CHAPTER 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.

1.1 Scope of the project

To build a system that can help translate sign language to text format so it is much easier for the dumb and deaf people to communicate with the world as most of the other population cannot understand the sign language. “Sign language Recognition System” recognises the hand gestures from the images and the hands will be segmented from the image and the gestures are classified. It will be providing the text as output for the user.

CHAPTER 2

LITERATURE SURVEY

Neelam K. Gilorkar, Manisha M. Ingle et al. [1] 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. [2]: 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.

Rahaman et al. [3]: 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.

Oscar Kellar et al. [4] introduced a Hybrid CNN-HMM for sign language recognition. They conducted experiments on three datasets namely RWTH-PHOENIX-Weather 2012 [19], RWTHPHOENIX-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.

Mathavan Suresh Anand, Nagarajan Mohan Kumar, Angappan Kumaresan et al [5]: 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.

Mahesh Kumar et al. [6] proposed a system which can recognize 26 hand gestures of Indian sign language based on Linear Discriminant Analysis (LDA). 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. [7] 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.

G. Anantha Rao et al. [8] 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%.

Tanuj Bohra et al. [9]: 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%.

Nakul Nagpal, Dr.Arun Mitra, Dr.Pankaj Agrawal et al. [10]: 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.

CHAPTER 3

SYSTEM ANALYSIS

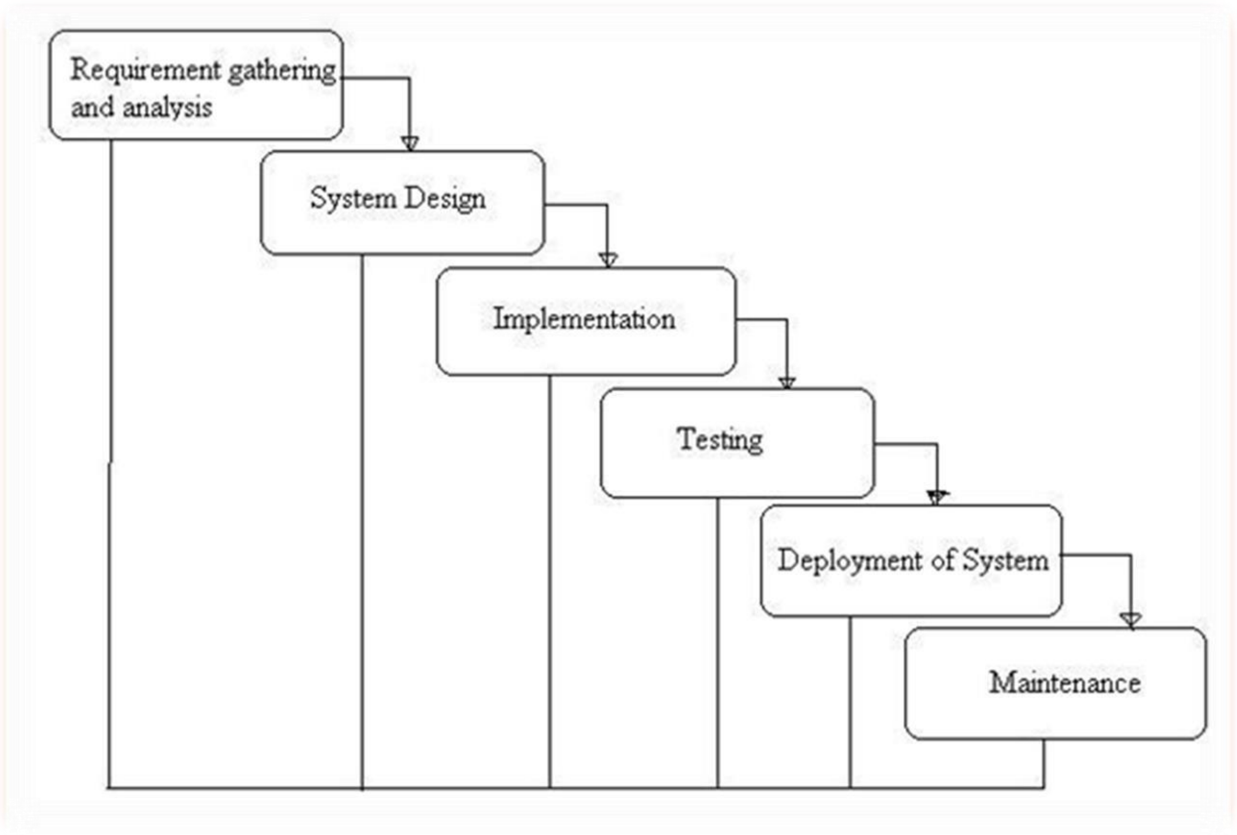

Fig. 3.1. Project SDLC

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

3.1 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.

3.2 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.

