step 6: The array named matches will contain all possible matches, so many false matches as well.

Step 7: The ratio test is applied to select only the good matches. It selects only the matches with lower distance, so higher quality.

3.3 SEQUENCE DIAGRAMS

The sequence diagram it refers to the interactions which are arranged in the given time sequence. The objects involve in scenario and which exchange the sequence.

Messages between the objects. The user collects the data of hotel reviews from the dataset and the data is pre-processed using particular algorithm. In the pre-processing step

the data is pre-processed and data splitting operation is performed and trained. The data is trained using CNN model then it sends the answer to view the results.

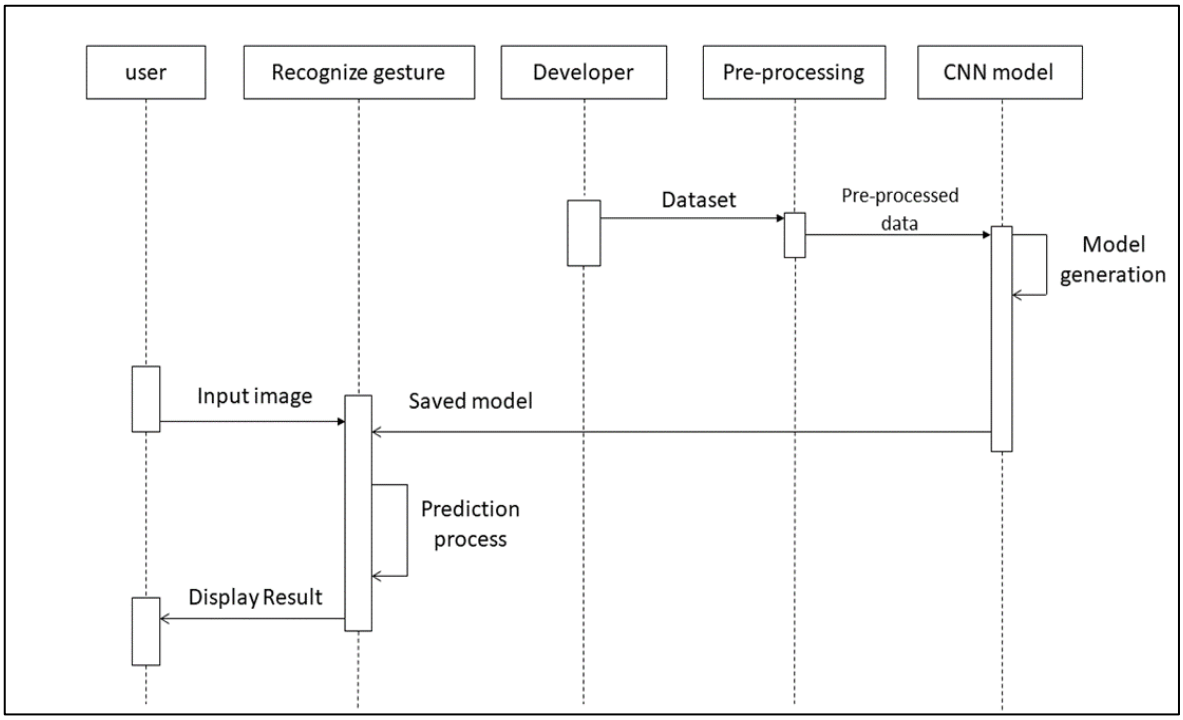

Fig 3.3: Sequence Diagram

Fig 3.3 illustrates the sequence diagram of Sign recognition system. And it also shows in which our project works.

3.4 DATA FLOW DIAGRAMS

1. A data flow diagram (DFD) is graphic representation of the "flow" of data through an information system. A data flow diagram can also be used for the visualization of data processing (structured design). It is common practice for a designer to draw a context level DFD first which shows the interaction between the system and outside entities. DFD's shows the flow of data from external entities into the system, how the data moves from one process to another, as well as its logical storage. There are only four symbols:
2. Squares representing external entities, which are sources and destinations of information entering and leaving the system.
3. Rounded rectangles representing processes, in other methodologies, may be called 'Activities', 'Actions', 'Procedures', 'Subsystems' etc. which take data as input, do processing to it, and output it.

4. Arrows representing the data flows, which can either, be electronic data or physical items. It is impossible for data to flow from data store to data store except via a process, and external entities are not allowed to access data stores directly.
5. The flat three-sided rectangle is representing data stores should both receive information for storing and provide it for further processing.

