A Project Report

SIGN LANGUAGE INTERPRETER

Submitted in partial fulfillment of the requirements for the award of degree

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

by

RATHOD MAHESHWARI (160118733009)

DARA SUPRIYA (160118733021)

Department of Computer Science and Engineering,

Chaitanya Bharathi Institute of Technology (Autonomous),

(Affiliated to Osmania University, Hyderabad)

Hyderabad, TELANGANA (INDIA) – 500 075

November-2020

CERTIFICATE

This is to certify that the project titled "**Sign Langugage Interpreter**" is the bonafide work carried out by **Rathod Maheshwari(160118733009) , Dara Supriya(160118733021)** students of B.E.(CSE) of Chaitanya Bharathi Institute of Technology, Hyderabad, affiliated to Osmania University, Hyderabad, Telangana(India) during the academic year 2020-2021, submitted in partial fulfillment of the requirements for the award of the degree in **Bachelor of Engineering** (**Computer Science and Engineering**) and that the project has not formed the basis for the award previously of any other degree, diploma, fellowship or any other similar title.

Mentor(s)
Smt. Naga jyothi
Asst.Professor
Department of CSE

Smt. Ch Madhavi Sudha
Asst.Professor
Dapartment of CSE

Smt. T Suvarna Kumari
Asst.Professor
Department of CSE

Head, CSE Dept.
Dr Y Ramadevi
Professor and head
Dept of CSE

Place: Hyderabad

Date: 28-11-2020

DECLARATION

I/we hereby declare that the project entitled "**Sign Language Interpreter**" submitted for the B. E (CSE) degree is my original work and the project has not formed the basis for the award of any other degree, diploma, fellowship or any other similar titles.

Rathod Maheshwari(160118733009)

Dara Supriya(160118733021)

Name(s) and Signature(s) of the Student

Place: HYDERABAD

Date: 28-11-2020

Table of Contents

ABSTRACT

Hand gesture is one of the method used in sign language for non-verbal communication. It is most commonly used by deaf & dumb people who have hearing or speech problems to communicate among themselves or with normal people. Various sign language systems has been developed by many makers around the world but they are neither flexible nor cost-effective for the end users. Hence in this paper introduced software which presents a system prototype that is able to automatically recognize sign language to help deaf and dumb people to communicate more effectively with each other or normal people. Hand Gesture recognition system provides us an innovative, natural, user friendly way of communication with the computer which is more familiar to the human beings. In our method, the hand is first passed through a filter and after the filter is applied the hand is passed through a classifier which predicts the class of the hand gestures. Our method provides 95.7 % accuracy for the 26 letters of the alphabet.

ACKNOWLEDGEMENT

It is our privilege to acknowledge with deep sense of gratitude and devotion for keen personal interest and invaluable guidance rendered by our Project mentors **Smt. Naga jyothi , Smt.Ch Madhavi Sudha**.

We hereby take this opportunity to add a special note of thanks to **Smt.Suvarna Kumari** for her invaluable suggestions that helped us in successful completion of the project.

Our respects and regards to **Dr. Y Rama Devi**, Professor, Department of Computer Science and Technology, Chaitanya Bharathi Institute of Technology for cooperation and encouragement. .

We are grateful to our Principal **Dr. P. Saradhi Varma**, Chaitanya Bharathi Institute of Technology, for having designed an excellent learning atmosphere.

Finally, we also thank all the staff members, faculty of Dept. of CSE, CBIT, our friends, and all our family members who with their valuable suggestions and support, directly or indirectly helped us in completing this project work

1.INTRODUCTION

1.1Problem Definition including the significance and objective

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)Audio results cannot be retrieved. They are not able to utilize (Artificial Intelligence/personal Butler) like google assistance, or Apple's SIRI etc. 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.

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. 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 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

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 display the result of the character detected on the screen, also allowing such users to build their customized gesture so that the problems faced by persons who aren't able to talk vocally can be accommodated with technological assistance and the barrier of expressing can be oversheld.

1.2 Methodologies

The system is a vision based approach. All the signs are represented with bare hands and so it eliminates the problem of using any artificial devices for interaction.

- **Data Set Generation**

For the project we tried to find already made datasets but we couldn't find dataset in the form of raw images that matched our requirements. All we could find were the datasets in the form of RGB values. Hence we decided to create our own data set. Steps we followed to create our data set are as follows.

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

- **Gesture Classification**

  The approach which we used for this project is to use two layers of algorithm to predict the final symbol of the user.

  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 .

  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:

  CNN Model :

  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 Densely 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 Densely Connected Layer.We are using a dropout layer of value 0.5 to avoid overfitting.

6. 2nd Densely Connected Layer : Now the output from the 1st Densely Connected Layer are used as an input to a fully connected layer with 96 neurons.

7. Final layer: The output of the 2nd Densely 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).

Activation Function : We have used ReLu (Rectified Linear Unit) in each of the layers(convolutional as well as fully connected neurons). ReLu calculates max(x,0) for each input pixel. This adds nonlinearity to the formula and helps to learn more complicated features.It helps in removing the vanishing gradient problem and speeding up the training by reducing the computation time.

Pooling Layer : We apply Max pooling to the input image with a pool size of (2, 2) with relu activation function.This reduces the amount of parameters thus lessening the computation cost and reduces overfitting.

Dropout Layers: The problem of overfitting, where after training, the weights of the network are so tuned to the training examples they are given that the network doesn't perform well when given new examples.This layer "drops out" a random set of activations in that layer by setting them to zero.The network should be able to provide the right classification or output for a specific example even if some of the activations are dropped out

Optimizer : We have used Adam optimizer for updating the model in response to the output of the loss function. Adam combines the advantages of two extensions of two stochastic gradient descent algorithms namely adaptive gradient algorithm(ADA GRAD) and root mean square propagation(RMSProp)

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 U : D and R
3. For I : T, D, K and I

4. 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}

- **Training and Testing :**

  We convert our input images(RGB) into grayscale and apply gaussian blur to remove unnecessary noise.We apply adaptive threshold to extract our hand from the background and resize our images to 128 x 128.

  We feed the input images after preprocessing to our model for training and testing after applying all the operations mentioned above.

  The prediction layer estimates how likely the image will fall under one of the classes. So the output is normalized between 0 and 1 and such that the sum of each values in each class sums to 1. We have achieved this using softmax function.

  At first the output of the prediction layer will be somewhat far from the actual value. To make it better we have trained the networks using labeled data. The cross-entropy is a performance measurement used in the classification. It is a continuous function which is positive at values which is not same as labeled value and is zero exactly when it is equal to the labeled value. Therefore we optimized the cross-entropy by minimizing it as close to zero. To do this in our network layer we adjust the weights of our neural networks. TensorFlow has an inbuilt function to calculate the cross entropy.

  As we have found out the cross entropy function, we have optimized it using Gradient Descent in fact with the best gradient descent optimizer is called Adam Optimizer.

1.3 Outline of the results

The role of the interpreter appears to be very straightforward—to effectively facilitate communication between deaf individuals and those who are hearing. However, the complexities of the task, the types of visual interpreting, and the enormous range of qualifications brought by the interpreter make it anything but simple. Interpreting requires a high level of fluency in two or more languages, keen ability to focus on what is being said, broad-based world knowledge, and professional, ethical conduct. Interpreters cannot interpret what they do not understand. Interpreters serve all parties in the communication exchange. While we often think of the

deaf person as the requester of interpreter services, the reality is, all parties have an equal and mutual need for the interpreter.

1.4 Scope of the project

One of the solutions to communicate with the deaf-mute people is by using the serve of sign language interpreter. But the usage of sign language interpreter expensive. Cost-effective solution is required so that the deaf-mute and normal people can communicate normally and easily.

Our strategy involves implementing such an application which detects pre-defined American sign language (ASL) through hand gestures. For the detection of movement of gesture, we would use basic level of hardware component like camera and interfacing is required. Our application would be a comprehensive User-friendly. Instead of using technology like gloves or kinect, we are trying to solve this problem using state of the art computer vision and machine learning algorithms.

2.LITERATURE SURVEY

2.1 Introduction to the problem domain terminology

The model built should be reasonably robust to variations in camera angle, video quality, distance from the camera to the subject, variations in lighting etc.

Such a model has several use-cases, and can be deployed either through an app or website for people to use. It has high relevance in the lives of not only the deaf-dumb, but also anyone who interacts with them.

There are 1.1 million deaf-and-dumb people in India. 98% of these people are illiterate, making sign language their only method of communication. This trend is not expected to change anytime soon, as only 2% of deaf children attend school.

One approach to sign language classification involves splitting a video into its constituent signs, and then running a pretrained model to classify each sign. However, segmentation of signs is a non-trivial problem, and has not been successfully solved yet. This approach can also be taken one step further, by splitting each sign into its constituent sub-units, running a pre-trained model to classify those sub-units, and using these classes to predict the meaning of the sentence as a whole. However, this also faces the same challenges as the sign-segmentation, as there are no clearly demarcated boundaries when transitioning from one sign to the next. Several signs are also combinations of other signs (for example, the sign for “truck” is the sign for “car” + the sign for “long”), and these will have to be taken care of as well.