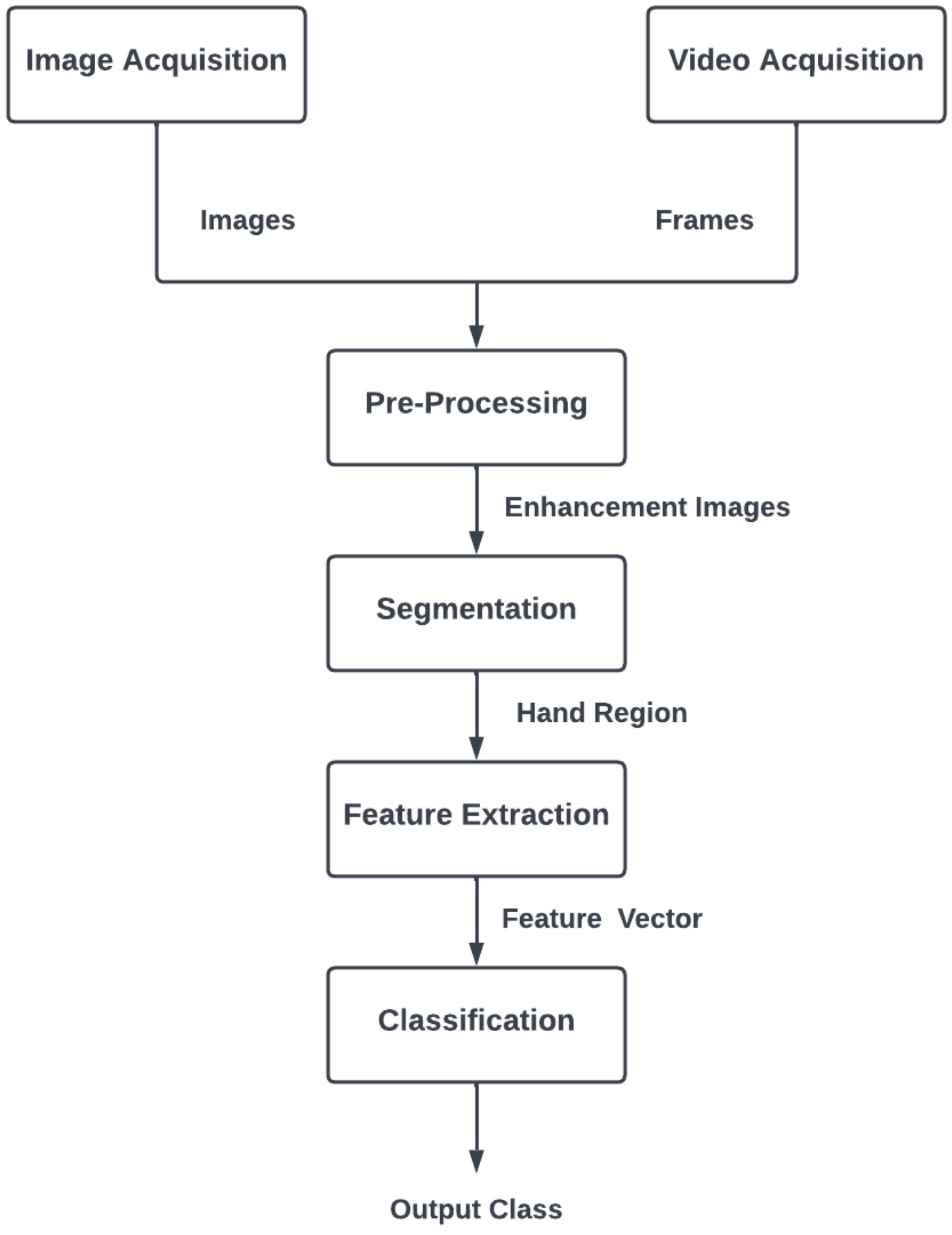

Fig.3.2.1. System Design

3.3 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.

3.4 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.

3.5 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.

3.6 Maintenance

The Maintenance of our Project is one time process.

CHAPTER 4

REQUIREMENTS

4.1 Functional Requirements

- Open Web Camera.
- Detect, segment, and pre-process the hands.
- Recognize the gesture.
- Display the output.

4.2 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.

CHAPTER 5

RELATED WORK

5.1 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.

