VISVESVARAYA TECHNOLOGICAL UNIVERSITY

JNANASANGAMA, BELAGAVI - 590018

A

PROJECT REPORT

On

SIGN-LANGUAGE RECOGNITION SYSTEM

Submitted in partial fulfilment for requirement for the award of the degree of

BACHELOR OF ENGINEERING
IN
COMPUTER SCIENCE AND ENGINEERING

BY

NIHAR HEGDE	**4KV19CS051**
PRATHIKA MUTHAMMA A D	**4KV19CS064**
SOMANNA A S	**4KV19CS073**
THUSHAR S PAWAR	**4KV19CS081**

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

K.V.G. COLLEGE OF ENGINEERING SULLIA, D.K. - 574 327

2021-2022

K.V.G. COLLEGE OF ENGINEERING, SULLIA, D.K.

(AFFILIATED TO VISVESVARAYA TECHNOLOGICAL UNIVERSITY, BELAGAVI)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

*Certified that the project work entitled "**SIGN-LANGUAGE RECOGNITION SYSTEM**" carried out by,*

NIHAR HEGDE	***4KV19CS051***
PRATHIKA MUTHAMMA A D	***4KV19CS064***
SOMANNA A S	***4KV19CS073***
THUSHAR S PAWAR	***4KV19CS081***

*The bonafide students of K.V.G. COLLEGE OF ENGINEERING in partial fulfillment for the award of **BACHELOR OF ENGINEERING** in **COMPUTER SCIENCE AND ENGINEERING** of the **VISVESVARAYA TECHNOLOGICAL UNIVERSITY, BELAGAVI** during the year 2021-22. It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in the Report deposited in the departmental library. The project report has been approved as it been satisfying the academic requirement in respect of project work prescribed for the Bachelor of Engineering Degree.*

Prof. VENKATESH U C
Associate Professor
Project Guide & Co-ordinator
Dept. of CS&E

Dr. SMITHA M L
Professor
Dept. of CS&E

Dr. UJWAL U. J
Head of the Department
Dept. of CS&E

Dr. SURESHA V
The Principal
KVGCE, Sullia

Name of the Examiners **Signature with Date**

1.

2.

ACKNOWLEDGEMENT

We are grateful to our institution K.V.G College of Engineering, Sullia with its ideas and inspiration for having provided us the facilities that have made this project a success.

We express our sincere gratitude and to our Project Guide **Prof. VENKATESH U.C**, Professor, Department of Computer Science and Engineering for guiding the work carried out by us and for his encouragement and support.

We are deeply indebted to project coordinator **Dr. SMITHA M L**, Professor, Department of CS&E and project coordinator **Prof. VENKATESH U.C**, Associate Professor, Department of CS&E, who guided us in the successful completion of this project.

We thankful **Dr. UJWAL U.J**, Head of the Dept. of Computer Science and Engineering, for extending their support during course of this investigation.

We express our heart full thanks to the principal, **Dr. SURESHA V** for providing congenial to work on it.

We express our heart full thanks to **Dr. RENUKA PRASAD K.V**, General Secretory AOLE, Sullia for providing congenial to work on it.

We are deeply indebted to late **Dr. KURUNJI VENKATARAMANA GOWDA**, President of AOLE, for having provided environment with all facilities that helped us in completing this project.

We should like to extend our thanks to all the teaching and non-teaching staff members and the students of CS&E department for their help, advice and support.

ABSTRACT

A sign language recognition system built using computer vision and machine learning methods is presented in this research. Hand gestures are recognized by the system, which then translates them into written language. The suggested method extracts elements from video input and categorizes them as Sign Language motions using a convolutional neural network (CNN) architecture. The system is an effective tool for communication between the deaf and hearing communities because of its excellent precision and real-time performance. The suggested system has a number of potential uses, including assistive technology for the deaf, automatic sign language translation, and instructional aids for teaching sign language. The testing findings show how proficiently and reliably the suggested system can recognize Hand Gestures.

TABLE OF CONTENTS

1. INTRODUCTION

Sign Language is mainly used by deaf (hard hearing) and dumb (cannot talk) people to exchange information between their own community and with other people. It is a language where people use their hand gestures to communicate as they can't speak or hear. Sign Language Recognition (SLR) deals with recognizing the hand gestures acquisition and continues till text or speech is generated for corresponding hand gestures.

The ordinary people cannot usually understand the sign language. Hence it is difficult for the deaf and dumb people to communicate with the world.

This project mainly helps to recognize the sign languages and translate it to English so that it is easy to understand and everyone can communicate with each other. Analysing the sign language and translating it to English and give the output in the form of text or audio. The proposed system has to properly recognize the hand sign and translate them.

1. LITERATURE SURVEY

Mahesh Kumar et al. [1] proposed a system which can recognize 26 hand gestures of **Indian sign language based on Linear Discriminant Analysis (LDA)** [11]. Pre-processing steps such as skin segmentation and morphological operations are applied on the dataset. Skin segmentation was carried out using otsu algorithm. Linear discriminant analysis is used for feature extraction. Each gesture is represented as a column vector in training phase which is then normalized with respect to average gesture. The algorithm finds the eigenvectors of the covariance matrix of normalized gestures. In recognition phase, subject vector is normalized with respect to average gesture and then projected onto gesture space using eigenvector matrix. Euclidean distance is computed between this projection and all known projections. The minimum value of these comparisons is selected.

Kshitij Bantupalli and Ying Xie et al. [2] worked on **American sign language recognition system** which works on video sequences based on CNN, LSTM and RNN. A CNN model named Inception was used to extract spatial features from frames, LSTM for longer time dependencies and RNN to extract temporal features. Various experiments were conducted with varying sample sizes and dataset consists of 100 different signs performed by 5 signers and maximum accuracy of 93% was obtained. Sequence is then fed to a LSTM for longer time dependencies. Outputs of softmax layer and maxpooling layer are fed to RNN architecture to extract temporal features from softmax layer.

Neelam K. Gilorkar, Manisha M. Ingle et al. [3] Author proposed a **Real time vision-based system for hand gesture recognition for human computer interaction in many applications**. The system can recognize 35 different hand gestures given by Indian and American Sign Language or ISL and ASL at faster rate with virtuous accuracy. RGB-to-GRAY segmentation technique was used to minimize the chances of false detection. Authors proposed a method of improvised Scale Invariant Feature Transform (SIFT) and same was used to extract features. The system is model using MATLAB. To design and efficient user-friendly hand gesture recognition system, a GUI model has been implemented.

Neelam K. Gilorkar, Manisha M. Ingle et al. [4]: **A Review on Feature Extraction for Indian and American Sign Language** in Paper presented the recent research and development of sign language based on manual communication and body language. Sign language recognition system typically elaborate three steps pre-processing, feature extraction and classification. Classification methods used for recognition are Neural Network (NN), Support Vector Machine (SVM), Hidden Markov Models (HMM), Scale Invariant Feature Transform (SIFT), etc.

Oscar Kellar et al**.** [5] introduced a **Hybrid CNN-HMM for sign language recognition**. They conducted experiments on three datasets namely RWTH-PHOENIX-Weather 2012 [19], RWTH-PHOENIX-Weather Multisigner 2014 [20] and SIGNUM single signer [21]. Training and validation set have a ratio of 10 to 1. After the CNN training is finished a softmax layer is added and results are used in HMM as observation probabilities.

G. Anantha Rao et al**.** [6] proposes an **Indian sign language gesture recognition using convolutional neural network**. This system works on videos captured from a mobile's front camera. Dataset is created manually for 200 ISL signs. CNN training is performed with 3 different datasets. In the first batch, dataset of only one set is given as input. Second batch has 2 sets of training data and third batch respectively has 3 sets of training data. Average recognition rate of this CNN model is 92.88%.

Rahaman et al**.** [7]: proposed a **Real-time camera-based sign language recognition system.** The dataset contains 7200 double handed Bengali signs which include signs of 6 vowels and 30 consonants. Features of finger position and fingertip were extracted and signs were classified using KNN. The disadvantage of the system is that it is not able to segment hand area accurately if objects with skin color appears.

Nakul Nagpal, Dr.Arun Mitra, Dr.Pankaj Agrawal et al. [8]: Design Issue and Proposed **Implementation of Communication Aid for Deaf & Dumb People** in In this paper author

proposed a system to aid communication of deaf and dumb people communication using Indian sign language (ISL) with normal people where hand gestures will be converted into appropriate text message. Main objective is to design an algorithm to convert dynamic gesture to text at real time finally after testing is done the system will be implemented on android platform and will be available as an application for smart phone and tablet pc.

Mathavan Suresh Anand, Nagarajan Mohan Kumar, Angappan Kumaresan et al [9]: An **Efficient Framework for Indian Sign Language Recognition** Using Wavelet Transform The proposed ISLR system is considered as a pattern recognition technique that has two important modules: feature extraction and classification. The joint use of Discrete Wavelet Transform (DWT) based feature extraction and nearest neighbour classifier is used to recognize the sign language. The experimental results show that the proposed hand gesture recognition system achieves maximum 99.23% classification accuracy while using cosine distance classifier.

Tanuj Bohra et al. [10] proposed a **Real-time two-way sign language communication system** built using image processing, deep learning and computer vision. Techniques such as hand detection, skin colour segmentation, median blur and contour detection are performed on images in the dataset for better results. CNN model trained with a large dataset for 40 classes and was able to predict 17600 test images in 14 seconds with an accuracy of 99%.