An alternate way to go about this (while building early versions of a translator) would be to introduce a minor inconvenience for the user, by asking him/her to shift to a rest pose after each sign, while signing. Adding this inconvenience would make segmenting a trivial job, and then each sign can be easily classified

2.2 Existing solutions

Globally, numerous attempts have been made to do Sign Language classification on videos. A few of the notable attempts are:

1. **Deep Hand**: This CVPR 2016 paper presents a new approach to learning a frame-based classifier on weakly labelled sequence data by embedding a CNN within an iterative EM algorithm. This allows the CNN to be trained on a vast number of example images when only loose sequence level information is available for the source videos. Its use in continuous sign language recognition is evident when it was tested on two publicly available large sign language data sets, where it outperforms the current state-of-the-art by a large margin. More details about the paper can be found
2. **Deep Sign**: Hybrid CNN-HMM for Continuous Sign Language Recognition – This paper introduces the end-to-end embedding of a CNN into a HMM, while interpreting the outputs of the CNN in a Bayesian fashion. The hybrid CNN-HMM combines the strong discriminative abilities of CNNs with the sequence modelling capabilities of HMMs. More information about the paper can be found

2.3 Proposed Systems

As more hearing and deaf people learn sign language, the range of careers open to that skill has broadened. In fact, the U.S. Bureau of Labor Statistics says a career as an interpreter is growing much faster than average, with 19% growth predicted through 2028.

In addition to traditional careers like interpreting, new career possibilities open up as technology evolves. Fluency in American Sign Language (ASL) is also needed for specialized careers in education, health, therapy, and recreational services.

Although it is a relatively new area of research, in the last two decades have been published a significant number of works focusing the development of techniques to automate the translation of sign languages with greater incidence for the American Sign Language, and the introduction of serious games in the education of people with speech and/or hearing disabilities. Several of the methods proposed to perform representation and recognition of sign language gestures, apply some of the main state-of-the-art techniques, involving segmentation, tracking and feature extraction as well as the use of specific hardware as depth sensors and data gloves.

Deusdado writes about the use of new technologies for dissemination and teaching of sign language, highlighting the use of 3D models (avatars) in the translation of words to sign language. Kulkarni et al. provide an automatic translation model of static gestures corresponding to the alphabet in American Sign Language

2.4 TOOLS/TECHNOLOGIES USED:

2.4.1 Software Requirement:

- Microsoft Windows XP or later / Ubuntu 12.0 LTS or later /MAC OS 10.1 or later.
- Python 3.6.6 or updated version
- Tensorflow 1.11.0
- OpenCV 3.4.3.18
- NumPy 1.15.3
- Matplotlib 3.0.0
- Keras 2.2.1
- PIL 5.3.0

2.4.2 Hardware Requirement:

- Intel Core i3 3rd gen processor or later.
- 512 MB disk space.
- 512 MB RAM.
- Any external or inbuild camera with minimum pixel resolution 200 x 200 (300ppi or 150lpi) 4-megapixel cameras and up.

3 DESIGN OF THE PROPOSED SYSTEM/METHOD/ ALGORITHM

3.1 Block Diagram

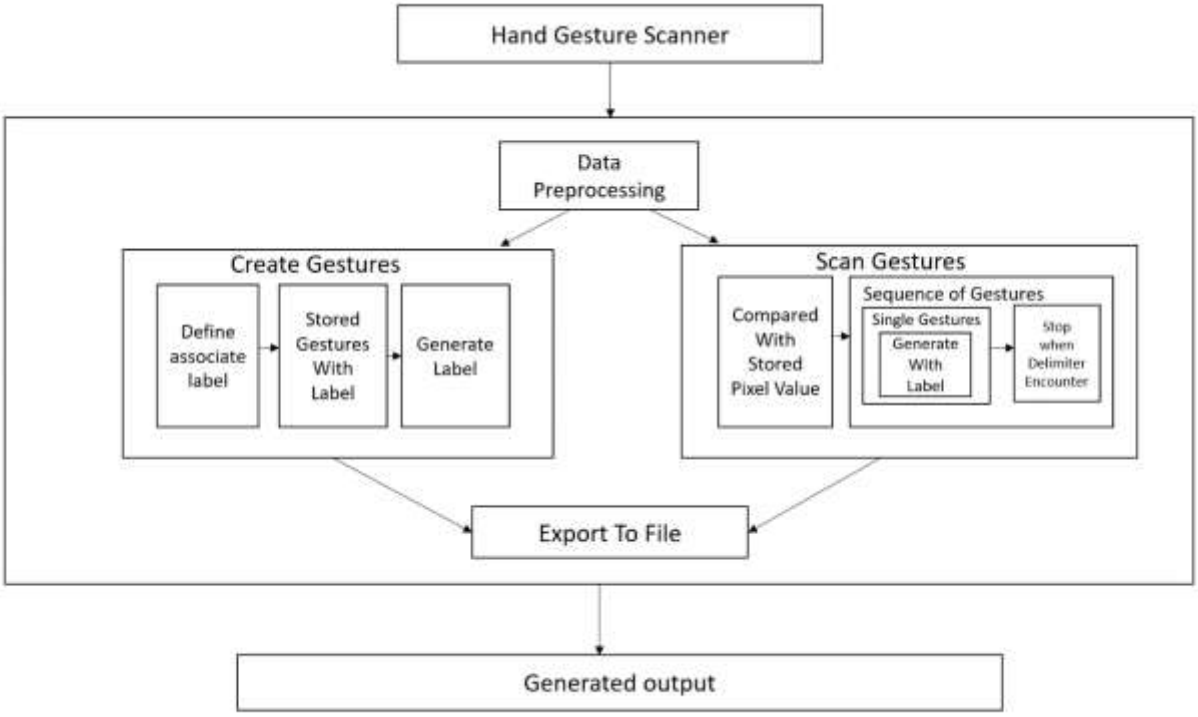
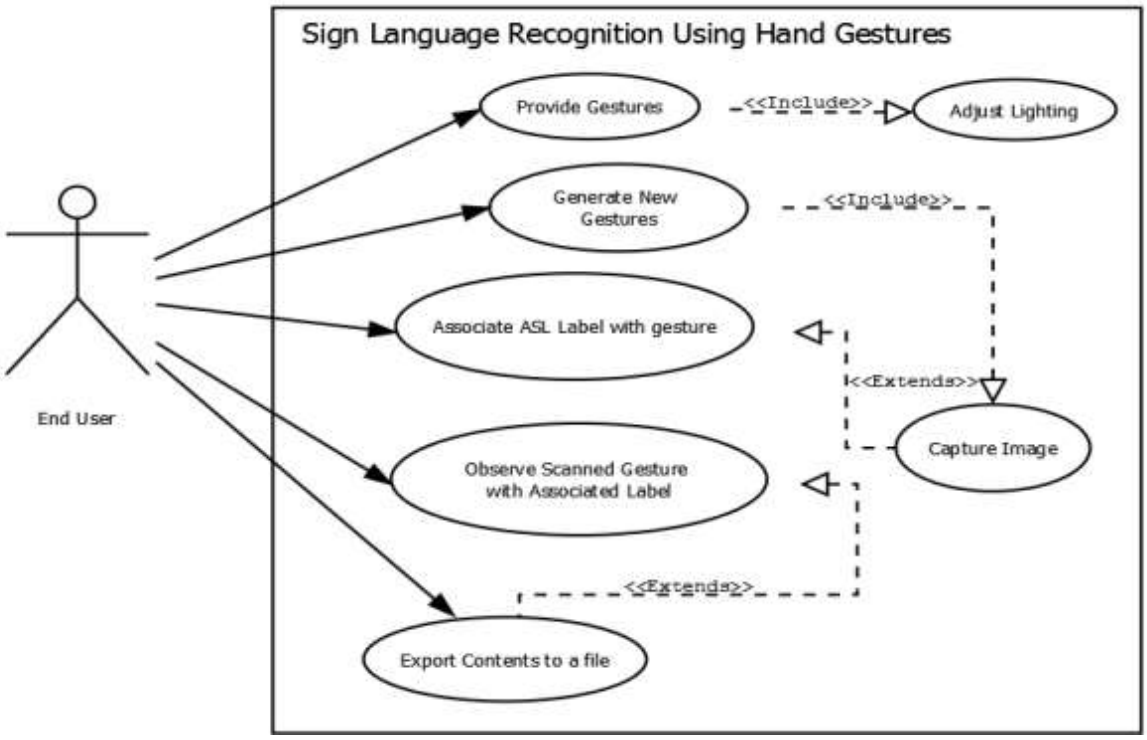


Fig 1: Block Diagram for Sign Language interpreter.