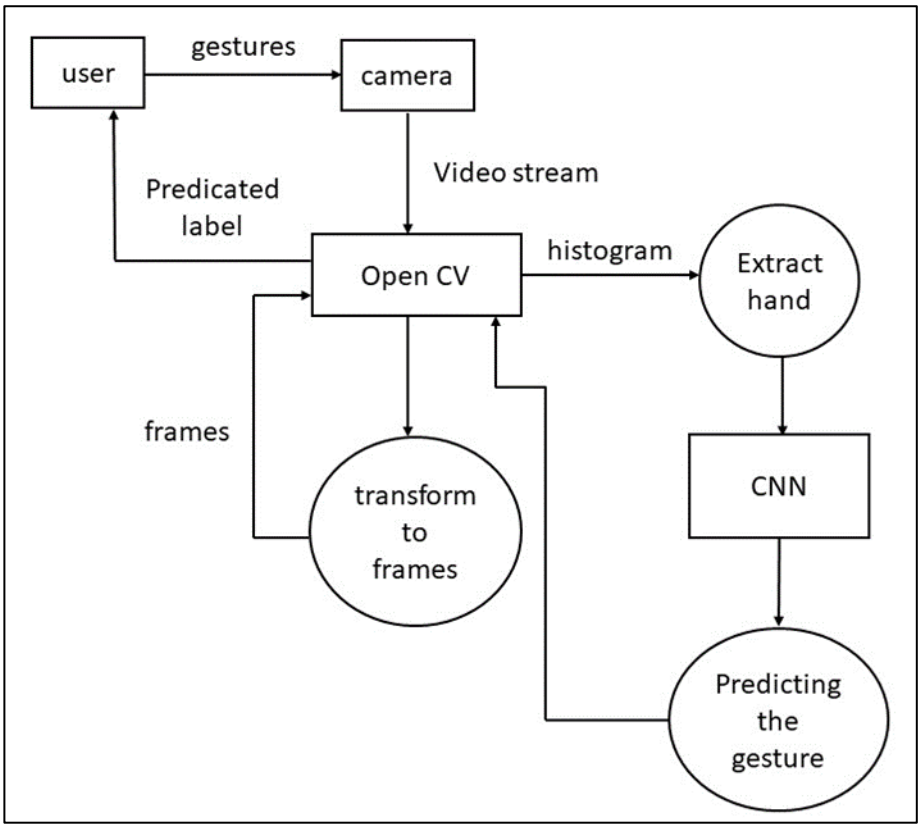

Fig 3.4: Data Flow Diagram

Fig 3.4 shows the flow of data in the sign language recognition.

3.5 FLOW CHART

The flow chart shows the flow of data in our project and its classification.

Step 1: Start

Step 2: Load dataset, the dataset is loaded to train and test the model. The data set contains the different alphabets and its classification.

Step 3: Pre-process data, in which we are going to process the data according to the need and also, we will remove all the unwanted features.

Step 4: Feature Extraction, in this step we are going to extract the feature term and also the Aspect terms will be extracted to analyse.

Step 5: Construct CNN model, in which we are going to analyse the signs by using the CNN model.