2. SYSTEM ANALYSIS

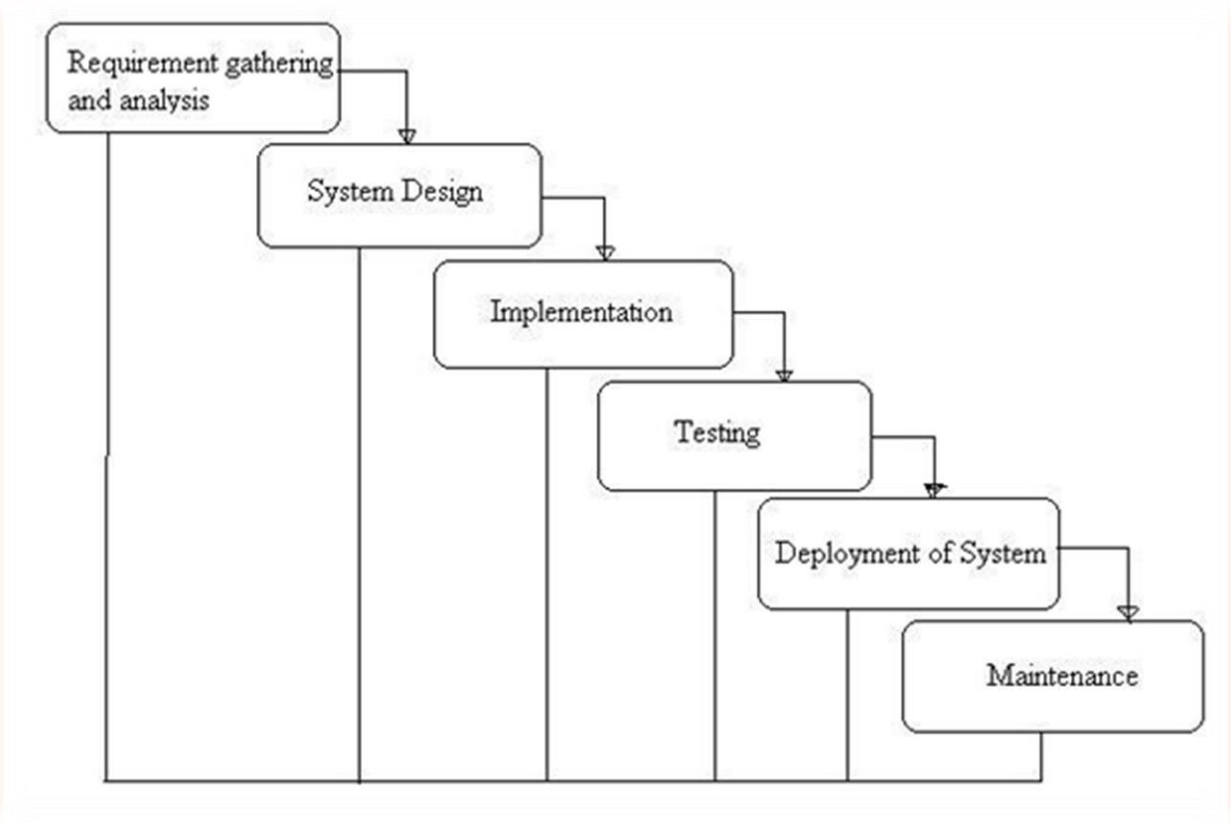

Fig: 1 Project SDLC

- Project Requisites Accumulating and Analysis
- Application System Design
- Practical Implementation
- Manual Testing of My Application
- Application Deployment of System
- Maintenance of the Project

Requisites Accumulating and Analysis

This stage is the first and foremost stage of all projects and as our project is a academic level project we amassed many IEEE papers and we took references from these papers and performed literature survey and then amassed all the requirements for this project.

System Design

In System Design we have designed the main user interface of the system that the user can use. And the machine learning model is also designed in the system design. The class diagram gives information about the different classes in the project. When it comes to our project the most important in system design is the machine learning model.

Implementation

The Implementation is Phase where we endeavor to give the practical output of the work done in designing stage and most of Coding in Business logic lay comes into action in this stage is the main and crucial part of the project.

Unit Testing

It is done by the developer itself in every stage of the project and fine-tuning the bug and module predicated additionally done by the developer only here we are going to solve all the runtime errors.

Deployment of System

Once the project development is complete, we will come to the deployment part. In the case of our project, we deployed it locally in our college systems with all the necessary software in a windows system.

Maintenance

The Maintenance of our Project is one time process.

3.1 Functional Requirements

- Open Web Camera
- Detect, segment and preprocess the hands
- Recognize the gesture
- Display the output

3.2 Non-Functional Requirements

- The System cannot predict if the capture images if of very poor quality.
- It cannot predict accurately if the sign is very complex.
- Due to illumination constraints.

3. RELATED WORK

4.1 Existing System:

Most of the existing Sign language recognition system do not have the functionality to train new gestures with good accuracy in the system.

4.2 Proposed System:

Here we have developed a sign language system that can recognize the pre trained gestures. And it also has the functionality of training new gestures in the UI by entering the logging mode.

Development Tools

Hardware requirements:

Processer	:	Intel i5
Ram	:	Min 4 GB
Hard Disk	:	Min 100 GB

Software requirements:

Operating System	:	Windows 10 and above
Technology	:	Python 3.9
IDE	:	PyCharm
Libraries	:	Tensorflow, OpenCV, Media Pipe

4. IMPLEMENTATION

5.1 System Architecture:

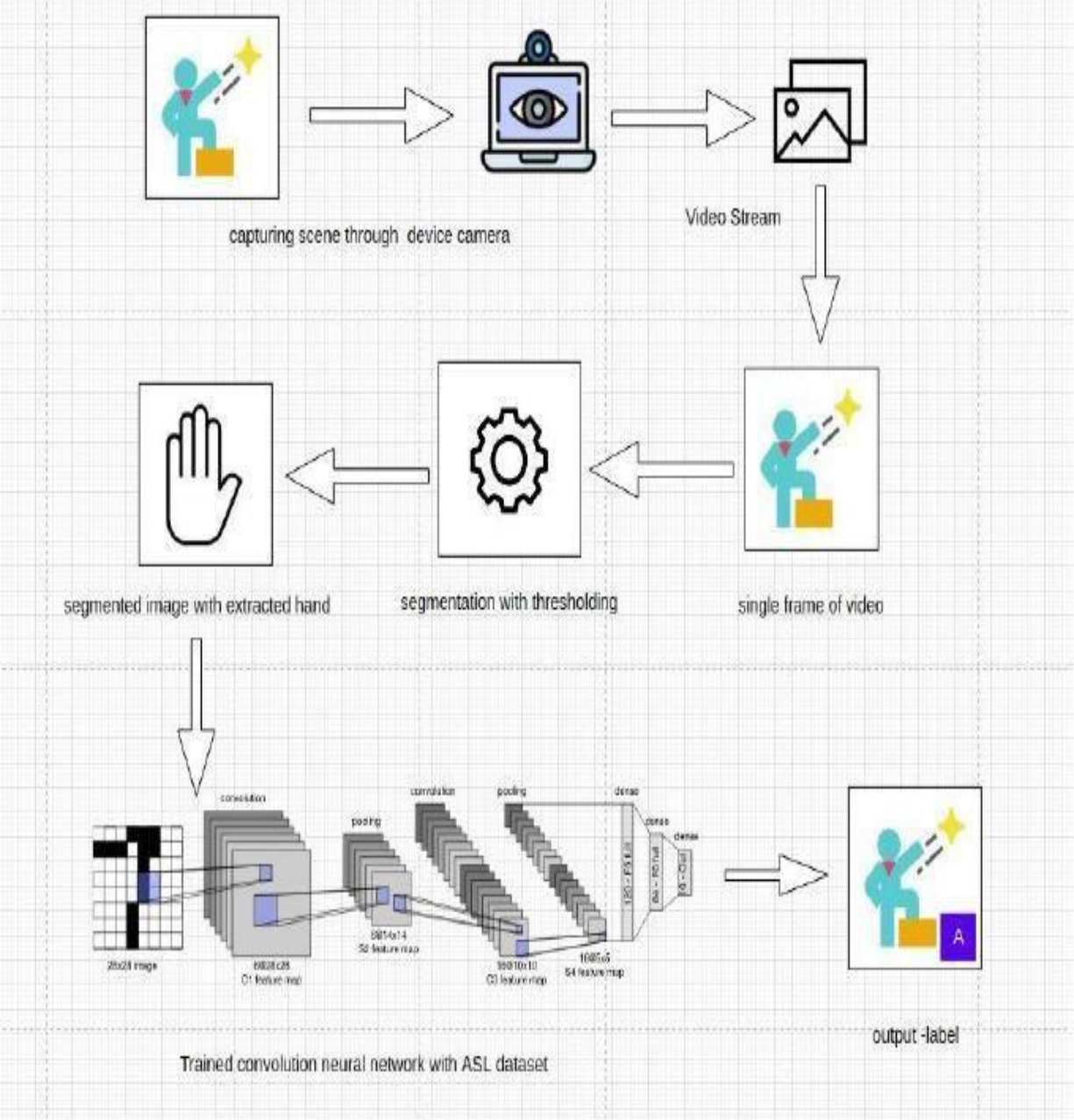

Fig: 2 System architecture

The system captures the images and video stream through a web camera and converts the videos to frames. The images and frames go through some preprocessing like segmentation and

thresholding. The hand is then extracted from the preprocessed date. Then the preprocessed data is fed into a pre trained CNN neural network, and the output is displayed.

5.2 Preprocessing:

Preprocessing input raw image in the context of Sign language recognition involves acquiring the Hand and standardizing images in a format compatible with the CNN architecture employed. Each CNN has a different input size requirement. The camera used, the environment the picture or video was taken can have impact on the final result so preprocessing is necessary.