3.2 USE CASE DAIGRAM



3.3 Module description

Data Pre-Processing – In this module, based on the object detected in front of the camera its binary images is being populated. Meaning the object will be filled with solid white and background will be filled with solid black. Based on the pixel's regions, their numerical value in range of either 0 or 1 is being given to next process for modules

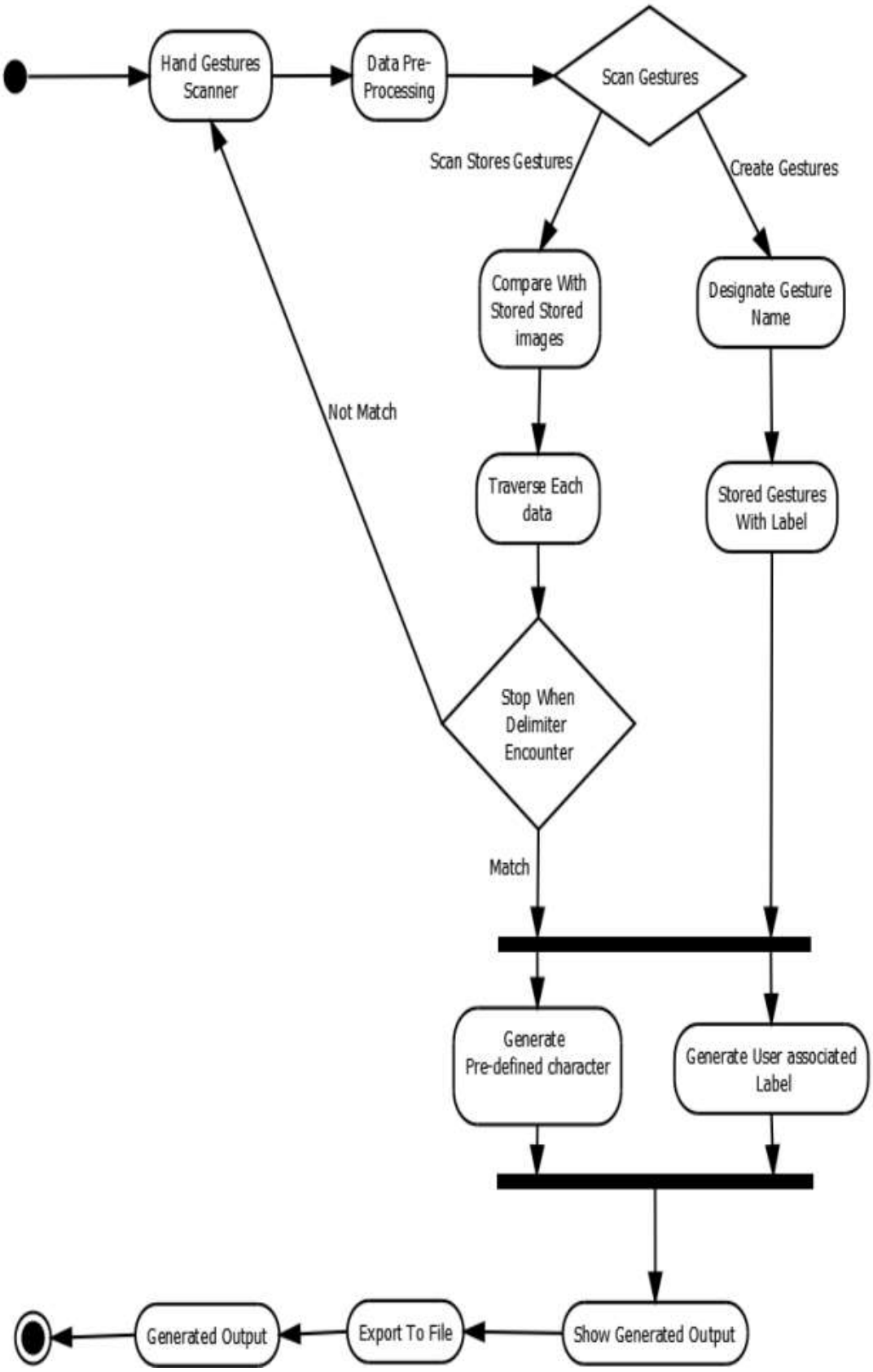
Scan Single Gesture – A gesture scanner will be available in front of the end user where the user will have to do a hand gesture. Based on Pre-Processed module output, a user shall be able to see associated label assigned for each hand gestures, based on the predefined American Sign Language (ASL) standard inside the output window screen.

Create gesture –A user will give a desired hand gesture as an input to the system with the text box available at the bottom of the screen where the user needs to type whatever he/she desires to associate that gesture with. This customize gesture will then be stored for future purposes and will be detected in the upcoming time.

Exporting – A user would be able to export the results of the scanned character into an ASCII standard textual file format.