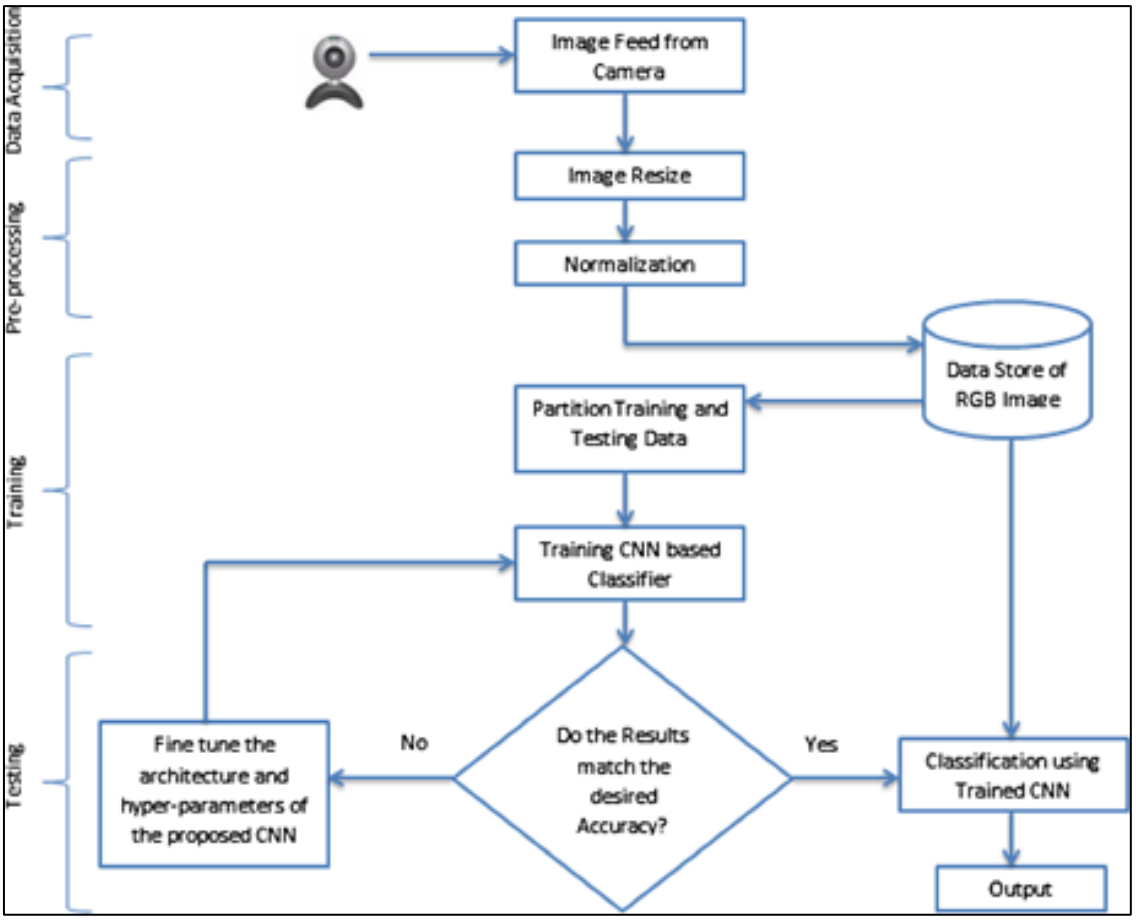

Fig 3.5: Flowchart

Fig 3.5 gives the complete picture of flow of the project. A flowchart is a picture of the separate steps of a process in sequential order. It is a generic tool that can be adapted for a wide variety of purposes, and can be used to describe various processes, such as a manufacturing process, an administrative or service process, or a project plan.

3.6 UTILITIES

The utilities used in this project are as follows

- **cv2** - OpenCV is a Python open-source library, which is used for computer vision in Artificial intelligence, Machine Learning, face recognition, etc.
- **numpy** - NumPy is a Python library used for working with arrays. It also has functions for working in domain of linear algebra, fourier transform, and matrices.
- **os** - The OS module in Python provides functions for creating and removing a directory (folder), fetching its contents, changing and identifying the current directory, etc.

- **PyQt5** - PyQt5 is cross-platform GUI toolkit, a set of python bindings for Qt v5. One can develop an interactive desktop application with so much ease because of the tools and simplicity provided by this library. A GUI application consists of Front-end and Back-end.
- **matplotlib** - Matplotlib is a cross-platform, data visualization and graphical plotting library for Python and its numerical extension NumPy.
- **win32api** - The win32api module provides various libraries and objects utilized to deal with the Windows system's Application Programming Interface (API). The PyWin32 library, which is already a part of the Python extension, enables the win32api module in Python.
- **shutil** - shutil to replicate the complete directory

CHAPTER 4

IMPLEMENTATION

This chapter deals with the implementation of the project.

4.1 SOFTWARE TOOLS USED

The software tools which are very important for developing our projects and also helps in implementation of our projects.

ANACONDA

Anaconda is a distribution of the Python and R programming languages for scientific computing (data science, machine learning applications, large-scale data processing, predictive analytics, etc.), that aims to simplify package management and deployment. The distribution includes data-science packages suitable for Windows, Linux, and macOS. It is developed and maintained by Anaconda, Inc., which was founded by Peter Wang and Travis Oliphant in 2012. As an Anaconda, Inc. product, it is also known as Anaconda Distribution or Anaconda Individual Edition, while other products from the company are Anaconda Team Edition and Anaconda Enterprise Edition, both of which are not free.

Package versions in Anaconda are managed by the package management system conda. This package manager was spun out as a separate open-source package as it ended up being useful on its own and for things other than Python. There is also a small, bootstrap version of Anaconda called Miniconda, which includes only conda, Python, the packages they depend on, and a small number of other packages.

Anaconda distribution comes with over 250 packages automatically installed, and over 7,500 additional open-source packages can be installed from PyPI as well as the conda package and virtual environment manager. It also includes a GUI, Anaconda Navigator, as a graphical alternative to the command-line interface (CLI).

The big difference between conda and the pip package manager is in how package dependencies are managed, which is a significant challenge for Python data science and the reason conda exists. Before version 20.3, when pip installed a package, it automatically installed any dependent Python packages without checking if these conflict with previously installed packages. It would install a package and any of its dependencies regardless of the state of the existing installation. Because of this, a user with a working installation of, for example, TensorFlow, could find that it stopped working having used pip to install a different package that requires a different version of the dependent numpy library than the

one used by TensorFlow. In some cases, the package would appear to work but produce different results in detail. While pip has since implemented consistent dependency resolution, this difference accounts for a historical differentiation of the conda package manager.

In contrast, conda analyses the current environment including everything currently installed, and, together with any version limitations specified (e.g., the user may wish to have TensorFlow version 2,0 or higher), works out how to install a compatible set of dependencies, and shows a warning if this cannot be done.

Open-source packages can be individually installed from the Anaconda repository, Anaconda Cloud (anaconda.org), or the user's own private repository or mirror, using the conda install command. Anaconda, Inc. compiles and builds the packages available in the Anaconda repository itself, and provides binaries for Windows 32/64 bit, Linux 64 bit and MacOS 64-bit.