5.3 Train New Gestures:

This Project can be used to train new gestures. After entering the logging mode perform the required gesture and capture the data. The amount of data captured can have a significant impact on the accuracy of the result. After Capturing the data, it is then it is preprocessed. The preprocessed data is then trained by a Convolutional Neural Network.

5.4 Recognize Gesture:

When a user performs a gesture in front of the camera it should recognize the gesture and give output as text.

5.5 Sample Code

```
import csv

import copy

import argparse

import itertools

from collections import Counter

from collections import deque

import cv2 as cv

import numpy as np

import mediapipe as mp

from utils import CvFpsCalc

from model import KeyPointClassifier

from model import PointHistoryClassifier


def get_args():

    parser = argparse.ArgumentParser()

    parser.add_argument("--device", type=int, default=0)

    parser.add_argument("--width", help='cap width', type=int, default=960)

    parser.add_argument("--height", help='cap height', type=int, default=540)

    parser.add_argument('--use_static_image_mode', action='store_true')

    parser.add_argument("--min_detection_confidence",
```

```
                help='min_detection_confidence',
                type=float,
                default=0.7)
    parser.add_argument("--min_tracking_confidence",
                help='min_tracking_confidence',
                type=int,
                default=0.5)
    args = parser.parse_args()
    return args


def main():
    args = get_args()
    cap_device = args.device
    cap_width = args.width
    cap_height = args.height


    use_static_image_mode = args.use_static_image_mode
    min_detection_confidence = args.min_detection_confidence
    min_tracking_confidence = args.min_tracking_confidence
    use_brect = True
    cap = cv.VideoCapture(cap_device)
```

```
    cap.set(cv.CAP_PROP_FRAME_WIDTH, cap_width)
    cap.set(cv.CAP_PROP_FRAME_HEIGHT, cap_height)
    mp_hands = mp.solutions.hands
    hands = mp_hands.Hands(
        static_image_mode=use_static_image_mode,
        max_num_hands=2,
        min_detection_confidence=min_detection_confidence,
        min_tracking_confidence=min_tracking_confidence,
    )
    keypoint_classifier = KeyPointClassifier()
    point_history_classifier = PointHistoryClassifier()
    with open('model/keypoint_classifier/keypoint_classifier_label.csv',
              encoding='utf-8-sig') as f:
        keypoint_classifier_labels = csv.reader(f)
        keypoint_classifier_labels = [
            row[0] for row in keypoint_classifier_labels
        ]
    with open(
            'model/point_history_classifier/point_history_classifier_label.csv',
            encoding='utf-8-sig') as f:
        point_history_classifier_labels = csv.reader(f)
```

```
    point_history_classifier_labels = [
        row[0] for row in point_history_classifier_labels
    ]
cvFpsCalc = CvFpsCalc(buffer_len=10)
history_length = 16
point_history = deque(maxlen=history_length)
finger_gesture_history = deque(maxlen=history_length)
mode = 0
while True:
    fps = cvFpsCalc.get()

    # Process Key (ESC: end)
    key = cv.waitKey(10)
    if key == 27:  # ESC
        break
    number, mode = select_mode(key, mode)

    # Camera capture
    ret, image = cap.read()
    if not ret:
        break
```

```
image = cv.flip(image, 1)  # Mirror display
debug_image = copy.deepcopy(image)

# Detection implementation
image = cv.cvtColor(image, cv.COLOR_BGR2RGB)
image.flags.writeable = False
results = hands.process(image)
image.flags.writeable = True
if results.multi_hand_landmarks is not None:
    for hand_landmarks, handedness in zip(results.multi_hand_landmarks,
                                          results.multi_handedness):
        # Bounding box calculation
        brect = calc_bounding_rect(debug_image, hand_landmarks)
        # Landmark calculation
        landmark_list = calc_landmark_list(debug_image, hand_landmarks)

        # Conversion to relative coordinates / normalized coordinates
        pre_processed_landmark_list = pre_process_landmark(
            landmark_list)
        pre_processed_point_history_list = pre_process_point_history(
            debug_image, point_history)
```

```
        # Write to the dataset file
        logging_csv(number, mode, pre_processed_landmark_list,
                    pre_processed_point_history_list)
    # Hand sign classification
        hand_sign_id = keypoint_classifier(pre_processed_landmark_list)
        if hand_sign_id == 2:  # Point gesture
            point_history.append(landmark_list[8])
        else:
            point_history.append([0, 0])

        # Finger gesture classification
        finger_gesture_id = 0
        point_history_len = len(pre_processed_point_history_list)
        if point_history_len == (history_length * 2):
            finger_gesture_id = point_history_classifier(
                pre_processed_point_history_list)

        # Calculates the gesture IDs in the latest detection
        finger_gesture_history.append(finger_gesture_id)
        most_common_fg_id = Counter(
            finger_gesture_history).most_common()
```

```
            # Drawing part
            debug_image = draw_bounding_rect(use_brect, debug_image, brect)
            debug_image = draw_landmarks(debug_image, landmark_list)
            debug_image = draw_info_text(
                debug_image,
                brect,
                handedness,
                keypoint_classifier_labels[hand_sign_id],
                point_history_classifier_labels[most_common_fg_id[0][0]],
            )
        else:
            point_history.append([0, 0])
        debug_image = draw_point_history(debug_image, point_history)
        debug_image = draw_info(debug_image, fps, mode, number)

        # Screen reflection
#############################################################
        cv.imshow('Hand Gesture Recognition', debug_image)

    cap.release()
    cv.destroyAllWindows()
```

```
def select_mode(key, mode):
    number = -1
    if 48 <= key <= 57:  # 0 ~ 9
        number = key - 48
    if key == 110:  # n
        mode = 0
    if key == 107:  # k
        mode = 1
    if key == 104:  # h
        mode = 2
    return number, mode


def calc_bounding_rect(image, landmarks):
    image_width, image_height = image.shape[1], image.shape[0]
    landmark_array = np.empty((0, 2), int)
    for _, landmark in enumerate(landmarks.landmark):
        landmark_x = min(int(landmark.x * image_width), image_width - 1)
        landmark_y = min(int(landmark.y * image_height), image_height - 1)
        landmark_point = [np.array((landmark_x, landmark_y))]
        landmark_array = np.append(landmark_array, landmark_point, axis=0)
    x, y, w, h = cv.boundingRect(landmark_array)
```

```
    return [x, y, x + w, y + h]


def calc_landmark_list(image, landmarks):
    image_width, image_height = image.shape[1], image.shape[0]
    landmark_point = []

    # Keypoint
    for _, landmark in enumerate(landmarks.landmark):
        landmark_x = min(int(landmark.x * image_width), image_width - 1)
        landmark_y = min(int(landmark.y * image_height), image_height - 1)
        # landmark_z = landmark.z
        landmark_point.append([landmark_x, landmark_y])
    return landmark_point


def pre_process_landmark(landmark_list):
    temp_landmark_list = copy.deepcopy(landmark_list)

    # Convert to relative coordinates
    base_x, base_y = 0, 0
    for index, landmark_point in enumerate(temp_landmark_list):
        if index == 0:
```

```
            base_x, base_y = landmark_point[0], landmark_point[1]
        temp_landmark_list[index][0] = temp_landmark_list[index][0] - base_x
        temp_landmark_list[index][1] = temp_landmark_list[index][1] - base_y

    # Convert to a one-dimensional list
    temp_landmark_list = list(
        itertools.chain.from_iterable(temp_landmark_list))

    # Normalization
    max_value = max(list(map(abs, temp_landmark_list)))
    def normalize_(n):
        return n / max_value
    temp_landmark_list = list(map(normalize_, temp_landmark_list))
    return temp_landmark_list

def pre_process_point_history(image, point_history):
    image_width, image_height = image.shape[1], image.shape[0]
    temp_point_history = copy.deepcopy(point_history)

    # Convert to relative coordinates
    base_x, base_y = 0, 0
```

```
    for index, point in enumerate(temp_point_history):
        if index == 0:
            base_x, base_y = point[0], point[1]
        temp_point_history[index][0] = (temp_point_history[index][0] -
                                        base_x) / image_width
        temp_point_history[index][1] = (temp_point_history[index][1] -
                                        base_y) / image_height

    # Convert to a one-dimensional list
    temp_point_history = list(
        itertools.chain.from_iterable(temp_point_history))
    return temp_point_history


def logging_csv(number, mode, landmark_list, point_history_list):
    if mode == 0:
        pass
    if mode == 1 and (0 <= number <= 9):
        csv_path = 'model/keypoint_classifier/keypoint.csv'
        with open(csv_path, 'a', newline="") as f:
            writer = csv.writer(f)
            writer.writerow([number, *landmark_list])
```

```
    if mode == 2 and (0 <= number <= 9):
        csv_path = 'model/point_history_classifier/point_history.csv'
        with open(csv_path, 'a', newline="") as f:
            writer = csv.writer(f)
            writer.writerow([number, *point_history_list])
    return


def draw_landmarks(image, landmark_point):
    if len(landmark_point) > 0:
        # Thumb
        cv.line(image, tuple(landmark_point[2]), tuple(landmark_point[3]),
                (0, 0, 0), 6)
        cv.line(image, tuple(landmark_point[2]), tuple(landmark_point[3]),
                (255, 255, 255), 2)
        cv.line(image, tuple(landmark_point[3]), tuple(landmark_point[4]),
                (0, 0, 0), 6)