5.2 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.

5.3 Development Tools

5.3.1 Hardware requirements:

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

5.3.2 Software requirements:

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

CHAPTER 6

IMPLEMENTATION

6.1 System Architecture:

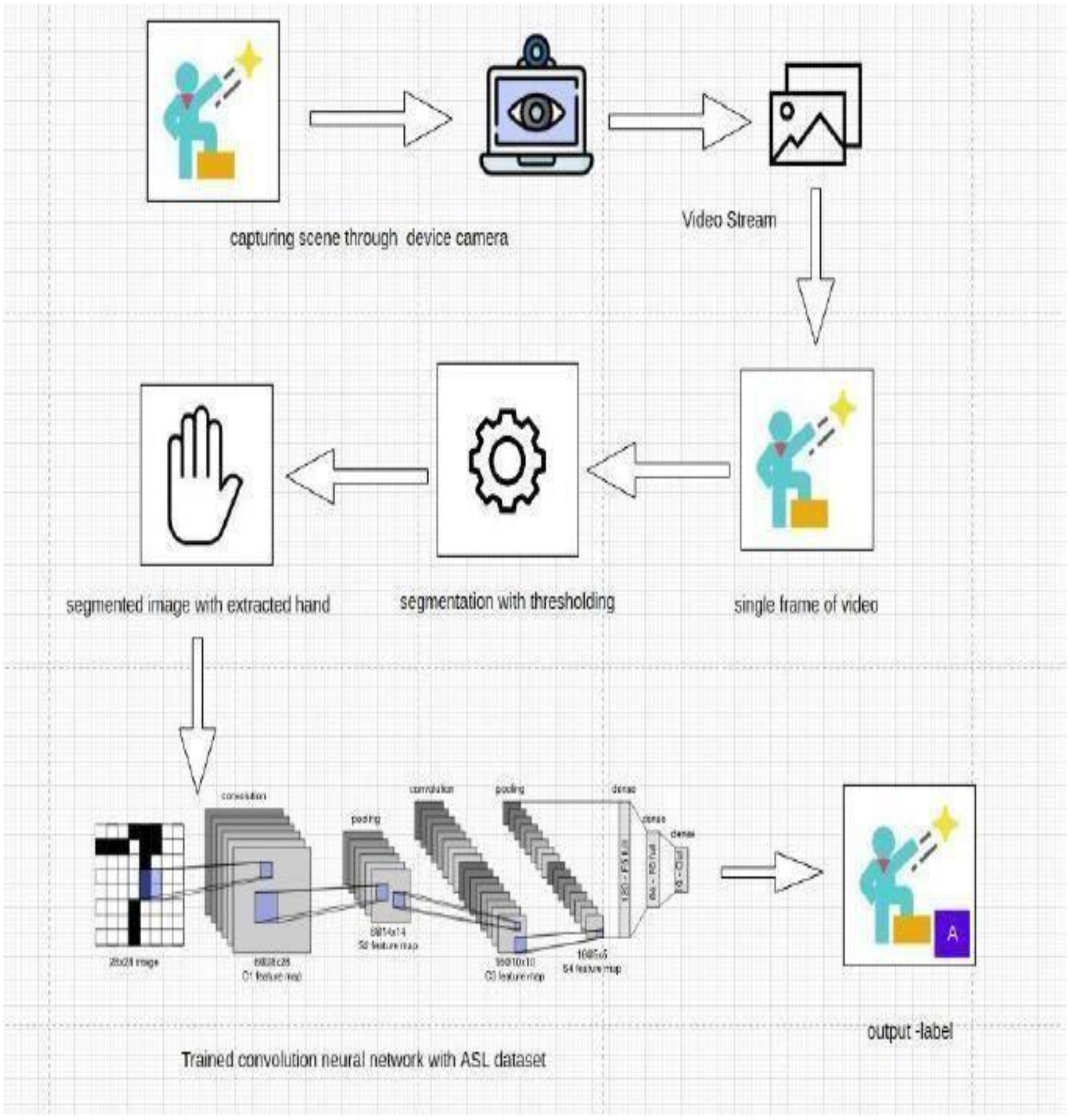
capturing scene through device camera
Video Stream
single frame of video
segmentation with thresholding
segmented image with extracted hand
Trained convolution neural network with ASL dataset
output -label
convolution
pooling
dense
A