Anything available on PyPI may be installed into a conda environment using pip, and conda will keep track of what it has installed itself and what pip has installed. Custom packages can be made using the conda build command, and can be shared with others by uploading them to Anaconda Cloud, PyPI or other repositories. The default installation of Anaconda2 includes Python 2.7 and Anaconda3 includes Python 3.7. However, it is possible to create new environments that include any version of Python packaged with conda.

4.2 IMPLEMENTATION DETAILS

The purpose of this project is to construct a model to output the answer for the sign shown.

Dataset: The Dataset is collected from Kaggle datasets which consists of 250 sign for each alphabet.

This project performs following steps.

1. Building CNN model
2. Preprocessing
3. Training
4. Saving the model

1. Building CNN Model

Classifier.compiler(optimizer = optimizers.SGD(lr=0.01), loss = ‘categorial_crosstropy’, metrics = [‘accuracy’])

2. Pre-processing Data

from keras.preprocessing.image import ImageDataGenerator train_datagen = ImageDataGenerator(rescale=1./255,shear_range=0.2,zoom_range=0.2, horizontal_flip = True)test_datagen = ImageDataGenerator(rescale=1./255)

3. Training

training_set = train_datagen.flow_from_directory('Dataset/training_set', target_size = (64, 64), batch_size=32, class_mode='categorical’)

model = classifier.fit_generator(training_set, steps_per_epoch=800, epochs=25, validation_data = test_set, validation_steps = 6500)

4. Saving the model

import h5py

classifier.save('Trained_model.h5')

CONVERTING COLORED IMAGE TO GAUSSIAN BLUR IMAGE

- Frame is taken from the image and value is returned from getting the camera frame, either true of false.
- Resizing of image is done.
- BGR is converted to RGB.
- Rectangle is created using cv2. rectangle().
- Image is converted to qimage.
- Qimage is image representation that allows direct access to the pixel data.
- QPixmap can be used to show an image in the window.
- Trackbars are created by specifying value property.
- Removal of blue background.
- Converting RGB image to hsv image.
- Threshold the HSV image.
- Denote the lower and upper boundary of the threshold region.

Code for 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 libraries
#from pilutil import *
#from scipy.ndimage import imread           #will help in reading the images
from matplotlib.pyplot import imread
#from scipy.misc import toimage
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                               #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 dirrectory 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 = cv2.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 = cv2.imread('template.png')
        cv2.imshow("Image",image)
        cv2.setWindowProperty("Image",cv2.WND_PROP_FULLSCREEN,cv2.WINDO
W_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 customgesture saved prediction is made by checking array or through SiFt algo"""
        import numpy as np
        from keras.preprocessing import image
```