        cv.line(image, tuple(landmark_point[3]), tuple(landmark_point[4]),
                (255, 255, 255), 2)

        # Index finger
        cv.line(image, tuple(landmark_point[5]), tuple(landmark_point[6]),
```

```
        (0, 0, 0), 6)
    cv.line(image, tuple(landmark_point[5]), tuple(landmark_point[6]),
        (255, 255, 255), 2)
    cv.line(image, tuple(landmark_point[6]), tuple(landmark_point[7]),
        (0, 0, 0), 6)
    cv.line(image, tuple(landmark_point[6]), tuple(landmark_point[7]),
        (255, 255, 255), 2)
    cv.line(image, tuple(landmark_point[7]), tuple(landmark_point[8]),
        (0, 0, 0), 6)
    cv.line(image, tuple(landmark_point[7]), tuple(landmark_point[8]),
        (255, 255, 255), 2)

    # Middle finger
    cv.line(image, tuple(landmark_point[9]), tuple(landmark_point[10]),
        (0, 0, 0), 6)
    cv.line(image, tuple(landmark_point[9]), tuple(landmark_point[10]),
        (255, 255, 255), 2)
    cv.line(image, tuple(landmark_point[10]), tuple(landmark_point[11]),
        (0, 0, 0), 6)
    cv.line(image, tuple(landmark_point[10]), tuple(landmark_point[11]),
        (255, 255, 255), 2)
```

```
cv.line(image, tuple(landmark_point[11]), tuple(landmark_point[12]),
        (0, 0, 0), 6)
cv.line(image, tuple(landmark_point[11]), tuple(landmark_point[12]),
        (255, 255, 255), 2)

# Ring finger
cv.line(image, tuple(landmark_point[13]), tuple(landmark_point[14]),
        (0, 0, 0), 6)
cv.line(image, tuple(landmark_point[13]), tuple(landmark_point[14]),
        (255, 255, 255), 2)
cv.line(image, tuple(landmark_point[14]), tuple(landmark_point[15]),
        (0, 0, 0), 6)
cv.line(image, tuple(landmark_point[14]), tuple(landmark_point[15]),
        (255, 255, 255), 2)
cv.line(image, tuple(landmark_point[15]), tuple(landmark_point[16]),
        (0, 0, 0), 6)
cv.line(image, tuple(landmark_point[15]), tuple(landmark_point[16]),
        (255, 255, 255), 2)

# Little finger
cv.line(image, tuple(landmark_point[17]), tuple(landmark_point[18]),
```

```
        (0, 0, 0), 6)
    cv.line(image, tuple(landmark_point[17]), tuple(landmark_point[18]),
        (255, 255, 255), 2)
    cv.line(image, tuple(landmark_point[18]), tuple(landmark_point[19]),
        (0, 0, 0), 6)
    cv.line(image, tuple(landmark_point[18]), tuple(landmark_point[19]),
        (255, 255, 255), 2)
    cv.line(image, tuple(landmark_point[19]), tuple(landmark_point[20]),
        (0, 0, 0), 6)
    cv.line(image, tuple(landmark_point[19]), tuple(landmark_point[20]),
        (255, 255, 255), 2)

    # Palm
    cv.line(image, tuple(landmark_point[0]), tuple(landmark_point[1]),
        (0, 0, 0), 6)
    cv.line(image, tuple(landmark_point[0]), tuple(landmark_point[1]),
        (255, 255, 255), 2)
    cv.line(image, tuple(landmark_point[1]), tuple(landmark_point[2]),
        (0, 0, 0), 6)
    cv.line(image, tuple(landmark_point[1]), tuple(landmark_point[2]),
        (255, 255, 255), 2)
```

```
        cv.line(image, tuple(landmark_point[2]), tuple(landmark_point[5]),
                (0, 0, 0), 6)
        cv.line(image, tuple(landmark_point[2]), tuple(landmark_point[5]),
                (255, 255, 255), 2)
        cv.line(image, tuple(landmark_point[5]), tuple(landmark_point[9]),
                (0, 0, 0), 6)
        cv.line(image, tuple(landmark_point[5]), tuple(landmark_point[9]),
                (255, 255, 255), 2)
        cv.line(image, tuple(landmark_point[9]), tuple(landmark_point[13]),
                (0, 0, 0), 6)
        cv.line(image, tuple(landmark_point[9]), tuple(landmark_point[13]),
                (255, 255, 255), 2)
        cv.line(image, tuple(landmark_point[13]), tuple(landmark_point[17]),
                (0, 0, 0), 6)
        cv.line(image, tuple(landmark_point[13]), tuple(landmark_point[17]),
                (255, 255, 255), 2)
        cv.line(image, tuple(landmark_point[17]), tuple(landmark_point[0]),
                (0, 0, 0), 6)
        cv.line(image, tuple(landmark_point[17]), tuple(landmark_point[0]),
                (255, 255, 255), 2)
    # Key Points
```

```
for index, landmark in enumerate(landmark_point):
    if index == 0:
        cv.circle(image, (landmark[0], landmark[1]), 5, (255, 255, 255),
                  -1)
        cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)
    if index == 1:
        cv.circle(image, (landmark[0], landmark[1]), 5, (255, 255, 255),
                  -1)
        cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)
    if index == 2:
        cv.circle(image, (landmark[0], landmark[1]), 5, (255, 255, 255),
                  -1)
        cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)
    if index == 3:
        cv.circle(image, (landmark[0], landmark[1]), 5, (255, 255, 255),
                  -1)
        cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)
    if index == 4:
        cv.circle(image, (landmark[0], landmark[1]), 8, (255, 255, 255),
                  -1)
        cv.circle(image, (landmark[0], landmark[1]), 8, (0, 0, 0), 1)
```

```
if index == 5:
    cv.circle(image, (landmark[0], landmark[1]), 5, (255, 255, 255),
              -1)
    cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)
if index == 6:
    cv.circle(image, (landmark[0], landmark[1]), 5, (255, 255, 255),
              -1)
    cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)
if index == 7:
    cv.circle(image, (landmark[0], landmark[1]), 5, (255, 255, 255),
              -1)
    cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)
if index == 8:
    cv.circle(image, (landmark[0], landmark[1]), 8, (255, 255, 255),
              -1)
    cv.circle(image, (landmark[0], landmark[1]), 8, (0, 0, 0), 1)
if index == 9:
    cv.circle(image, (landmark[0], landmark[1]), 5, (255, 255, 255),
              -1)
    cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)
if index == 10:
```

```
        cv.circle(image, (landmark[0], landmark[1]), 5, (255, 255, 255),
                  -1)
        cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)
    if index == 11:


        cv.circle(image, (landmark[0], landmark[1]), 5, (255, 255, 255),
                  -1)
        cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)
    if index == 12:
        cv.circle(image, (landmark[0], landmark[1]), 8, (255, 255, 255),
                  -1)
        cv.circle(image, (landmark[0], landmark[1]), 8, (0, 0, 0), 1)
    if index == 13:
        cv.circle(image, (landmark[0], landmark[1]), 5, (255, 255, 255),
                  -1)
        cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)
    if index == 14:
        cv.circle(image, (landmark[0], landmark[1]), 5, (255, 255, 255),
                  -1)
        cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)
    if index == 15:
```

```
            cv.circle(image, (landmark[0], landmark[1]), 5, (255, 255, 255),
                -1)
            cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)
        if index == 16:
            cv.circle(image, (landmark[0], landmark[1]), 8, (255, 255, 255),
                -1)
            cv.circle(image, (landmark[0], landmark[1]), 8, (0, 0, 0), 1)
        if index == 17:
            cv.circle(image, (landmark[0], landmark[1]), 5, (255, 255, 255),
                -1)
            cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)


            cv.circle(image, (landmark[0], landmark[1]), 5, (255, 255, 255),
                -1)
            cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)
        if index == 19:
            cv.circle(image, (landmark[0], landmark[1]), 5, (255, 255, 255),
                -1)
            cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)
        if index == 20:
            cv.circle(image, (landmark[0], landmark[1]), 8, (255, 255, 255),
```

```
                -1)
        cv.circle(image, (landmark[0], landmark[1]), 8, (0, 0, 0), 1)
    return image
def draw_bounding_rect(use_brect, image, brect):
    if use_brect:
        # Outer rectangle
        cv.rectangle(image, (brect[0], brect[1]), (brect[2], brect[3]),
                     (0, 0, 0), 1)
    return image
def draw_info_text(image, brect, handedness, hand_sign_text,
                   finger_gesture_text):
    cv.rectangle(image, (brect[0], brect[1]), (brect[2], brect[1] - 22),
                 (0, 0, 0), -1)

    info_text = handedness.classification[0].label[0:]
    if hand_sign_text != "":
        info_text = info_text + ':' + hand_sign_text
    cv.putText(image, info_text, (brect[0] + 5, brect[1] - 4),
               cv.FONT_HERSHEY_SIMPLEX, 0.6, (255, 255, 255), 1, cv.LINE_AA)

    if finger_gesture_text != "":
```

```
        cv.putText(image, "Finger Gesture:" + finger_gesture_text, (10, 60),
                   cv.FONT_HERSHEY_SIMPLEX, 1.0, (0, 0, 0), 4, cv.LINE_AA)
        cv.putText(image, "Finger Gesture:" + finger_gesture_text, (10, 60),
                   cv.FONT_HERSHEY_SIMPLEX, 1.0, (255, 255, 255), 2,
                   cv.LINE_AA)
    return image
def draw_point_history(image, point_history):
    for index, point in enumerate(point_history):
        if point[0] != 0 and point[1] != 0:
            cv.circle(image, (point[0], point[1]), 1 + int(index / 2),
                      (152, 251, 152), 2)
    return image
def draw_info(image, fps, mode, number):
    cv.putText(image, "FPS:" + str(fps), (10, 30), cv.FONT_HERSHEY_SIMPLEX,
               1.0, (0, 0, 0), 4, cv.LINE_AA)
    cv.putText(image, "FPS:" + str(fps), (10, 30), cv.FONT_HERSHEY_SIMPLEX,
               1.0, (255, 255, 255), 2, cv.LINE_AA)
    mode_string = ['Logging Key Point', 'Logging Point History']
    if 1 <= mode <= 2:
        cv.putText(image, "MODE:" + mode_string[mode - 1], (10, 90),
                   cv.FONT_HERSHEY_SIMPLEX, 0.6, (255, 255, 255), 1,
```

```
                cv.LINE_AA)
        if 0 <= number <= 9:
            cv.putText(image, "NUM:" + str(number), (10, 110),
                       cv.FONT_HERSHEY_SIMPLEX, 0.6, (255, 255, 255), 1,
                       cv.LINE_AA)
    return image
if __name__ == '__main__':
    main()
```