Fig.6.1 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 pre-processing like segmentation and thresholding. The hand is then extracted from the pre-processed date. Then the pre-processed data is fed into a pre trained CNN neural network, and the output is displayed.

6.2 Pre-processing:

Pre-processing 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 pre-processing is necessary.

6.3 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 pre-processed. The pre-processed data is then trained by a Convolutional Neural Network.

6.4 Recognize Gesture:

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

6.5 Sample Code:

6.5.1 Main Function:

```
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=1,
    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()
```

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

ret, image = cap.read()
if not ret:
    break
image = cv.flip(image, 1)  # Mirror display
debug_image = copy.deepcopy(image)

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)
        cv.imshow('Hand Gesture Recognition', debug_image)
    cap.release()
    cv.destroyAllWindows()
```

6.5.2 Model Building:

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Input((21 * 2, )),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(20, activation='relu'),
    tf.keras.layers.Dropout(0.4),
    tf.keras.layers.Dense(10, activation='relu'),
    tf.keras.layers.Dense(NUM_CLASSES, activation='softmax')
])
```

6.5.3 Model Training:

```
model.fit(
    X_train,
    y_train,
    epochs=1000,
    batch_size=128,
    validation_data=(X_test, y_test),
    callbacks=[cp_callback, es_callback]
)
```

6.5.4 Draw Landmarks:

```
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)
    if index == 18:
        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
```

CHAPTER 7

SYSTEM DESIGN

7.1 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 investigate some specific purpose, which will distinguish it from other four diagrams.