```
        test_image = image.load_img('1.png', target_size=(64, 64))
        test_image = 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 = cv2.imread("./SampleGestures/"+fileEntry[i])
                original = cv2.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):                    #goodpoints
and matcches sum from 1.png and customgestureimages is grater than 20
                        gesname=fileEntry[i]
                        gesname=gesname.replace('.png','')
                        if(gesname=='sp'):
        #sp is replaced with <space>
                                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")
                        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.resizeWindow("mask",118,108)
    cv2.moveWindow("mask", 905,271)
    hwnd = winGuiAuto.findTopWindow("mask")
    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
                            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()
        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.resizeWindow("mask",118,108)
                    cv2.moveWindow("mask", 894,271)
                    hwnd = winGuiAuto.findTopWindow("mask")
                    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
app = QtWidgets.QApplication([])
win = Dashboard()
win.show()
sys.exit(app.exec())
```

4.3 FRONT END FORMS DESIGN

The frontend form designs are

1. **HTML:** abbreviated as Hyper Text Markup Language is the most general, basic and important front-end language. HTML is very flexible and also easily understandable syntax. HTML is the building block of the web page construction and also by using HTML we can insert the images, text and also the videos into the web page and make the web page more attractive and effective.

2. **CSS:** It is abbreviated as Cascading Style Sheet and it is used for the styling the web page and also the styling the HTML pages as we required. By using CSS the web pages can be more attractive and also very colourful. The CSS is very important in today's world because the people will give much importance for the presentation rather than the content so it should be given more importance to know the syntax of the CSS code and also implement it effectively.
3. **JS:** It is abbreviated as JavaScript; it is also a high-level programming language and it is used to make the web page interactive and useful. In our project we have designed the login and the register pages by using the java script. It is also an object-oriented programming language. It is the most important language for making the website interactive.
4. **Bootstrap:** It is used for making the website more attractive and also responsive. The bootstrap provides many functions which are used to provide in our program. The bootstrap provides the functions to provide both mobile view and the desktop view so that the format of the web Page will not be damaged.

Code of Dashboard.ui file

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>MainWindow</class>
 <widget class="QMainWindow" name="MainWindow">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>741</width>
    <height>642</height>
   </rect>
  </property>
  <property name="windowTitle">
   <string>MainWindow</string>
  </property>
  <widget class="QWidget" name="centralwidget">
```

```
  <widget class="QGraphicsView" name="graphicsView_2">
   <property name="geometry">
    <rect>
     <x>0</x>
     <y>0</y>
     <width>741</width>
     <height>161</height>
    </rect>
   </property>
   <property name="toolTip">

<string>&lt;html&gt;&lt;head/&gt;&lt;body&gt;&lt;p&gt;&lt;br/&gt;&lt;/p&gt;&lt;/bod
y&gt;&lt;/html&gt;</string>
   </property>
   <property name="styleSheet">
    <string notr="true">background-color: qlineargradient(spread:pad, x1:0.543136,
y1:0.074, x2:0.52, y2:1, stop:0.880682 rgba(75, 161, 223, 255), stop:1 rgba(255, 255,
255, 255));
</string>
   </property>
  </widget>
  <widget class="QLabel" name="label">
   <property name="geometry">
    <rect>
     <x>130</x>
     <y>10</y>
     <width>531</width>
     <height>31</height>
    </rect>
   </property>
   <property name="font">
    <font>
```

```
      <family>Times New Roman</family>
      <pointsize>18</pointsize>
      <weight>75</weight>
      <bold>true</bold>
     </font>
    </property>
    <property name="styleSheet">
     <string notr="true">color:white;</string>
    </property>
    <property name="text">
     <string>Sign Language Recognition Using Hand Gestures</string>
    </property>
   </widget>
   <widget class="QLabel" name="label_3">
    <property name="geometry">
     <rect>
      <x>0</x>
      <y>160</y>
      <width>741</width>
      <height>431</height>
     </rect>
    </property>
    <property name="styleSheet">
     <string notr="true">
font-size: 40px;
color:white;
</string>
    </property>
    <property name="text">
     <string/>
```

```
  </property>
 </widget>
 <widget class="QGraphicsView" name="graphicsView">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>160</y>
    <width>741</width>
    <height>441</height>
   </rect>
  </property>
  <property name="styleSheet">
   <string notr="true">background-color:#fefefe;
</string>
  </property>
 </widget>
 <widget class="QPushButton" name="create">
  <property name="geometry">
   <rect>
    <x>10</x>
    <y>70</y>
    <width>171</width>
    <height>61</height>
   </rect>
  </property>
  <property name="font">
   <font>
    <family>TImes New Roman</family>
    <pointsize>12</pointsize>
    <weight>50</weight>
```

```
      <italic>false</italic>
      <bold>false</bold>
     </font>
    </property>
    <property name="layoutDirection">
     <enum>Qt::RightToLeft</enum>
    </property>
    <property name="styleSheet">
     <string notr="true">QPushButton#create{
padding-left:-40px;
color:black;
background-color:#fefefe;
border: 2px solid #58e870; border-radius:30px;
font: 12pt &amp;quot;TImes New Roman&amp;quot;;
background-image:url(&amp;quot;icons/create.png&amp;quot;);
background-repeat:none;
}
QPushButton#create:hover
{
  background-color:lightgreen;
          color:white;
}</string>
    </property>
    <property name="text">
     <string>   Create Gesture</string>
    </property>
   </widget>
   <widget class="QPushButton" name="scan_sen">
    <property name="geometry">
     <rect>
```

```
      <x>370</x>
      <y>70</y>
      <width>171</width>
      <height>61</height>
     </rect>
    </property>
    <property name="font">
     <font>
      <family>TImes New Roman</family>
      <pointsize>12</pointsize>
      <weight>50</weight>
      <italic>false</italic>
      <bold>false</bold>
     </font>
    </property>
    <property name="styleSheet">
     <string notr="true">QPushButton#scan_sen{
padding-left:-40px;
color:black;
background-color:#fefefe;
border: 2px solid #58e870; border-radius:30px;
font: 12pt &amp;quot;TImes New Roman&amp;quot;;
background-image:url(&amp;quot;icons/sent.png&amp;quot;);
background-repeat:none;
}
QPushButton#scan_sen:hover
{
  background-color:lightgreen;
        color:white;
}</string>
```

```
 </property>
 <property name="text">
  <string>                    Scan Sentence </string>
 </property>
</widget>
<widget class="QPushButton" name="scan_sinlge">
 <property name="geometry">
  <rect>
   <x>190</x>
   <y>70</y>
   <width>171</width>
   <height>61</height>
  </rect>
 </property>
 <property name="font">
  <font>
   <family>TImes New Roman</family>
   <pointsize>12</pointsize>
   <weight>50</weight>
   <italic>false</italic>
   <bold>false</bold>
  </font>
 </property>
 <property name="layoutDirection">
  <enum>Qt::LeftToRight</enum>
 </property>
 <property name="autoFillBackground">
  <bool>false</bool>
 </property>
 <property name="styleSheet">
```

```
    <string notr="true">QPushButton#scan_sinlge{
padding-left:-40px;
color:black;
background-color:#fefefe;
border: 2px solid #58e870; border-radius:30px;
font: 12pt &amp;quot;TImes New Roman&amp;quot;;
background-image:url(&amp;quot;icons/scan.png&amp;quot;);
background-repeat:none;
}
QPushButton#scan_sinlge:hover
{
  background-color:lightgreen;
          color:white;
}</string>
   </property>
   <property name="text">
    <string>                    Scan Gestures</string>
   </property>
  </widget>
  <widget class="QPushButton" name="exp2">
   <property name="geometry">
    <rect>
     <x>560</x>
     <y>70</y>
     <width>171</width>
     <height>61</height>
    </rect>
   </property>
   <property name="font">
    <font>
```

```
      <family>TImes New Roman</family>
      <pointsize>12</pointsize>
      <weight>50</weight>
      <italic>false</italic>
      <bold>false</bold>
     </font>
    </property>
    <property name="styleSheet">
     <string notr="true">QPushButton#exp2{
padding-left:-40px;
color:black;
background-color:#fefefe;
border: 2px solid #58e870; border-radius:30px;
font: 12pt &amp;quot;TImes New Roman&amp;quot;;
background-image:url(&amp;quot;icons/export.png&amp;quot;);
background-repeat:none;
}
QPushButton#exp2:hover
{
  background-color:lightgreen;
          color:white;
}</string>
    </property>
    <property name="text">
     <string>                    Export To File</string>
    </property>
    <property name="checkable">
     <bool>false</bool>
    </property>
    <property name="autoRepeat">
```

```
    <bool>false</bool>
   </property>
   <property name="autoExclusive">
    <bool>false</bool>
   </property>
   <property name="autoDefault">
    <bool>false</bool>
   </property>
   <property name="default">
    <bool>false</bool>
   </property>
   <property name="flat">
    <bool>false</bool>
   </property>
  </widget>
  <widget class="QPushButton" name="exit_button">
   <property name="geometry">
    <rect>
     <x>710</x>
     <y>2</y>
     <width>31</width>
     <height>31</height>
    </rect>
   </property>
   <property name="styleSheet">
    <string notr="true">QPushButton#exit_button{
background-image:url(&quot;icons/quit.png&quot;);
background-repeat:none;
}</string>
   </property>
```

```
   <property name="text">
    <string/>
   </property>
  </widget>
 </widget>
 <widget class="QMenuBar" name="menubar">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>741</width>
    <height>21</height>
   </rect>
  </property>
 </widget>
 <widget class="QStatusBar" name="statusbar"/>
</widget>
<resources/>
<connections/>
</ui>
```