5. SYSTEM DESIGN

Use Case Diagrams:

Use Case during requirement elicitation and analysis to represent the functionality of the system. Use case describes a function by the system that yields a visible result for an actor. The identification of actors and use cases result in the definitions of the boundary of the system i.e., differentiating the task s accomplished by the system and the tasks accomplished by its environment. The actors are on the outside of the system's border, whilst the use cases are on the inside. The behaviour of the system as viewed through the eyes of the actor is described in a use case. It explains the system's role as a series of events that result in a visible consequence for the actor. Use Case Diagrams: What Are They Good For? The objective of a use case diagram is to capture a system's dynamic nature.. However, this definition is too generic to describe the purpose, as other four diagrams (activity, sequence, collaboration, and State chart) also have the same purpose. We will look into some specific purpose, which will distinguish it from other four diagrams.

Data Flow:

The DFD is also known as bubble chart. It is a simple graphical Formalism that can be used to represent a system in terms of the input data to the system, various Processing carried out on these data, and the output data is generated by the system. It maps out the flow of information for any process or system, how data is processed in terms of inputs and outputs. It uses defined symbols like rectangles, circles and arrows to show data inputs, outputs, storage points and the routes between each destination. They can be used to analyse an existing system or model of a new one. A DFD can often visually “say” things that would be hard to explain in words and they work for both technical and non- technical. There are four components in DFD:

- External Entity
- Process
- Data Flow
- data Store

SEQUENCE DIAGRAMS:

Sequence diagram displays the time sequence of the objects participating in the interaction. This consists of the vertical dimension (time) and horizontal dimension (different objects). Objects: An object can be thought of as an entity that exists at a specified time and has a definite value, as

well as a holder of identity. A sequence diagram depicts item interactions in chronological order. It illustrates the scenario's objects and classes, as well as the sequence of messages sent between them in order to carry out the scenario's functionality. In the Logical View of the system under development, sequence diagrams are often related with use case realisations. Event diagrams and event scenarios are other names for sequence diagrams. A sequence diagram depicts multiple processes or things that exist simultaneously as parallel vertical lines (lifelines), and the messages passed between them as horizontal arrows, in the order in which they occur. This enables for the graphical specification of simple runtime scenarios.

Class diagram:

Class diagrams model class structure and contents using design elements such as classes, packages and objects. Class diagram describe the different perspective when designing a system-conceptual, specification and implementation. Classes are composed of three things: name, attributes, and operations. Class diagram also display relationships such as containment, inheritance, association etc. The association relationship is most common relationship in a class diagram. The association shows the relationship between instances of classes