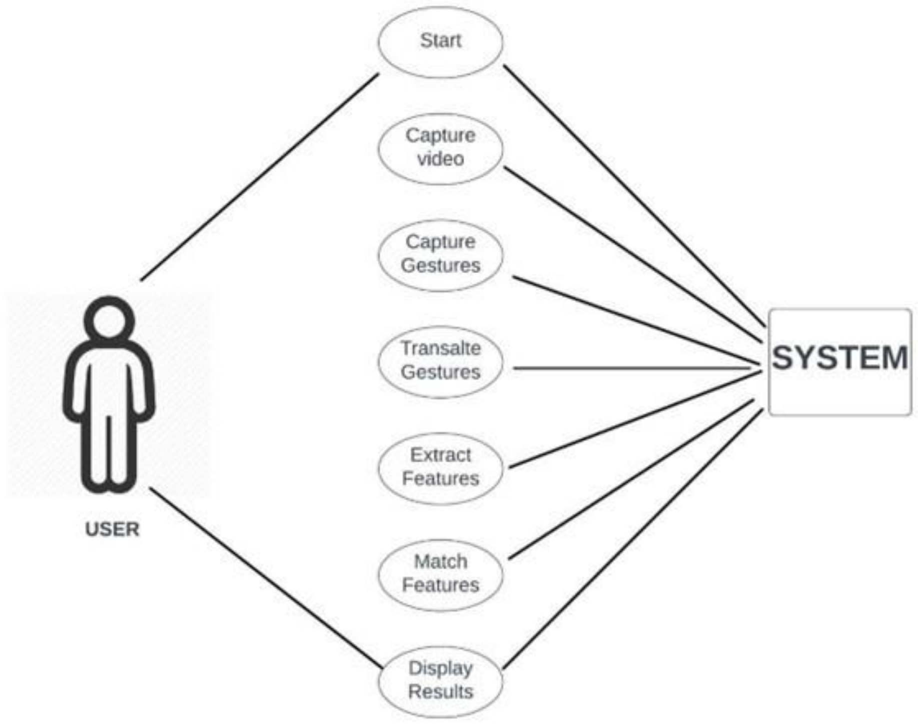

Fig. 7.1. Use Case diagram.