4 IMPLEMENTATION OF THE PROPOSED SYSTEM

4.1 Flowchart


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

4.2 Algorithm /pseudo code

Collect_data.py

```
import cv2
import numpy as np
import os
import string

# Create the directory structure
if not os.path.exists("data"):
    os.makedirs("data")
if not os.path.exists("data/train"):
    os.makedirs("data/train")
if not os.path.exists("data/test"):
    os.makedirs("data/test")

for i in string.ascii_uppercase:
    if not os.path.exists("data/train/" + i):
        os.makedirs("data/train/" + i)
    if not os.path.exists("data/test/" + i):
        os.makedirs("data/test/" + i)

# Train or test
mode = 'train'
directory = 'data/' + mode + '/'

''' mode = 'test'
directory = 'data/' + mode + '/' '''

minValue = 70

cap = cv2.VideoCapture(0)
interrupt = -1

while True:
    _, frame = cap.read()
    # Simulating mirror image
    frame = cv2.flip(frame, 1)

    # Getting count of existing images
    count = {
        'a': len(os.listdir(directory + "/A")),
        'b': len(os.listdir(directory + "/B")),
        'c': len(os.listdir(directory + "/C")),
        'd': len(os.listdir(directory + "/D")),
        'e': len(os.listdir(directory + "/E")),
        'f': len(os.listdir(directory + "/F")),
        'g': len(os.listdir(directory + "/G")),
        'h': len(os.listdir(directory + "/H")),
        'i': len(os.listdir(directory + "/I")),
        'j': len(os.listdir(directory + "/J")),
        'k': len(os.listdir(directory + "/K")),
        'l': len(os.listdir(directory + "/L")),
        'm': len(os.listdir(directory + "/M")),
        'n': len(os.listdir(directory + "/N")),
        'o': len(os.listdir(directory + "/O")),
        'p': len(os.listdir(directory + "/P")),
        'q': len(os.listdir(directory + "/Q")),
        'r': len(os.listdir(directory + "/R")),
        's': len(os.listdir(directory + "/S")),
        't': len(os.listdir(directory + "/T")),
        'u': len(os.listdir(directory + "/U")),
        'v': len(os.listdir(directory + "/V")),
```

```
        'w': len(os.listdir(directory + "/W")),
        'x': len(os.listdir(directory + "/X")),
        'y': len(os.listdir(directory + "/Y")),
        'z': len(os.listdir(directory + "/Z"))
    }

    # Printing the count in each set to the screen
    cv2.putText(frame, "a : " + str(count['a']), (10, 100), cv2.FONT_HERSHEY_PLAIN, 1,
(0, 255, 255), 1)
    cv2.putText(frame, "b : " + str(count['b']), (10, 110), cv2.FONT_HERSHEY_PLAIN, 1,
(0, 255, 255), 1)
    cv2.putText(frame, "c : " + str(count['c']), (10, 120), cv2.FONT_HERSHEY_PLAIN, 1,
(0, 255, 255), 1)
    cv2.putText(frame, "d : " + str(count['d']), (10, 130), cv2.FONT_HERSHEY_PLAIN, 1,
(0, 255, 255), 1)
    cv2.putText(frame, "e : " + str(count['e']), (10, 140), cv2.FONT_HERSHEY_PLAIN, 1,
(0, 255, 255), 1)
    cv2.putText(frame, "f : " + str(count['f']), (10, 150), cv2.FONT_HERSHEY_PLAIN, 1,
(0, 255, 255), 1)
    cv2.putText(frame, "g : " + str(count['g']), (10, 160), cv2.FONT_HERSHEY_PLAIN, 1,
(0, 255, 255), 1)
    cv2.putText(frame, "h : " + str(count['h']), (10, 170), cv2.FONT_HERSHEY_PLAIN, 1,
(0, 255, 255), 1)
    cv2.putText(frame, "i : " + str(count['i']), (10, 180), cv2.FONT_HERSHEY_PLAIN, 1,
(0, 255, 255), 1)
    cv2.putText(frame, "j : " + str(count['j']), (10, 190), cv2.FONT_HERSHEY_PLAIN, 1,
(0, 255, 255), 1)
    cv2.putText(frame, "k : " + str(count['k']), (10, 200), cv2.FONT_HERSHEY_PLAIN, 1,
(0, 255, 255), 1)
    cv2.putText(frame, "l : " + str(count['l']), (10, 210), cv2.FONT_HERSHEY_PLAIN, 1,
(0, 255, 255), 1)
    cv2.putText(frame, "m : " + str(count['m']), (10, 220), cv2.FONT_HERSHEY_PLAIN, 1,
(0, 255, 255), 1)
    cv2.putText(frame, "n : " + str(count['n']), (10, 230), cv2.FONT_HERSHEY_PLAIN, 1,
(0, 255, 255), 1)
    cv2.putText(frame, "o : " + str(count['o']), (10, 240), cv2.FONT_HERSHEY_PLAIN, 1,
(0, 255, 255), 1)
    cv2.putText(frame, "p : " + str(count['p']), (10, 250), cv2.FONT_HERSHEY_PLAIN, 1,
(0, 255, 255), 1)
    cv2.putText(frame, "q : " + str(count['q']), (10, 260), cv2.FONT_HERSHEY_PLAIN, 1,
(0, 255, 255), 1)
    cv2.putText(frame, "r : " + str(count['r']), (10, 270), cv2.FONT_HERSHEY_PLAIN, 1,
(0, 255, 255), 1)
    cv2.putText(frame, "s : " + str(count['s']), (10, 280), cv2.FONT_HERSHEY_PLAIN, 1,
(0, 255, 255), 1)
    cv2.putText(frame, "t : " + str(count['t']), (10, 290), cv2.FONT_HERSHEY_PLAIN, 1,
(0, 255, 255), 1)
    cv2.putText(frame, "u : " + str(count['u']), (10, 300), cv2.FONT_HERSHEY_PLAIN, 1,
(0, 255, 255), 1)
    cv2.putText(frame, "v : " + str(count['v']), (10, 310), cv2.FONT_HERSHEY_PLAIN, 1,
(0, 255, 255), 1)
    cv2.putText(frame, "w : " + str(count['w']), (10, 320), cv2.FONT_HERSHEY_PLAIN, 1,
(0, 255, 255), 1)
    cv2.putText(frame, "x : " + str(count['x']), (10, 330), cv2.FONT_HERSHEY_PLAIN, 1,
(0, 255, 255), 1)
    cv2.putText(frame, "y : " + str(count['y']), (10, 340), cv2.FONT_HERSHEY_PLAIN, 1,
(0, 255, 255), 1)
    cv2.putText(frame, "z : " + str(count['z']), (10, 350), cv2.FONT_HERSHEY_PLAIN, 1,
(0, 255, 255), 1)
    # Coordinates of the ROI
    x1 = int(0.5 * frame.shape[1])
    y1 = 10
    x2 = frame.shape[1] - 10
    y2 = int(0.5 * frame.shape[1])
    # Drawing the ROI
```

```
    # The increment/decrement by 1 is to compensate for the bounding box
    cv2.rectangle(frame, (220 - 1, 9), (620 + 1, 419), (255, 0, 0), 1)
    # Extracting the ROI
    roi = frame[10:410, 220:520]
    #    roi = cv2.resize(roi, (64, 64))

    cv2.imshow("Frame", frame)
    gray = cv2.cvtColor(roi, cv2.COLOR_BGR2GRAY)

    blur = cv2.GaussianBlur(gray, (5, 5), 2)
    # #blur = cv2.bilateralFilter(roi,9,75,75)

    th3 = cv2.adaptiveThreshold(blur, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
cv2.THRESH_BINARY_INV, 11, 2)
    ret, test_image = cv2.threshold(th3, minValue, 255, cv2.THRESH_BINARY_INV +
cv2.THRESH_OTSU)
    # time.sleep(5)

    test_image = cv2.resize(test_image, (300, 300))
    cv2.imshow("test", test_image)


    interrupt = cv2.waitKey(10)
    if interrupt & 0xFF == 27:  # esc key
        break
    if interrupt & 0xFF == ord('a'):
        cv2.imwrite(directory + 'A/' + str(count['a']) + '.jpg', roi)
    if interrupt & 0xFF == ord('b'):
        cv2.imwrite(directory + 'B/' + str(count['b']) + '.jpg', roi)
    if interrupt & 0xFF == ord('c'):
        cv2.imwrite(directory + 'C/' + str(count['c']) + '.jpg', roi)
    if interrupt & 0xFF == ord('d'):
        cv2.imwrite(directory + 'D/' + str(count['d']) + '.jpg', roi)
    if interrupt & 0xFF == ord('e'):
        cv2.imwrite(directory + 'E/' + str(count['e']) + '.jpg', roi)
    if interrupt & 0xFF == ord('f'):
        cv2.imwrite(directory + 'F/' + str(count['f']) + '.jpg', roi)
    if interrupt & 0xFF == ord('g'):
        cv2.imwrite(directory + 'G/' + str(count['g']) + '.jpg', roi)
    if interrupt & 0xFF == ord('h'):
        cv2.imwrite(directory + 'H/' + str(count['h']) + '.jpg', roi)
    if interrupt & 0xFF == ord('i'):
        cv2.imwrite(directory + 'I/' + str(count['i']) + '.jpg', roi)
    if interrupt & 0xFF == ord('j'):
        cv2.imwrite(directory + 'J/' + str(count['j']) + '.jpg', roi)
    if interrupt & 0xFF == ord('k'):
        cv2.imwrite(directory + 'K/' + str(count['k']) + '.jpg', roi)
    if interrupt & 0xFF == ord('l'):
        cv2.imwrite(directory + 'L/' + str(count['l']) + '.jpg', roi)
    if interrupt & 0xFF == ord('m'):
        cv2.imwrite(directory + 'M/' + str(count['m']) + '.jpg', roi)
    if interrupt & 0xFF == ord('n'):
        cv2.imwrite(directory + 'N/' + str(count['n']) + '.jpg', roi)
    if interrupt & 0xFF == ord('o'):
        cv2.imwrite(directory + 'O/' + str(count['o']) + '.jpg', roi)
    if interrupt & 0xFF == ord('p'):
        cv2.imwrite(directory + 'P/' + str(count['p']) + '.jpg', roi)
    if interrupt & 0xFF == ord('q'):
        cv2.imwrite(directory + 'Q/' + str(count['q']) + '.jpg', roi)
    if interrupt & 0xFF == ord('r'):
        cv2.imwrite(directory + 'R/' + str(count['r']) + '.jpg', roi)
    if interrupt & 0xFF == ord('s'):
        cv2.imwrite(directory + 'S/' + str(count['s']) + '.jpg', roi)
    if interrupt & 0xFF == ord('t'):
        cv2.imwrite(directory + 'T/' + str(count['t']) + '.jpg', roi)
```

```
    if interrupt & 0xFF == ord('u'):
        cv2.imwrite(directory + 'U/' + str(count['u']) + '.jpg', roi)
    if interrupt & 0xFF == ord('v'):
        cv2.imwrite(directory + 'V/' + str(count['v']) + '.jpg', roi)
    if interrupt & 0xFF == ord('w'):
        cv2.imwrite(directory + 'W/' + str(count['w']) + '.jpg', roi)
    if interrupt & 0xFF == ord('x'):
        cv2.imwrite(directory + 'X/' + str(count['x']) + '.jpg', roi)
    if interrupt & 0xFF == ord('y'):
        cv2.imwrite(directory + 'Y/' + str(count['y']) + '.jpg', roi)
    if interrupt & 0xFF == ord('z'):
        cv2.imwrite(directory + 'Z/' + str(count['z']) + '.jpg', roi)
cap.release()
cv2.destroyAllWindows()
```

Image_preprocessing.py

```
mport cv2

minValue = 70


def func(path):
    frame = cv2.imread(path)

    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    blur = cv2.GaussianBlur(gray, (5, 5), 2)

    th3 = cv2.adaptiveThreshold(blur, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
cv2.THRESH_BINARY_INV, 11, 2)
    ret, res = cv2.threshold(th3, minValue, 255, cv2.THRESH_BINARY_INV +
cv2.THRESH_OTSU)
    return res
```

preprocessing.py

```
from image_processing import func

if not os.path.exists("data2"):
    os.makedirs("data2")
if not os.path.exists("data2/train"):
    os.makedirs("data2/train")
if not os.path.exists("data2/test"):
    os.makedirs("data2/test")

path = "C:\\Users\\D VISHAL\\PycharmProjects\\hand_gesture_recognition\\data\\train"
#path = "C:\\Users\\D VISHAL\\PycharmProjects\\hand_gesture_recognition\\data\\test"
path1 = "data2"
a = ['label']

for i in range(64 * 64):
    a.append("pixel" + str(i))


label = 0
var = 0
c1 = 0
c2 = 0

for (dirpath, dirnames, filenames) in os.walk(path):
    for dirname in dirnames:
        print(dirname)
        for (direcpath, direcnames, files) in os.walk(path + "/" + dirname):
            if not os.path.exists(path1 + "/train/" + dirname):
                os.makedirs(path1 + "/train/" + dirname)
```

```
            if not os.path.exists(path1 + "/test/" + dirname):
                os.makedirs(path1 + "/test/" + dirname)
            # num=0.75*len(files)
            num = 100000000000000000
            i = 0
            for file in files:
                var += 1
                actual_path = path + "/" + dirname + "/" + file
                actual_path1 = path1 + "/" + "train/" + dirname + "/" + file
                actual_path2 = path1 + "/" + "test/" + dirname + "/" + file
                img = cv2.imread(actual_path, 0)
                bw_image = func(actual_path)
                if i < num:
                    c1 += 1
                    cv2.imwrite(actual_path1, bw_image)
                else:
                    c2 += 1
                    cv2.imwrite(actual_path2, bw_image)

                i = i + 1

        label = label + 1
print(var)
print(c1)
print(c2)
```

train.py

```
rom keras.models import Sequential
from keras.layers import Convolution2D
from keras.layers import MaxPooling2D
from keras.layers import Flatten
from keras.layers import Dense , Dropout
import os
os.environ["CUDA_VISIBLE_DEVICES"] = "1"
sz = 128

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)
training_set = train_datagen.flow_from_directory('C:\\Users\\D
VISHAL\\PycharmProjects\\hand_gesture_recognition\\data2\\train',
                                                 target_size=(sz, sz),
                                                 batch_size=10,
                                                 color_mode='grayscale',
                                                 class_mode='categorical')
test_set = test_datagen.flow_from_directory('C:\\Users\\D
VISHAL\\PycharmProjects\\hand_gesture_recognition\\data2\\test',
                                            target_size=(sz , sz),
                                            batch_size=10,
                                            color_mode='grayscale',
                                            class_mode='categorical')
# Initializing the CNN
classifier = Sequential()
# First convolution layer and pooling
classifier.add(Convolution2D(32, (3, 3), input_shape=(sz, sz, 1), activation='relu'))
classifier.add(MaxPooling2D(pool_size=(2, 2)))
# Second convolution layer and pooling
classifier.add(Convolution2D(32, (3, 3), activation='relu'))
```

```
# input_shape is going to be the pooled feature maps from the previous convolution
layer
classifier.add(MaxPooling2D(pool_size=(2, 2)))
classifier.add(Convolution2D(64, (3, 3), activation='relu'))
# input_shape is going to be the pooled feature maps from the previous convolution
layer
classifier.add(MaxPooling2D(pool_size=(2, 2)))
# Flattening the layers
classifier.add(Flatten())

# Adding a fully connected layer
classifier.add(Dense(units=64, activation='relu'))
classifier.add(Dropout(0.40))
classifier.add(Dense(units=96, activation='relu'))
classifier.add(Dropout(0.40))
classifier.add(Dense(units=64, activation='relu'))
classifier.add(Dense(units=27, activation='softmax')) # softmax for more than 2

# Compiling the CNN
classifier.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy']) # categorical_crossentropy for more than 2

# Step 2 - Preparing the train/test data and training the model
classifier.summary()
# Code copied from - https://keras.io/preprocessing/image/

classifier.fit_generator(
        training_set,
        steps_per_epoch=27000, # No of images in training set
        epochs=5,
        validation_data=test_set,
        validation_steps=8000)# No of images in test set

# Saving the model
model_json = classifier.to_json()
with open("model-bw.json", "w") as json_file:
    json_file.write(model_json)
print('Model Saved')
classifier.save_weights('model-bw.h5')
print('Weights saved')
```