6.1 Use case:

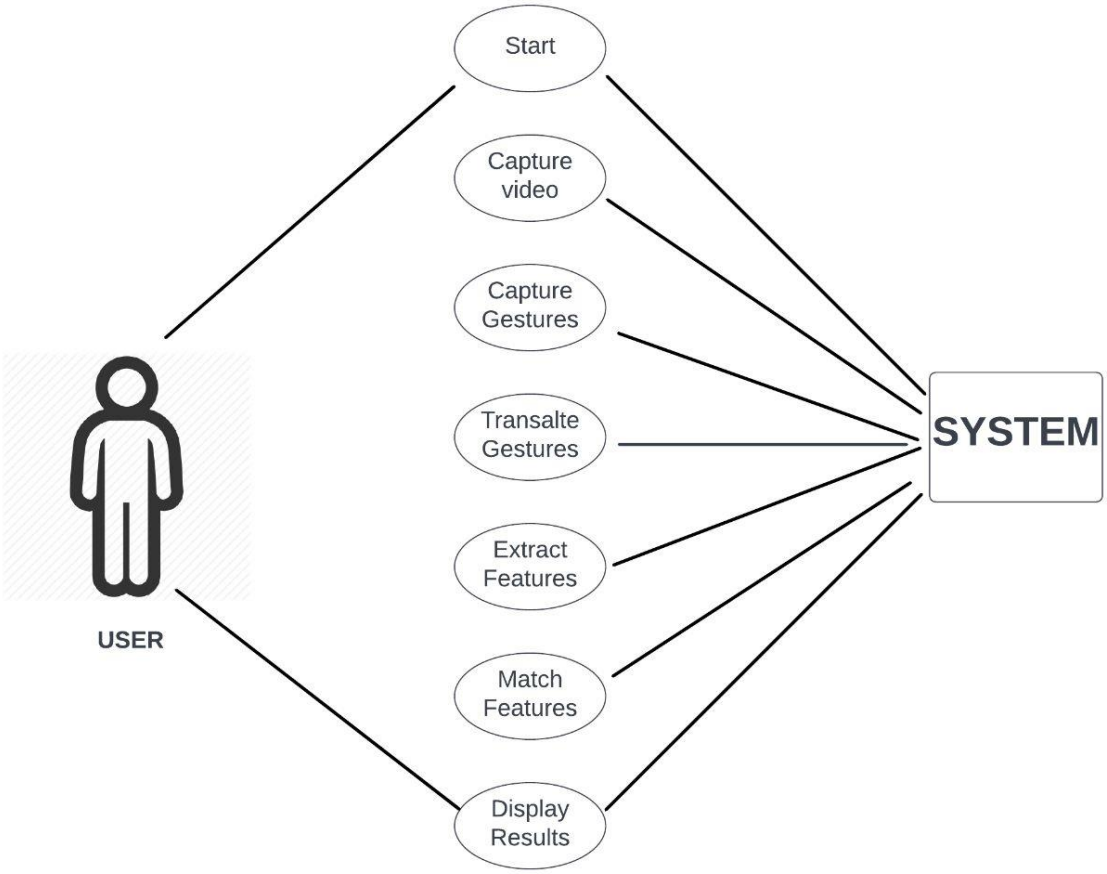

Fig: 3 Use case diagram

6.2 Architectural design:

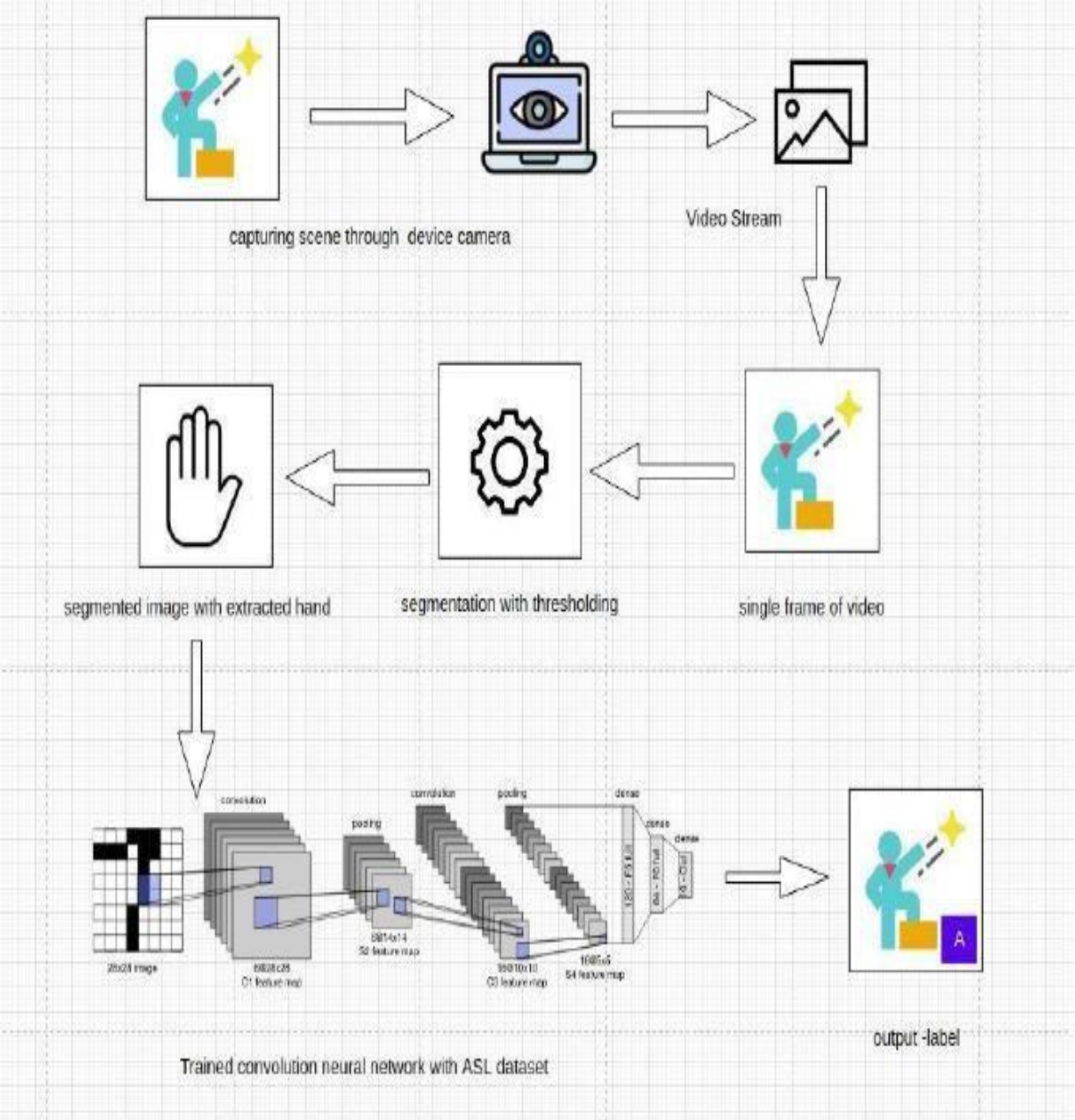

Fig: 4 System architectural design

6.3 Sequence:

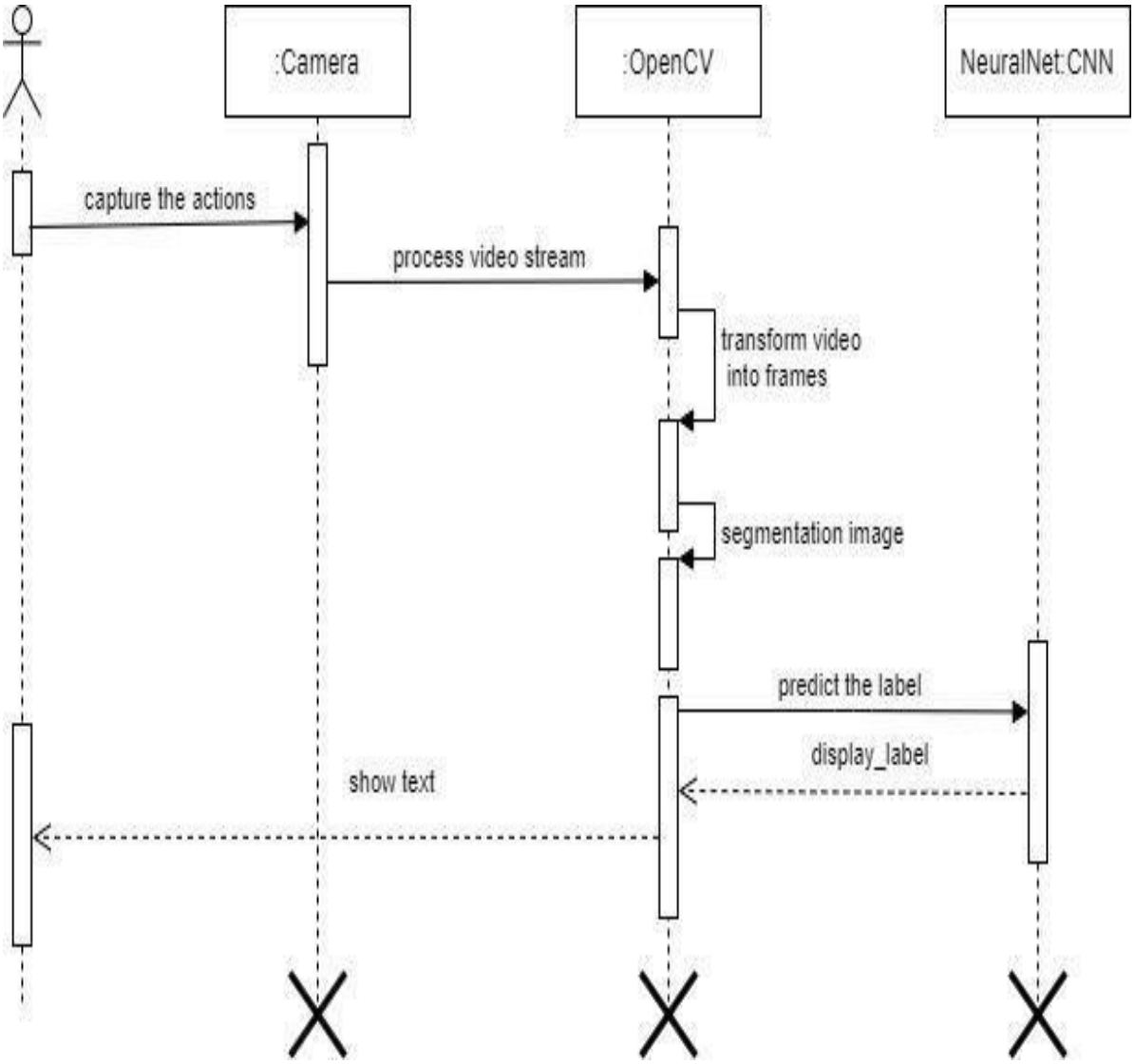

Fig: 5 Sequence diagram

6.4 Data Flow:

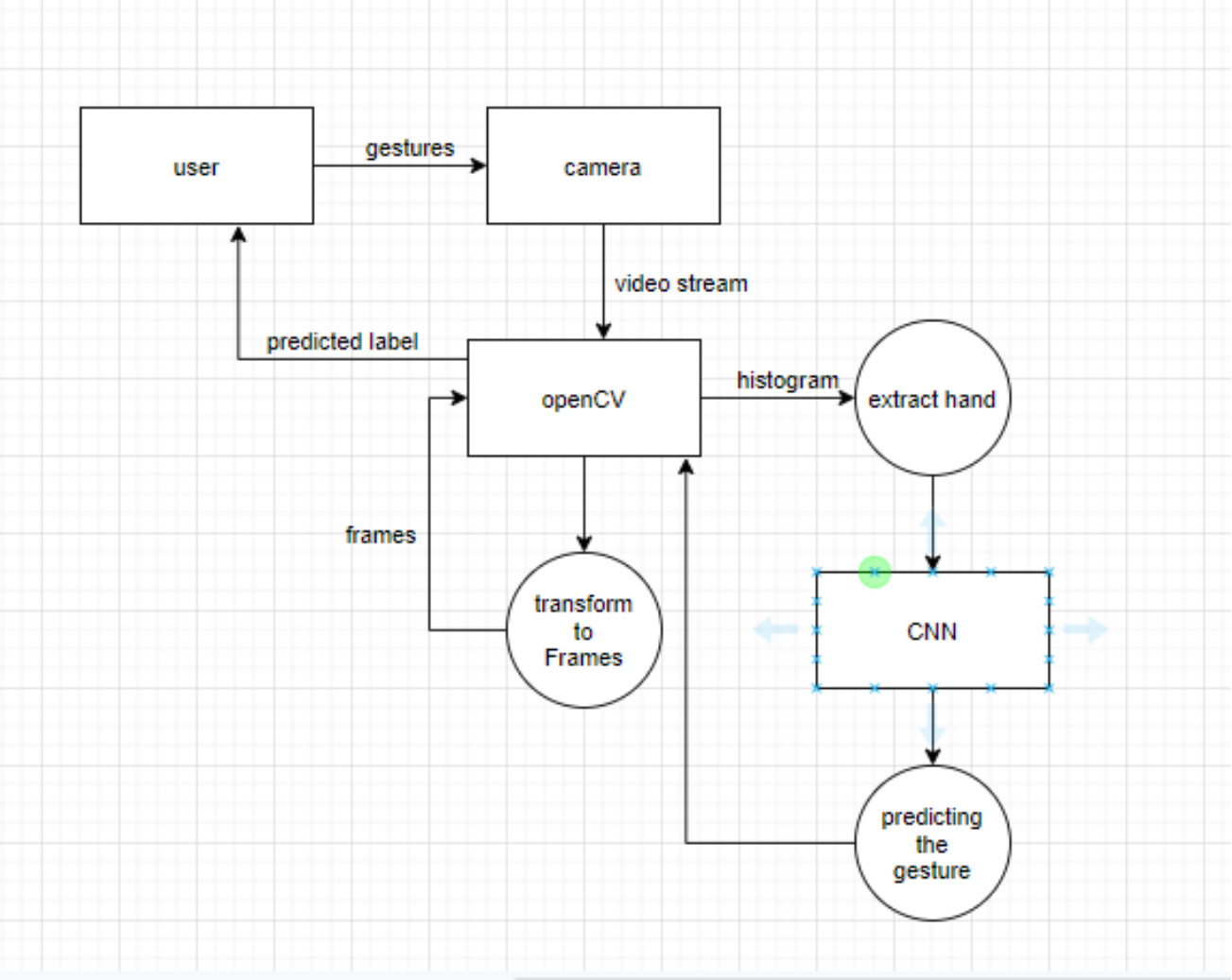

Fig: 6 Data flow diagram

6.5 Activity Diagram:

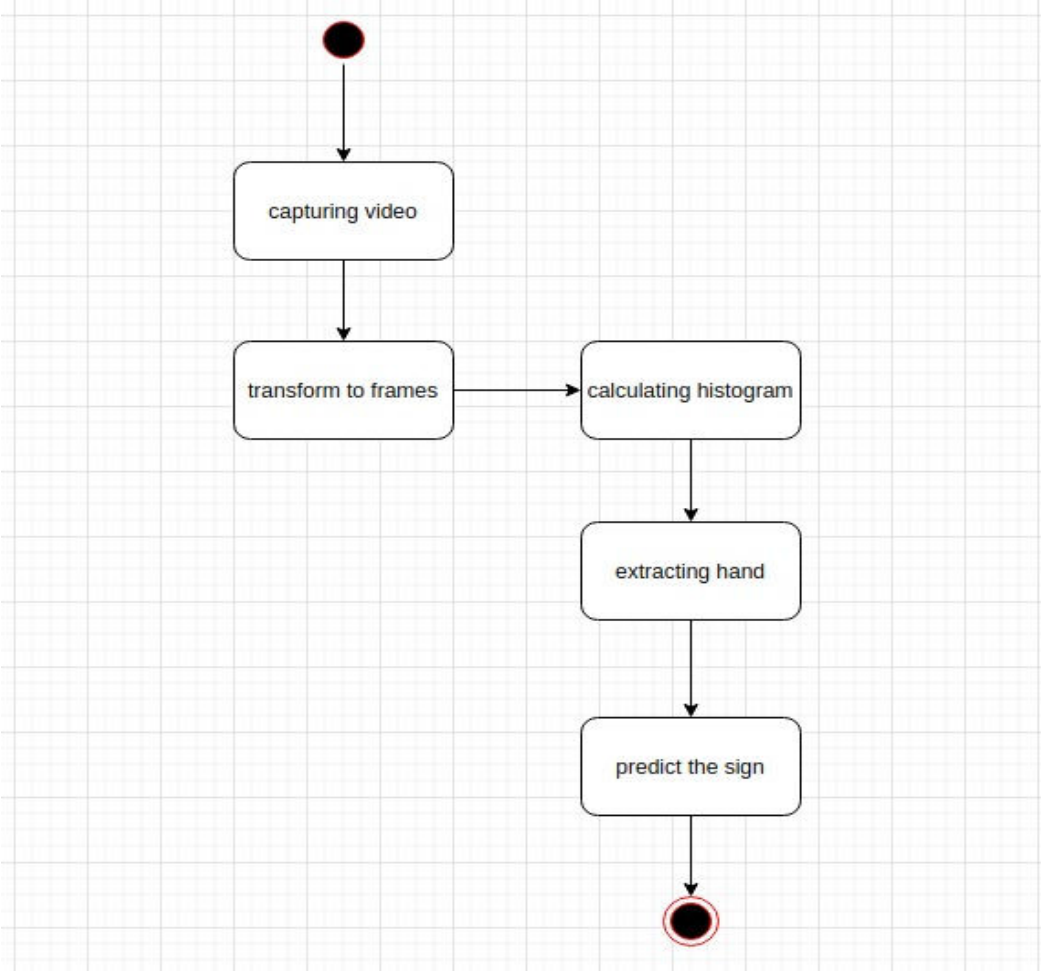

Fig: 7 Activity Diagram

6.6 Class diagram:

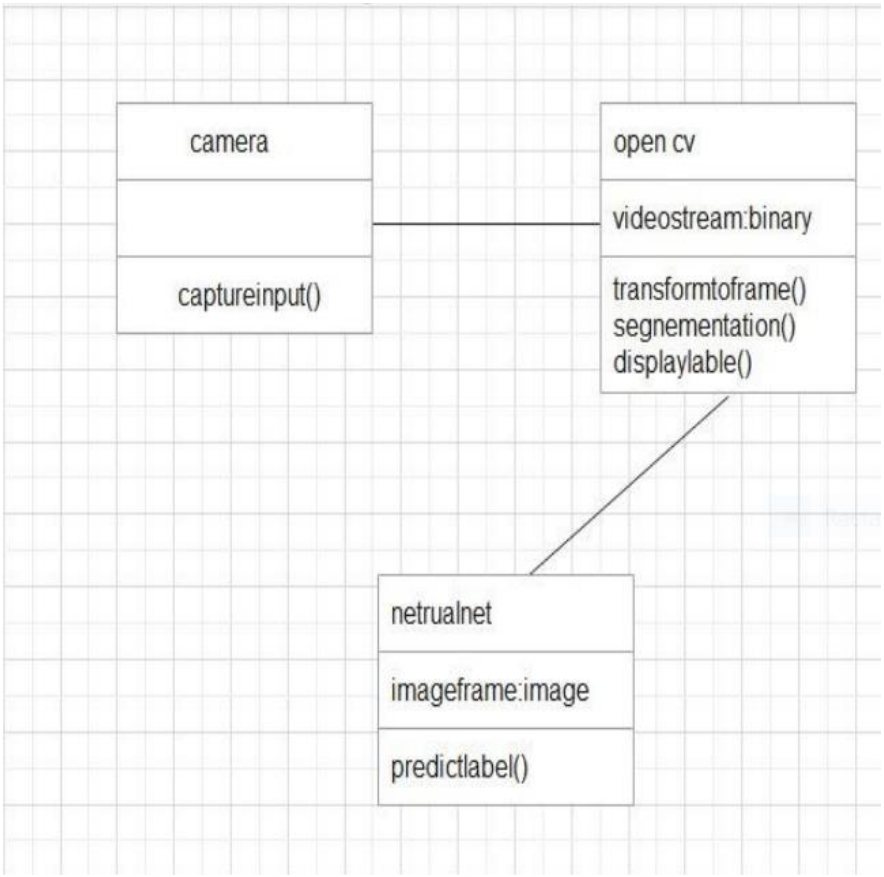

Fig: 8 Class diagram

6. SCREEN SHORTS

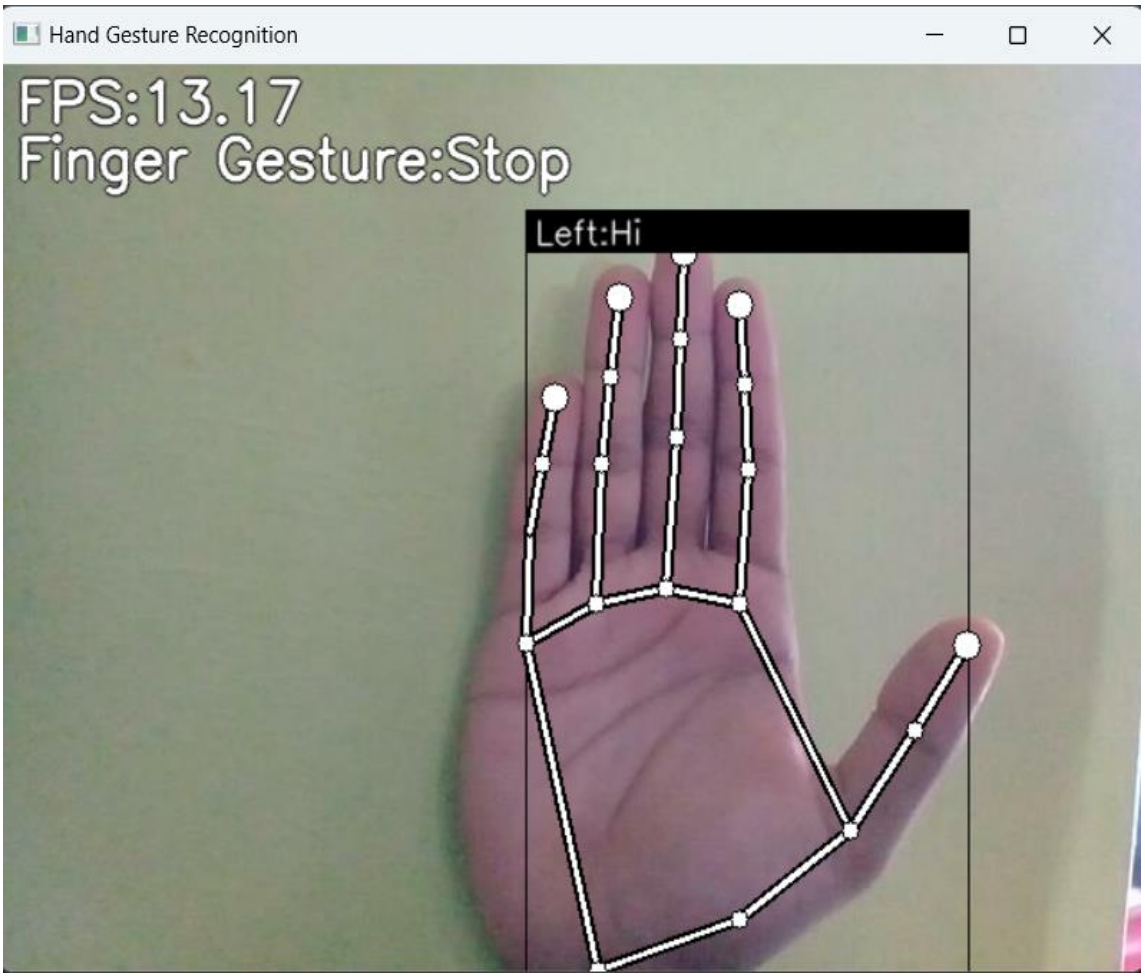

*Fig: 9 Sign Language detection for the word **Hi***

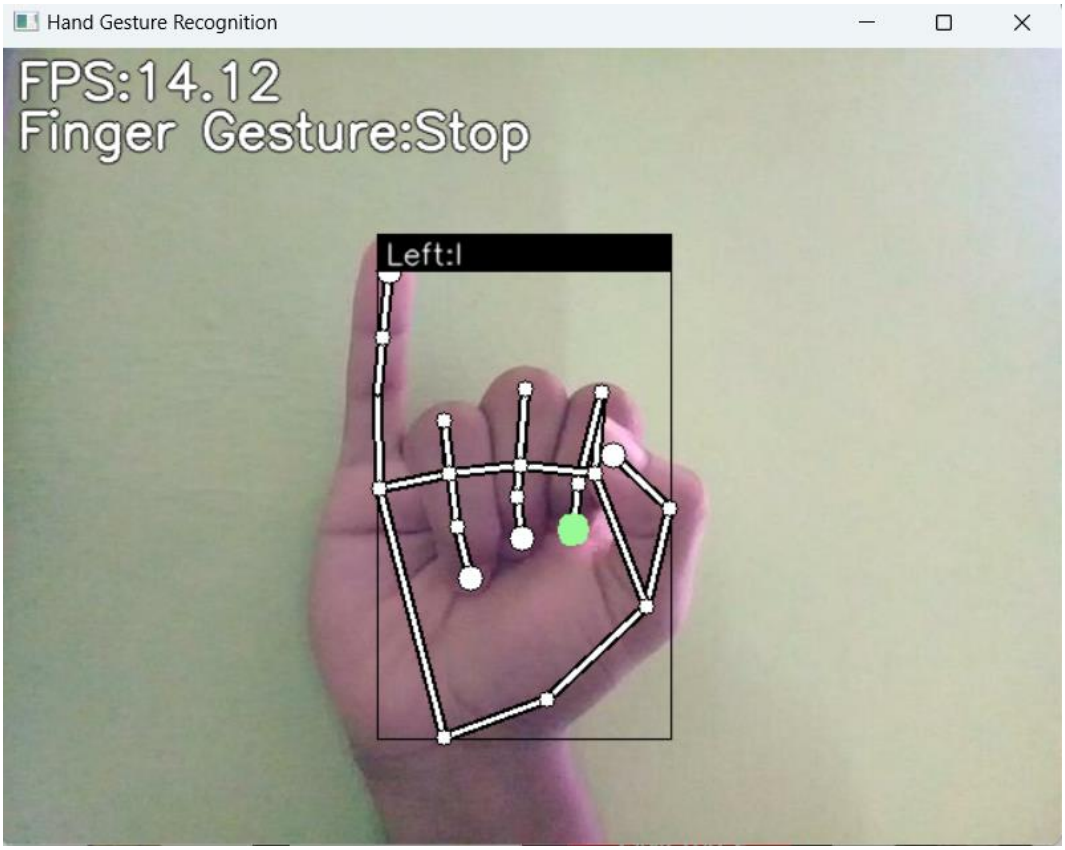

*Fig: 10 Sign Language detection for the word **I***

*Fig: 11 Sign Language detection for the word **Need***

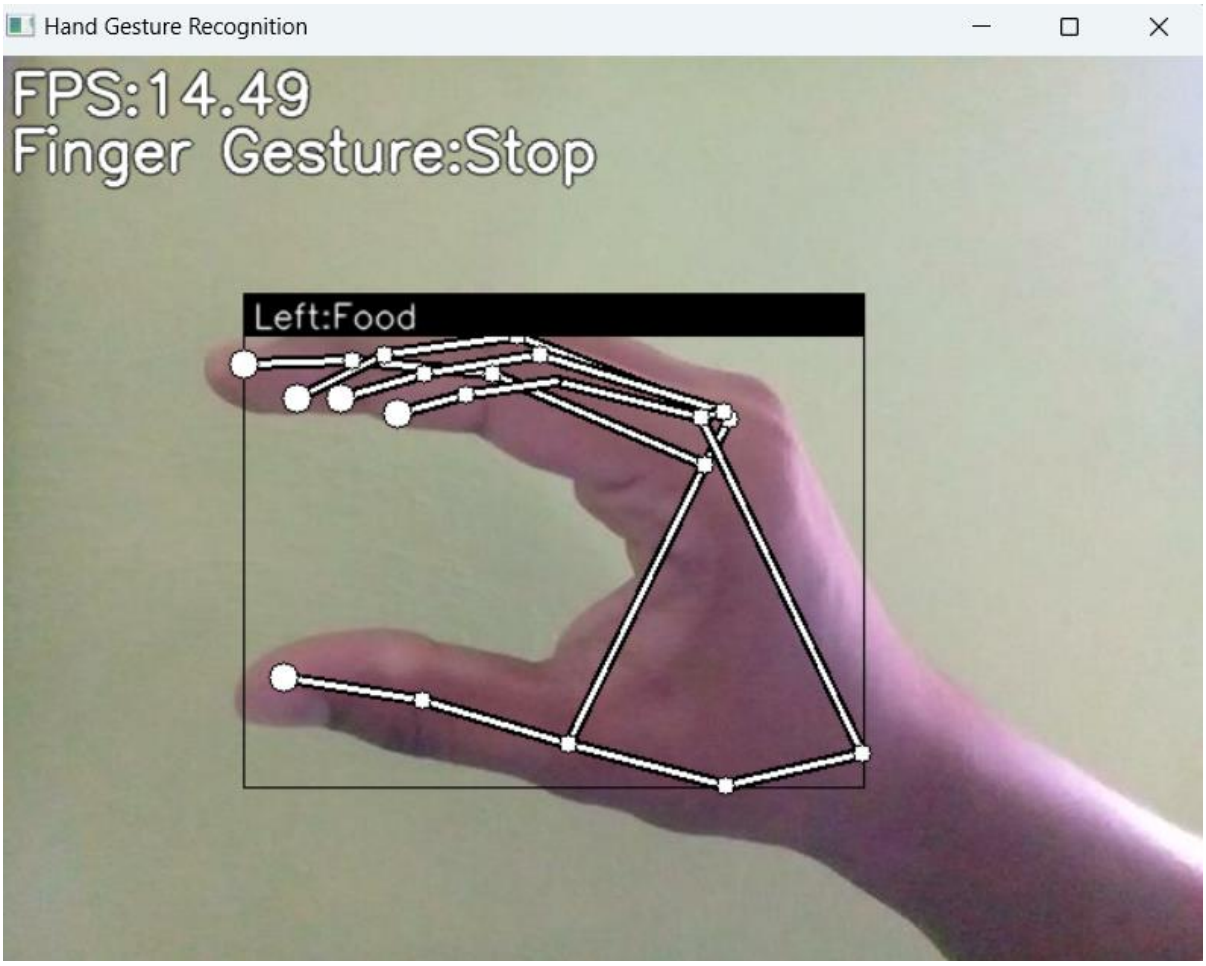

*Fig: 12 Sign Language detection for the word **Food***

7. CONCLUSION

This project is a smart solution for deaf and dumb people to communicate. This project will help people who communicate using sign language to communicate easily and bridge the gap between them and the rest of the world. Hence, we focus on developing an application which will help convert sign language to English text and audio. This project will make sure that the deaf and dumb people are not left out of the rest of the world as this is a very fast passed world.