CHAPTER 5

TESTING

Software testing is an investigation conducted to provide stake holders with information about the quality of software product service under test.

5.1 SCOPE

The scope of testing are as follows

- The scope of this project is to extract most important aspect of a sign language recognition.
- It is also performed for people with disabilities like hearing impairment and mutes.

5.2 UNIT TESTING

All decision branches and internal code flow should be validated. It is the testing of individual software units of the application. It is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

Table 5.1: Unit Testing

Sl No	Test Case	Expected Output	Actual Output	Remarks
1	Create Sign	Captures the sign successfully	Captures the sign successfully	Pass
2	Scan Sign	Detects and scans the sign	Detects 200 out of 250 signs	Pass
3	Scan Sentence	Detects and scans the sentence	Detects 200 out of 250 signs	Pass
4	Export to File	Saves the sign and exports to system files	Saves the sign and exports to system files	Pass

The above table 5.1 represents unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs.

5.3 INTEGRATION TESTING

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent.

Table 5.2: Integration Testing

Test Case	Expected Output	Actual Output	Remarks
Hand capture	Active ROI	Active ROI	Pass
Sign detection	100% detection	Detects up to 80%	Pass

Table 5.2 illustrates the integration testing. It specifically aimed at exposing the problems that arise from the combination of components.

5.4 SYSTEM TESTING

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration-oriented system integration test.

Table 5.3: System Testing

Sl No.	Testing	Expected Output	Actual Output	Remarks
1	Using the available resources which help to obtain expected results	Detects the sign and provide an accurate output	Detects the sign and provide an accurate output	Pass
2	OS Compatibility	Performance is good in windows 7	Performance is good in windows 7	Pass

Table 5.3 illustrates the system testing. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

CHAPTER 6

RESULTS

In this chapter we discuss about the snapshots of the project. Here it contains layout information and results that are retrieved at a specific point in time.

6.1 SNAP SHOTS OF THE PROJECT

Fig 6.1: Dashboard

Fig 6.1 is the dashboard of our project. It contains the options such as scan gestures, scan sentences and export file.