final.py

```
from PIL import Image, ImageTk
import tkinter as tk
import cv2
import os
import numpy as np
from keras.models import model_from_json
import operator
import time
import sys, os
import matplotlib.pyplot as plt
from string import ascii_uppercase


class Application:
    def __init__(self):
        self.directory = 'model'
        self.vs = cv2.VideoCapture(0)
        self.current_image = None
        self.current_image2 = None

        self.json_file = open(self.directory + "model-bw.json", "r")
        self.model_json = self.json_file.read()
```

```
        self.json_file.close()
        self.loaded_model = model_from_json(self.model_json)
        self.loaded_model.load_weights(self.directory + "model-bw.h5")

        self.json_file_dru = open(self.directory + "model-bw_dru.json", "r")
        self.model_json_dru = self.json_file_dru.read()
        self.json_file_dru.close()
        self.loaded_model_dru = model_from_json(self.model_json_dru)
        self.loaded_model_dru.load_weights("model-bw_dru.h5")

        self.json_file_tkdi = open(self.directory + "model-bw_tkdi.json", "r")
        self.model_json_tkdi = self.json_file_tkdi.read()
        self.json_file_tkdi.close()
        self.loaded_model_tkdi = model_from_json(self.model_json_tkdi)
        self.loaded_model_tkdi.load_weights(self.directory + "model-bw_tkdi.h5")

        self.json_file_smn = open(self.directory + "model-bw_smn.json", "r")
        self.model_json_smn = self.json_file_smn.read()
        self.json_file_smn.close()
        self.loaded_model_smn = model_from_json(self.model_json_smn)
        self.loaded_model_smn.load_weights(self.directory + "model-bw_smn.h5")

        self.ct = {}
        self.ct['blank'] = 0
        self.blank_flag = 0
        for i in ascii_uppercase:
            self.ct[i] = 0
        print("Loaded model from disk")
        self.root = tk.Tk()
        self.root.title("Sign language to Text Converter")
        self.root.protocol('WM_DELETE_WINDOW', self.destructor)
        self.root.geometry("900x1100")
        self.panel = tk.Label(self.root)
        self.panel.place(x=135, y=10, width=640, height=640)
        self.panel2 = tk.Label(self.root)  # initialize image panel
        self.panel2.place(x=460, y=95, width=310, height=310)

        self.T = tk.Label(self.root)
        self.T.place(x=31, y=17)
        self.T.config(text="Sign Language to Text", font=("courier", 40, "bold"))
        self.panel3 = tk.Label(self.root)  # Current SYmbol
        self.panel3.place(x=500, y=640)
        self.T1 = tk.Label(self.root)
        self.T1.place(x=10, y=640)
        self.T1.config(text="Character :", font=("Courier", 40, "bold"))
        self.panel4 = tk.Label(self.root)  # Word
        self.panel4.place(x=220, y=700)
        self.T2 = tk.Label(self.root)
        self.T2.place(x=10, y=700)
        self.T2.config(text="Word :", font=("Courier", 40, "bold"))
        self.panel5 = tk.Label(self.root)  # Sentence
        self.panel5.place(x=350, y=760)
        self.T3 = tk.Label(self.root)
        self.T3.place(x=10, y=760)
        self.T3.config(text="Sentence :", font=("Courier", 40, "bold"))

        self.T4 = tk.Label(self.root)
        self.T4.place(x=250, y=820)
        self.T4.config(text="Suggestions", fg="red", font=("Courier", 40, "bold"))

        self.btcall = tk.Button(self.root, command=self.action_call, height=0, width=0)
        self.btcall.config(text="About", font=("Courier", 14))
        self.btcall.place(x=825, y=0)

        self.bt1 = tk.Button(self.root, command=self.action1, height=0, width=0)
```

```
        self.bt1.place(x=26, y=890)
        # self.bt1.grid(padx = 10, pady = 10)
        self.bt2 = tk.Button(self.root, command=self.action2, height=0, width=0)
        self.bt2.place(x=325, y=890)
        # self.panel3.place(x = 10,y=660)
        # self.bt2.grid(row = 4, column = 1, columnspan = 1, padx = 10, pady = 10,
sticky = tk.NW)
        self.bt3 = tk.Button(self.root, command=self.action3, height=0, width=0)
        self.bt3.place(x=625, y=890)
        # self.bt3.grid(row = 4, column = 2, columnspan = 1, padx = 10, pady = 10,
sticky = tk.NW)
        self.bt4 = tk.Button(self.root, command=self.action4, height=0, width=0)
        self.bt4.place(x=125, y=950)
        # self.bt4.grid(row = bt1, column = 0, columnspan = 1, padx = 10, pady = 10,
sticky = tk.N)
        self.bt5 = tk.Button(self.root, command=self.action5, height=0, width=0)
        self.bt5.place(x=425, y=950)
        # self.bt5.grid(row = 5, column = 1, columnspan = 1, padx = 10, pady = 10,
sticky = tk.N)
        self.str = ""
        self.word = ""
        self.current_symbol = "Empty"
        self.photo = "Empty"
        self.video_loop()

    def video_loop(self):
        ok, frame = self.vs.read()
        if ok:
            cv2image = cv2.flip(frame, 1)
            x1 = int(0.5 * frame.shape[1])
            y1 = 10
            x2 = frame.shape[1] - 10
            y2 = int(0.5 * frame.shape[1])
            cv2.rectangle(frame, (x1 - 1, y1 - 1), (x2 + 1, y2 + 1), (255, 0, 0), 1)
            cv2image = cv2.cvtColor(cv2image, cv2.COLOR_BGR2RGBA)
            self.current_image = Image.fromarray(cv2image)
            imgtk = ImageTk.PhotoImage(image=self.current_image)
            self.panel.imgtk = imgtk
            self.panel.config(image=imgtk)
            cv2image = cv2image[y1:y2, x1:x2]
            gray = cv2.cvtColor(cv2image, cv2.COLOR_BGR2GRAY)
            blur = cv2.GaussianBlur(gray, (5, 5), 2)
            th3 = cv2.adaptiveThreshold(blur, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
cv2.THRESH_BINARY_INV, 11, 2)
            ret, res = cv2.threshold(th3, 70, 255, cv2.THRESH_BINARY_INV +
cv2.THRESH_OTSU)
            self.predict(res)
            self.current_image2 = Image.fromarray(res)
            imgtk = ImageTk.PhotoImage(image=self.current_image2)
            self.panel2.imgtk = imgtk
            self.panel2.config(image=imgtk)
            self.panel3.config(text=self.current_symbol, font=("Courier", 50))
            self.panel4.config(text=self.word, font=("Courier", 40))
            self.panel5.config(text=self.str, font=("Courier", 40))
            predicts = self.hs.suggest(self.word)
            if (len(predicts) > 0):
                self.bt1.config(text=predicts[0], font=("Courier", 20))
            else:
                self.bt1.config(text="")
            if (len(predicts) > 1):
                self.bt2.config(text=predicts[1], font=("Courier", 20))
            else:
                self.bt2.config(text="")
            if (len(predicts) > 2):
                self.bt3.config(text=predicts[2], font=("Courier", 20))
```

```
            else:
                self.bt3.config(text="")
            if (len(predicts) > 3):
                self.bt4.config(text=predicts[3], font=("Courier", 20))
            else:
                self.bt4.config(text="")
            if (len(predicts) > 4):
                self.bt4.config(text=predicts[4], font=("Courier", 20))
            else:
                self.bt4.config(text="")
        self.root.after(30, self.video_loop)

    def predict(self, test_image):
        test_image = cv2.resize(test_image, (128, 128))
        result = self.loaded_model.predict(test_image.reshape(1, 128, 128, 1))
        result_dru = self.loaded_model_dru.predict(test_image.reshape(1, 128, 128, 1))
        result_tkdi = self.loaded_model_tkdi.predict(test_image.reshape(1, 128, 128,
1))
        result_smn = self.loaded_model_smn.predict(test_image.reshape(1, 128, 128, 1))
        prediction = {}
        prediction['blank'] = result[0][0]
        inde = 1
        for i in ascii_uppercase:
            prediction[i] = result[0][inde]
            inde += 1
        # LAYER 1
        prediction = sorted(prediction.items(), key=operator.itemgetter(1),
reverse=True)
        self.current_symbol = prediction[0][0]
        # LAYER 2
        if (self.current_symbol == 'D' or self.current_symbol == 'R' or
self.current_symbol == 'U'):
            prediction = {}
            prediction['D'] = result_dru[0][0]
            prediction['R'] = result_dru[0][1]
            prediction['U'] = result_dru[0][2]
            prediction = sorted(prediction.items(), key=operator.itemgetter(1),
reverse=True)
            self.current_symbol = prediction[0][0]

        if (
                self.current_symbol == 'D' or self.current_symbol == 'I' or
self.current_symbol == 'K' or self.current_symbol == 'T'):
            prediction = {}
            prediction['D'] = result_tkdi[0][0]
            prediction['I'] = result_tkdi[0][1]
            prediction['K'] = result_tkdi[0][2]
            prediction['T'] = result_tkdi[0][3]
            prediction = sorted(prediction.items(), key=operator.itemgetter(1),
reverse=True)
            self.current_symbol = prediction[0][0]

        if (self.current_symbol == 'M' or self.current_symbol == 'N' or
self.current_symbol == 'S'):
            prediction1 = {}
            prediction1['M'] = result_smn[0][0]
            prediction1['N'] = result_smn[0][1]
            prediction1['S'] = result_smn[0][2]
            prediction1 = sorted(prediction1.items(), key=operator.itemgetter(1),
reverse=True)
            if (prediction1[0][0] == 'S'):
                self.current_symbol = prediction1[0][0]
            else:
                self.current_symbol = prediction[0][0]
        if (self.current_symbol == 'blank'):
```

```
            for i in ascii_uppercase:
                self.ct[i] = 0
        self.ct[self.current_symbol] += 1
        if (self.ct[self.current_symbol] > 60):
            for i in ascii_uppercase:
                if i == self.current_symbol:
                    continue
                tmp = self.ct[self.current_symbol] - self.ct[i]
                if tmp < 0:
                    tmp *= -1
                if tmp <= 20:
                    self.ct['blank'] = 0
                    for i in ascii_uppercase:
                        self.ct[i] = 0
                    return
            self.ct['blank'] = 0
            for i in ascii_uppercase:
                self.ct[i] = 0
            if self.current_symbol == 'blank':
                if self.blank_flag == 0:
                    self.blank_flag = 1
                    if len(self.str) > 0:
                        self.str += " "
                    self.str += self.word
                    self.word = ""
            else:
                if (len(self.str) > 16):
                    self.str = ""
                self.blank_flag = 0
                self.word += self.current_symbol

    def action1(self):
        predicts = self.hs.suggest(self.word)
        if (len(predicts) > 0):
            self.word = ""
            self.str += " "
            self.str += predicts[0]

    def action2(self):
        predicts = self.hs.suggest(self.word)
        if (len(predicts) > 1):
            self.word = ""
            self.str += " "
            self.str += predicts[1]

    def action3(self):
        predicts = self.hs.suggest(self.word)
        if (len(predicts) > 2):
            self.word = ""
            self.str += " "
            self.str += predicts[2]

    def action4(self):
        predicts = self.hs.suggest(self.word)
        if (len(predicts) > 3):
            self.word = ""
            self.str += " "
            self.str += predicts[3]

    def action5(self):
        predicts = self.hs.suggest(self.word)
        if (len(predicts) > 4):
            self.word = ""
            self.str += " "
            self.str += predicts[4]
```

```
    def destructor(self):
        print("Closing Application...")
        self.root.destroy()
        self.vs.release()
        cv2.destroyAllWindows()

    def destructor1(self):
        print("Closing Application...")
        self.root1.destroy()

    def action_call(self):

        self.root1 = tk.Toplevel(self.root)
        self.root1.title("About")
        self.root1.protocol('WM_DELETE_WINDOW', self.destructor1)
        self.root1.geometry("900x900")

        # img = cv2.imread("Pictures/sir.jpg", 1)
        # # img = cv2.resize(img, (300, 300))
        # cv2.imwrite("Pictures/sir.png", img)
        # return

        self.tx = tk.Label(self.root1)
        self.tx.place(x=330, y=20)
        self.tx.config(text="Efforts By", fg="red", font=("Courier", 30, "bold"))

        self.photo1 = tk.PhotoImage(file='Pictures/ravi.png')
        self.w1 = tk.Label(self.root1, image=self.photo1)
        self.w1.place(x=20, y=105)
        self.tx6 = tk.Label(self.root1)
        self.tx6.place(x=20, y=250)
        self.tx6.config(text="RC\nIIT2016141", font=("Courier", 15, "bold"))

        self.photo2 = tk.PhotoImage(file='Pictures/nitin.png')
        self.w2 = tk.Label(self.root1, image=self.photo2)
        self.w2.place(x=200, y=105)
        self.tx2 = tk.Label(self.root1)
        self.tx2.place(x=200, y=250)
        self.tx2.config(text="Nitin\nIIT2016132", font=("Courier", 15, "bold"))

        self.photo3 = tk.PhotoImage(file='Pictures/luv.png')
        self.w3 = tk.Label(self.root1, image=self.photo3)
        self.w3.place(x=380, y=105)
        self.tx3 = tk.Label(self.root1)
        self.tx3.place(x=380, y=250)
        self.tx3.config(text="Luv\nIIT2016085", font=("Courier", 15, "bold"))

        self.photo4 = tk.PhotoImage(file='Pictures/sheldon.png')
        self.w4 = tk.Label(self.root1, image=self.photo4)
        self.w4.place(x=560, y=105)
        self.tx4 = tk.Label(self.root1)
        self.tx4.place(x=560, y=250)
        self.tx4.config(text="Sheldon\nIIT2016137", font=("Courier", 15, "bold"))

        self.photo5 = tk.PhotoImage(file='Pictures/sid.png')
        self.w5 = tk.Label(self.root1, image=self.photo5)
        self.w5.place(x=740, y=105)
        self.tx5 = tk.Label(self.root1)
        self.tx5.place(x=740, y=250)
        self.tx5.config(text="Siddhant\nIIT2016069", font=("Courier", 15, "bold"))

        self.tx7 = tk.Label(self.root1)
        self.tx7.place(x=170, y=360)
        self.tx7.config(text="Under the supervision of", fg="red", font=("Courier", 30,
```