REFERENCES

1. Kumar Mahesh, “Conversion of Sign Language into Text,” International Journal of Applied Engineering Research ISSN 0973- 4562 Volume 13, Number 9 (2018) pp. 7154-7161.

2. K. Bantupalli and Y. Xie, "American Sign Language Recognition using Deep Learning and Computer Vision," 2018 IEEE International Conference on Big Data (Big Data), Seattle, WA, USA, 2018, pp. 4896-4899, doi: 10.1109/BigData.2018.8622141.

3. Neelam K. Gilorkar, Manisha M. Ingle, “Real Time Detection And Recognition Of Indian And American Sign Language Using Sift”, International Journal of Electronics and Communication Engineering & Technology (IJECET), Volume 5, Issue 5, pp. 11-18 , May 2014

4. Neelam K. Gilorkar, Manisha M. Ingle, “A Review on Feature Extraction for Indian and American Sign Language”, International Journal of Computer Science and Information Technologies, Volume 5 (1) , pp314-318, 2014.

5. Koller, Oscar & Zargaran, Sepehr & Ney, Hermann & Bowden, Richard. (2016). Deep Sign: Hybrid CNN-HMM for Continuous Sign Language Recognition. 10.5244/C.30.136.

6. G. A. Rao, K. Syamala, P. V. V. Kishore and A. S. C. S. Sastry, "Deep convolutional neural networks for sign language recognition," 2018 Conference on Signal Processing And Communication Engineering Systems (SPACES), Vijayawada, 2018, pp. 194-197, doi: 10.1109/SPACES.2018.8316344