Fig 6.2: Scan Gesture

Fig 6.2 shows the scan gesture feature. After scanning the gesture, we have to click on stop scan and it gets saved.

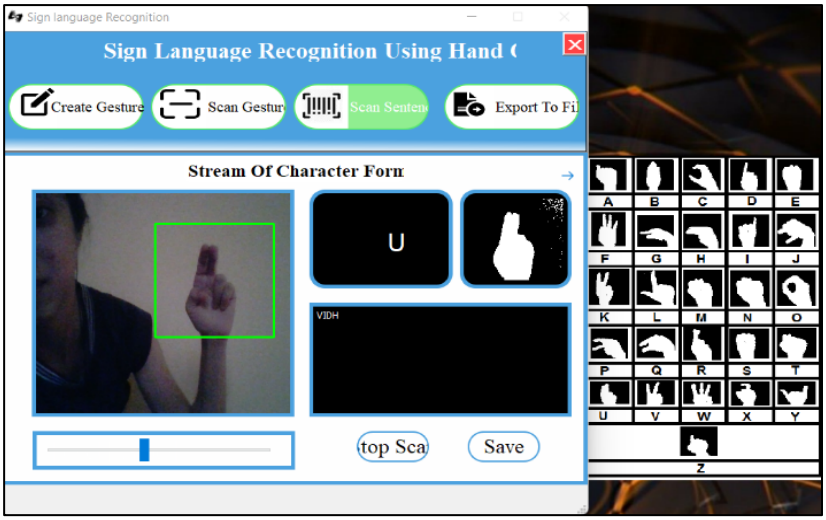

Fig 6.3: Scan Sentence

Fig 6.3 shows the scan sentence feature. After scanning the sentence, we have to click on stop scan and click save hence it gets saved.

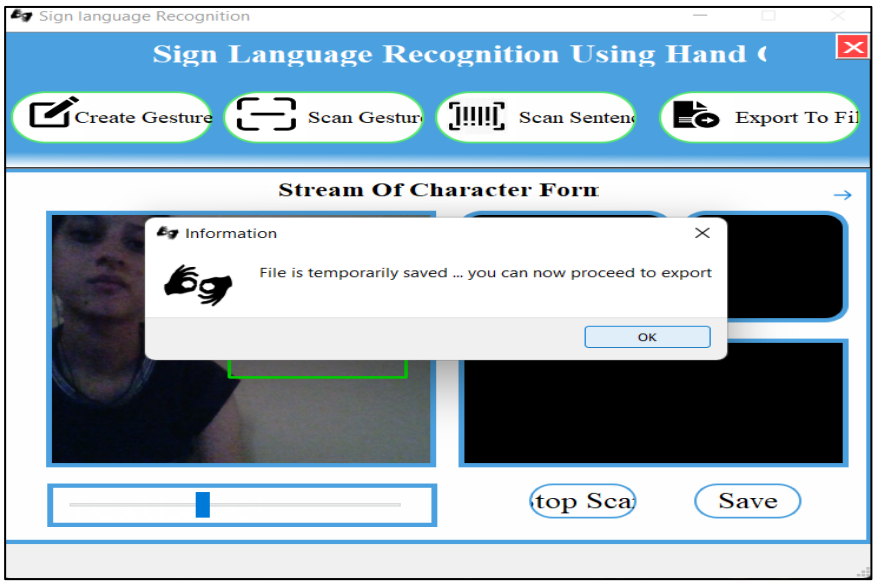

Fig 6.4: Save

Fig 6.4 illustrates the window that pop when the sign is saved to the disk drive.

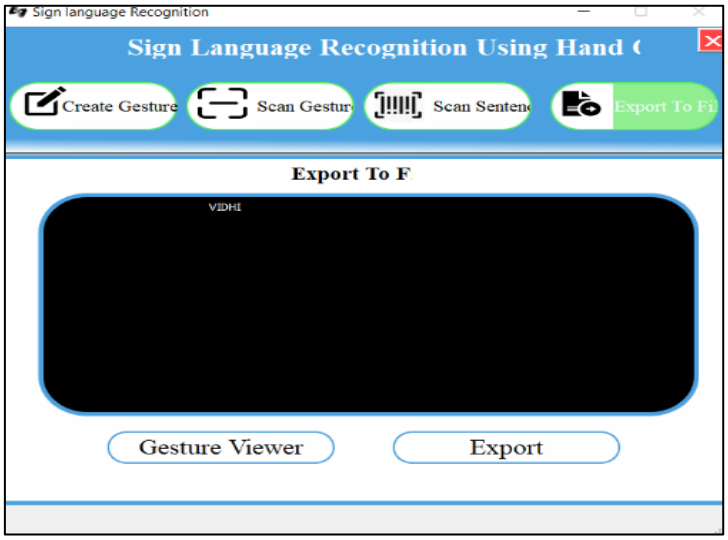

Fig 6.5: Export to File

Fig 6.5 shows the export option and its other facilities.