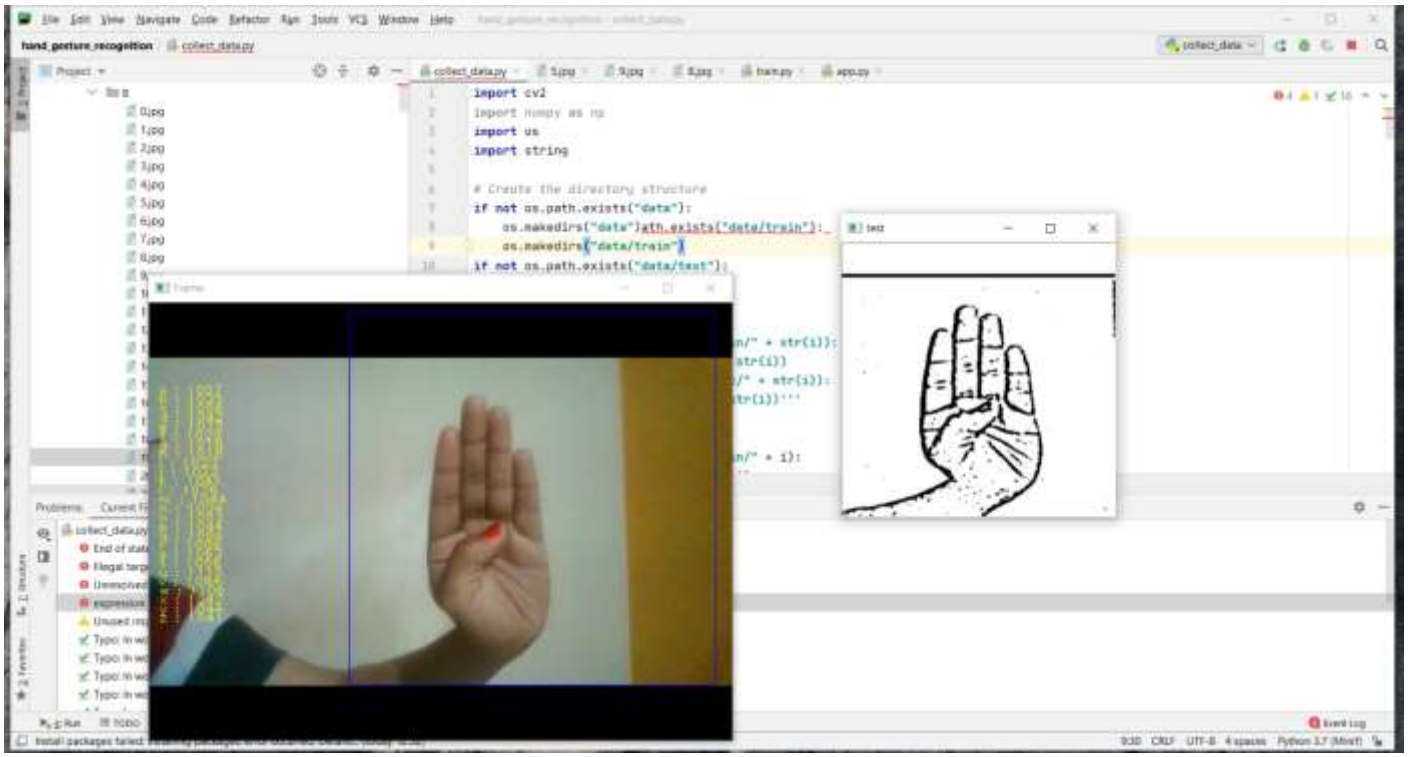
```
"bold"))

        self.photo6 = tk.PhotoImage(file='Pictures/sir.png')
        self.w6 = tk.Label(self.root1, image=self.photo6)
        self.w6.place(x=350, y=420)
        self.tx6 = tk.Label(self.root1)
        self.tx6.place(x=230, y=670)
        self.tx6.config(text="Dr. Vrijendra Singh", font=("Courier", 30, "bold"))


print("Starting Application...")
pba = Application()
pba.root.mainloop()
```