7. Rahaman MA, Jasim M, Ali MH, Hasanuzzaman M (2014) Realtime computer vision-

based Bengali Sign Language recognition. In: 17th IEEE international conference on computer and information technology (ICCIT), pp 192–197

8. Nakul Nagpal, Dr.Arun Mitra, Dr.Pankaj Agrawal, Design Issue and Proposed Implementation of Communication Aid for Deaf & Dumb People, International Journal on Recent and Innovation Trends in Computing and Communication ,Volume: 3 Issue: 5,pp- 147 149.

9. Mathavan Suresh Anand, Nagarajan Mohan Kumar, Angappan Kumaresan, An Efficient

Framework for Indian SignLanguage Recognition Using Wavelet Transform Circuits and

Systems, Volume 7, pp 1874- 1883, 2016.

10. T. Bohra, S. Sompura, K. Parekh and P. Raut, "Real-Time Two-Way Communication

System for Speech and Hearing-Impaired Using Computer Vision and Deep Learning,"

2019 International Conference on Smart Systems and Inventive Technology (ICSSIT),

Tirunelveli, India, 2019, pp. 734-739, doi: 10.1109/ICSSIT46314.2019.8987908.

11. Priyankur Sarkar, “What is LDA: Linear Discriminant Analysis for Machine Learning”, September 2019, https://www.knowledgehut.com/blog/data-science/linear-discriminant- analysis-for-machine-learning.