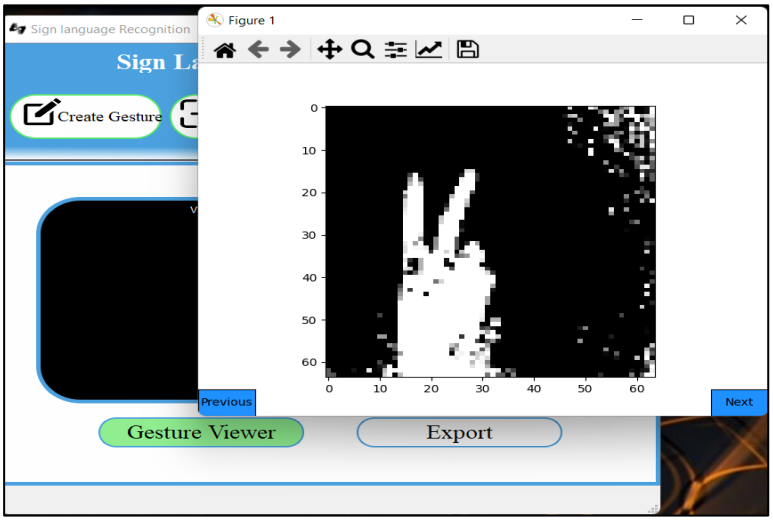

Fig 6.6: Gesture View

Fig 6.6 shows the gesture view of the sign provided as an input.

CHAPTER 7

CONCLUSION

From this project we have tried to overshadow some of the major problems faced by the disabled persons in terms of talking. We found out the root cause of why they can't express more freely. The result that we got was the other side of the audience are not able to interpret what these persons are trying to say or what is the message that they want to convey.

Thereby this application serves the person who wants to learn and talk in sign languages. With this application a person will quickly adapt various gestures and their meaning as per Indian Sign Language standards. They can quickly learn what alphabet is assigned to which gesture. Add-on to this custom gesture facility is also provided along with sentence formation. A user need not be a literate person if they know the action of the gesture, they can quickly form the gesture and appropriate assigned character will be shown onto the screen.

Appropriate user-friendly messages are prompted as per the user actions along with what gesture means. Additionally, an export to file module is also provided with TTS(Text-To-Speech) assistance. Whatever may be the sentence was formed a user will be able to listen to it and then quickly export along with observing what gesture he/she made during the sentence formation.

REFERENCES

[1] Neel Kamal Bhagat, Vishnusai Y, Rathna G N, “Indian Sign Language Gesture Recognition using Image Processing and Deep Learning”, IEEE publisher, ID No. 978-1-7281-3857-2/19/, 2019.

[2] Suharjito, Meita Chandra Ariesta, Fanny Wiryana and Gede Putra Kusuma, “A Survey of Hand Gesture Recognition Methods in Sign Language Recognition”, IEEE publisher, ID No. 2231-8526, 2017.

[3] Areesha Gul, Batool Zehra, Sadia Shah, Nazish Javed, Muhammad Imran Saleem, “Two-way Smart Communication System for Deaf & Dumb and Normal People”, IEEE publisher, ID No. 978-1-7281-6899-9, 2020.

[4] Ms.Manisha D.Raut, ,Ms. Pallavi Dhok, Mr.Ketan Machhale, Ms. Jaspreet Manjeet Hora, “A System for Recognition of Indian Sign Language for Deaf People using Otsu’s Algorithm”, IEEE publisher, ID No 2395 -0056, 2015.

[5] Omkar Vedak, Prasad Zavre, Abhijeet Todkar, Manoj Patil, “Sign Language Interpreter using Image Processing and Machine Learning”, IEEE publisher, ID No. 2395-0056, 2019.

[6] Prerna Sharma, Naman Sharma, “Gesture Recognition System”, IEEE publisher, ID No. 978-1-7281-1253-4/19/, 2019.

[7] Muneer Al-Hammadi, Ghulam Muhammad, Wadood Abdul, “Deep Learning-Based Approach for Sign Language Gesture Recognition with Efficient Hand Gesture Representation”, IEEE publisher, 2020.

[8] A. S. Ghotkar, “Dynamic Hand Gesture Recognition and Novel Sentence Interpretation Algorithm for Indian Sign Language Using Microsoft Kinect Sensor,” vol. 1, pp. 24–38, 2015.

[9] K. Dabre, “Machine Learning Model for Sign Language Interpretation using Webcam Images,” pp. 317–321, 2014.

[10] O. De, P. Deb, S. Mukherjee, S. Nandy, T. Chakraborty, and S. Saha, “Computer Vision Based Framework for Digit Recognition by Hand Gesture Analysis,” 2016.

[11] Q. Wu, Y. Liu, Q. Li, S. Jin and F. Li, “The application of deep learning in computer vision,” 2017 Chinese Automation Congress (CAC), Jinan, 2017, pp. 6522-6527.