4.3 Data Set Description

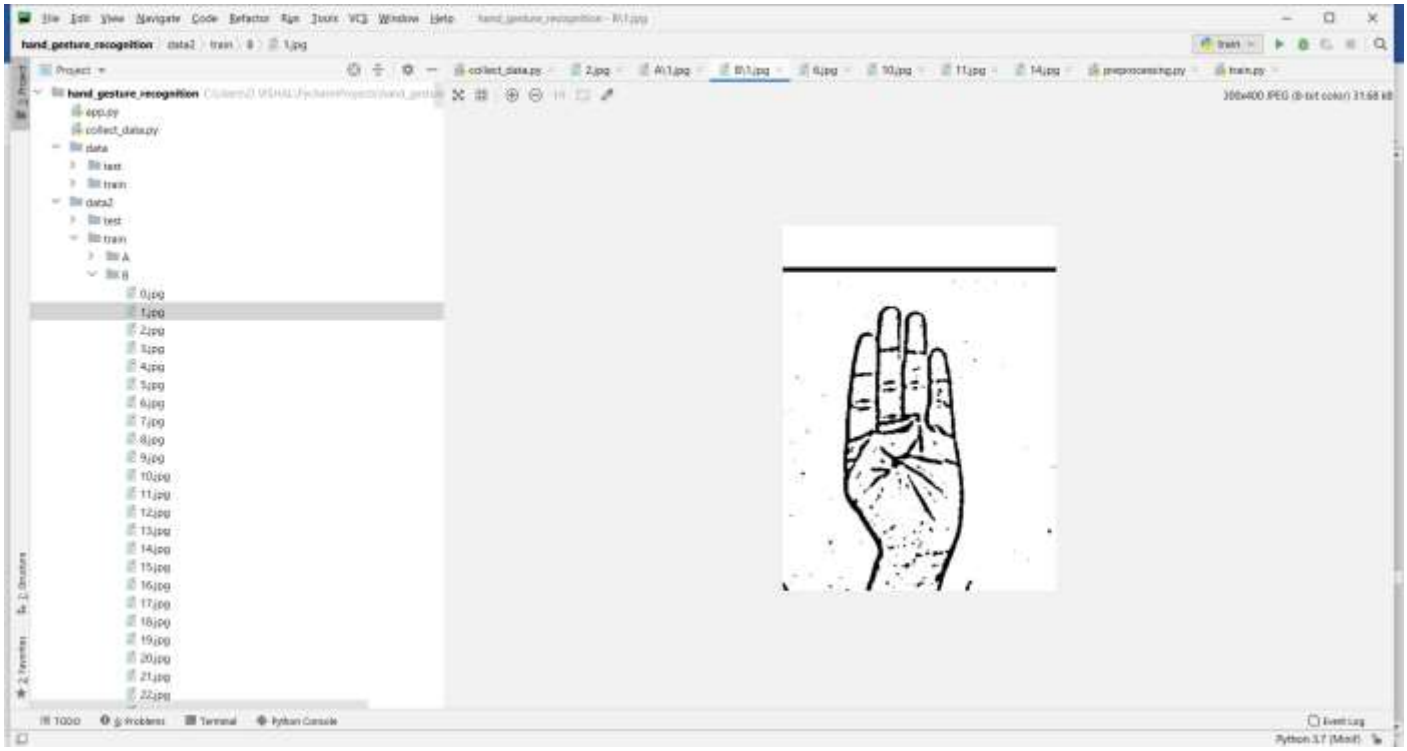
For this project, we created our own **American Sign Language (ASL) dataset :** It is a collection of 27,000 images, 1000 images for each of the 26 classes. These gestures are recorded for a total of 800 for training and 200 for testing. The gestures include alphabets A-Z except ‘J’ and ‘Z’, because these require movements of hand and thus cannot be captured in the form of an image.

5 RESULTS OUTPUTS AND DISCUSSION

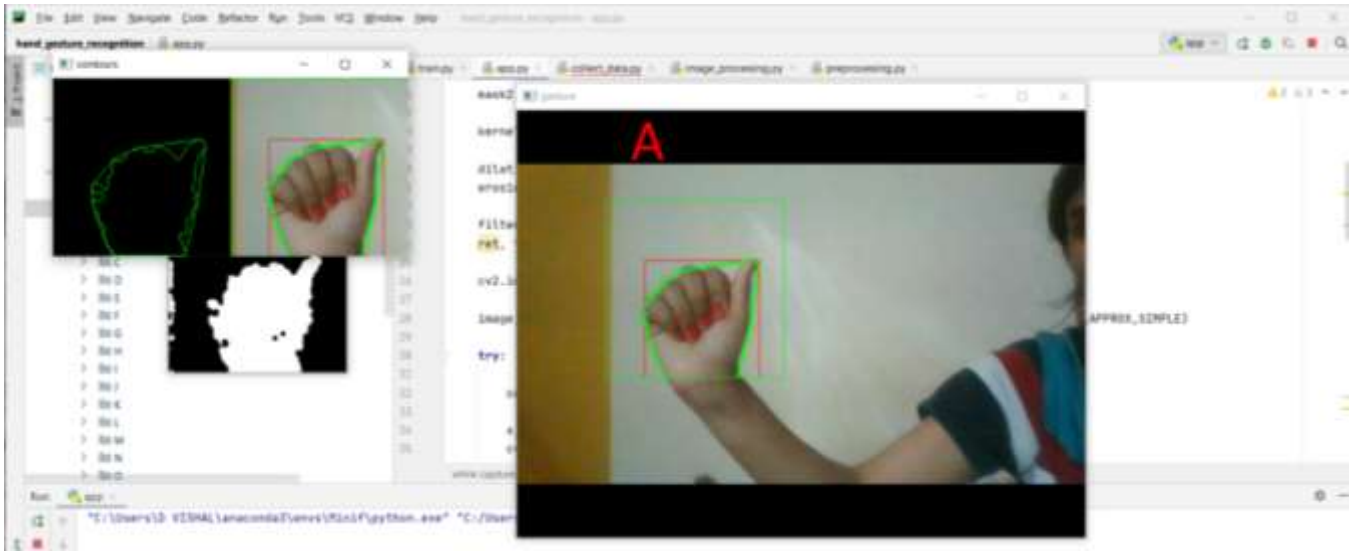
- On running collect_data.py .we can create gestures by placing the hand gesture in frame ,whereas showing the extracting features in the therefore on clicking the respected letter we can store the corresponding gesture in the particular folder. Hence frame shows the count of how many gestures has captured .

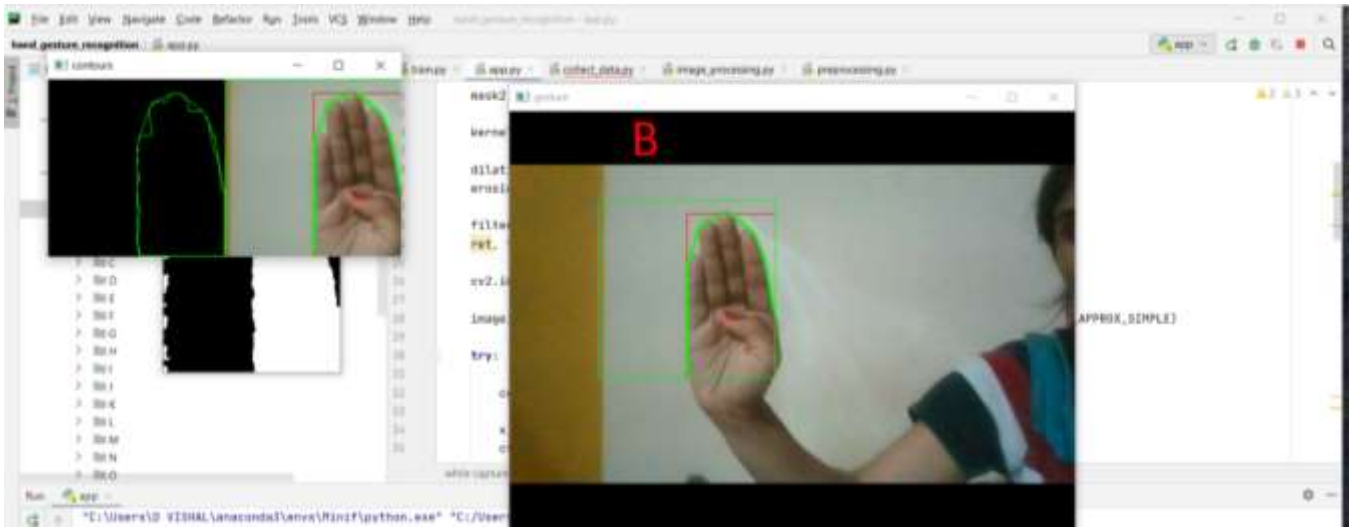


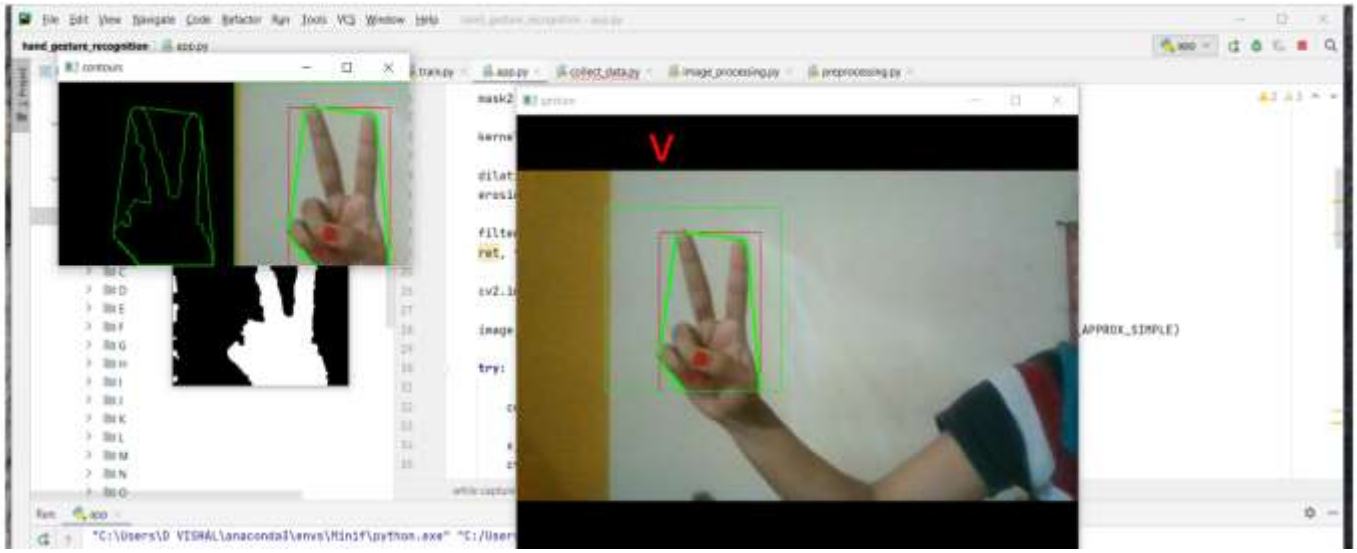
- On running preprocessing.py the captured images that are in RGB(red GREEN BLUE) schema i.e, colored images are converted to gray scale. On applying Guassian Blur filter the ROI(region of interest) is created thereby extracting important features from the captured image. Later the preprocessed images are stored in new directory folder .

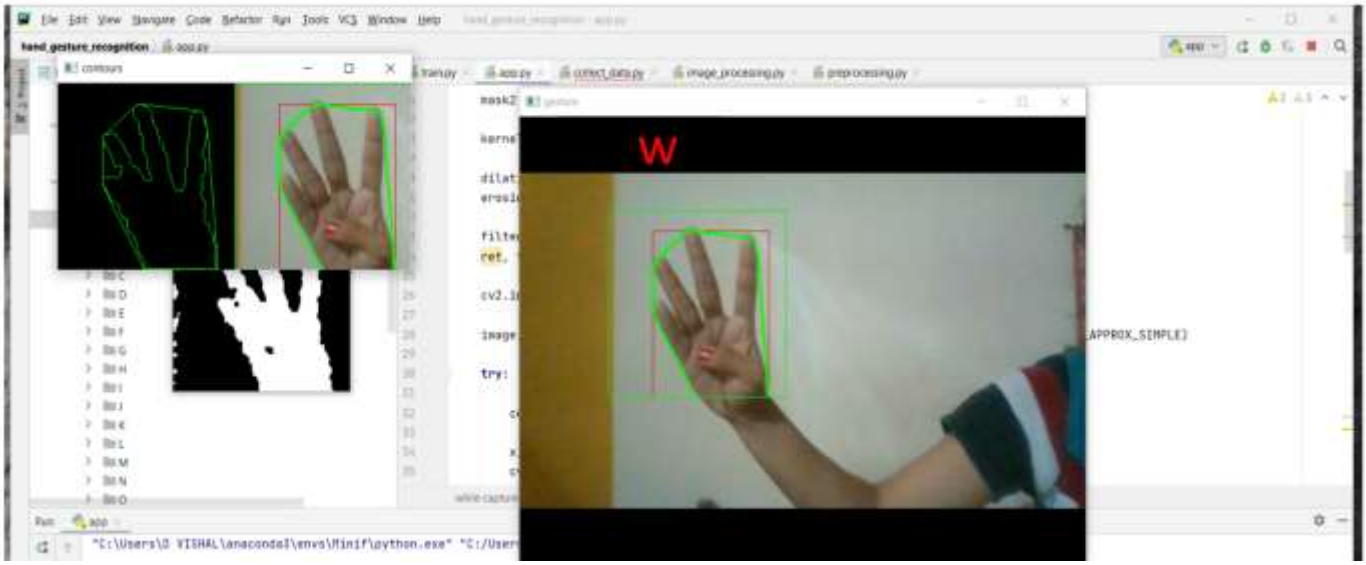


- Now on train.py .the model is created by taking the 800 images each from 26 classes of ASL dataset thereby ,taking 200 images for testing the model .
- Later after successful execution of final.py , the model is ready to detect the image from the real Time video capture .
- Therefore the predicted images are shown like below.


A


B


V



6 CONCLUSIONS/RECOMMENDATIONS

6.1 Conclusions

From this project/application 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 ASL 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.

We are able to improve our prediction after implementing two layers of algorithms in which we verify and predict symbols which are more similar to each other. This way we are able to detect almost all the symbols provided that they are shown properly, there is no noise in the background and lighting is adequate.

6.1.1 Limitations

The model works well only in good lighting conditions. Plain background is needed for the model to detect with accuracy. As Sign language translation technologies are limited in the same way as spoken language

translation. None can translate with 100% accuracy. In fact, sign language translation technologies are far behind their spoken language counterparts. This is, in no trivial way, due to the fact that signed languages have multiple articulators. Where spoken languages are articulated through the vocal tract, signed languages are articulated through the hands, arms, head, shoulders, torso, and parts of the face. This multi-channel articulation makes translating sign languages very difficult. An additional challenge for sign language MT is the fact that there is no formal written format for signed languages. There are notations systems but no writing system has been adopted widely enough, by the international Deaf community, that it could be considered the 'written form' of a given sign language. Sign Languages then are recorded in various video formats.

6.2 Recommendations /Future Work/Future Scope

We are planning to achieve higher accuracy even in case of complex backgrounds by trying out various background subtraction algorithms. We are also thinking of improving the preprocessing to predict gestures in low light conditions with a higher accuracy.

It can be integrated with various search engines and texting application such as google, WhatsApp. So that even the illiterate people could be able to chat with other persons, or query something from web just with the help of gesture.

This project is working on image currently, further development can lead to detecting the motion of video sequence and assigning it to a meaningful sentence with TTS assistance.User dependent model using pre-training. Pre-training the model on a larger dataset (e.g. ILSRVC), that consists of around 14,000 classes, and then fine-tuning it with ISL dataset, so that the model can show good results even when trained with a small dataset.

REFERENCES

[1]T. Yang, Y. Xu, and “A. , Hidden Markov Model for Gesture Recognition”, CMU-RI-TR-94 10, Robotics Institute, Carnegie Mellon Univ.,Pittsburgh,PA, May 1994.

[2]Pujan Ziaie, Thomas M¨uller , Mary Ellen Foster , and Alois Knoll“A Na¨ıve Bayes Munich,Dept. of Informatics VI, Robotics and Embedded Systems,Boltzmannstr. 3, DE-85748 Garching, Germany.
[3]https://docs.opencv.org/2.4/doc/tutorials/imgproc/gausian_median_blur_b
ilateral_filter/gausian_median_blur_bilateral_filter.html

[4]Mohammed Waleed Kalous, Machine recognition of Auslan signs using PowerGloves: Towards large-lexicon recognition of sign language.

[5]aeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks-Part-2/

[6]http://www-i6.informatik.rwth-aachen.de/~dreuw/database.php

[7] Pigou L., Dieleman S., Kindermans PJ., Schrauwen B. (2015) Sign Language Recognition Using Convolutional Neural Networks. In: Agapito L., Bronstein M., Rother C. (eds) Computer Vision - ECCV 2014 Workshops. ECCV 2014. Lecture Notes in Computer Science, vol 8925. Springer, Cham

[8]Zaki, M.M., Shaheen, S.I.: Sign language recognition using a combination of new vision based features. Pattern Recognition Letters 32(4), 572–577 (2011) 25

[9] N. Mukai, N. Harada and Y. Chang, "Japanese Fingerspelling Recognition Based on Classification Tree and Machine Learning," 2017 Nicograph International (NicoInt), Kyoto, Japan, 2017, pp. 19-24. doi:10.1109/NICOInt.2017.9

[10]Byeongkeun Kang , Subarna Tripathi , Truong Q. Nguyen ”Real-time sign language fingerspelling recognition using convolutional neural networks from depth map” 2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR)

[11]https://opencv.org/

[12]https://en.wikipedia.org/wiki/TensorFlow

[13]https://en.wikipedia.org/wiki/Convolutional_neural_network