7.2 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

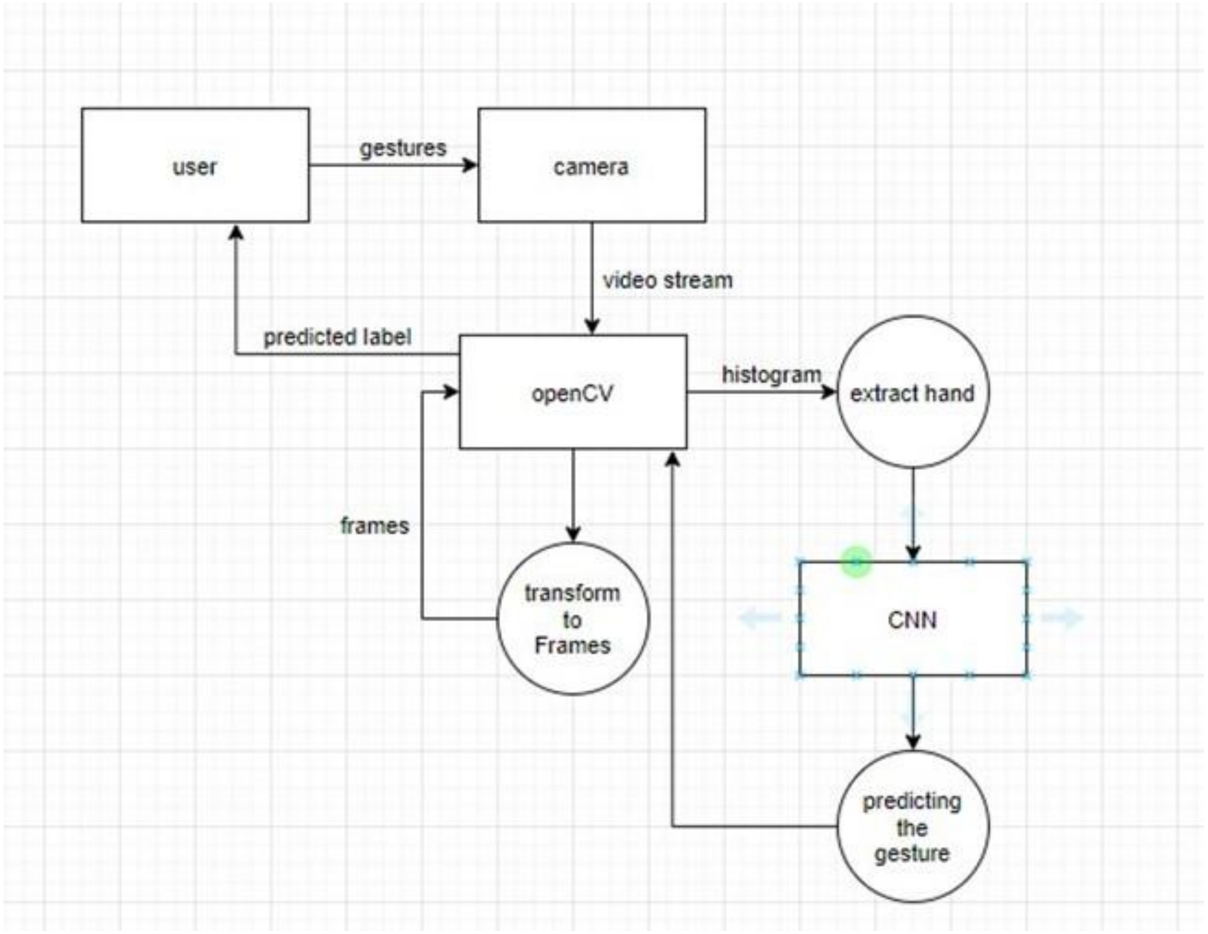

Fig. 7.2. Data Flow Diagram

7.3 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.

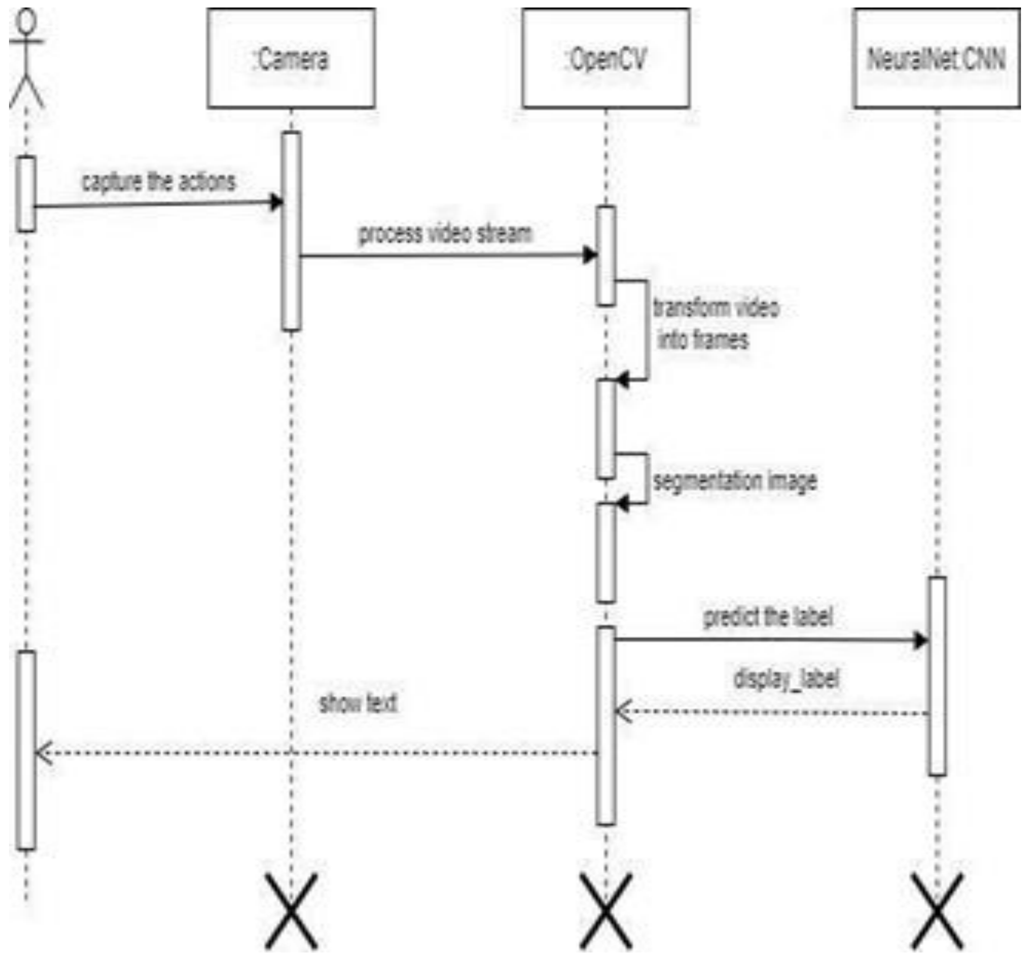

Fig.7.3. Sequence Diagram

7.4 Class diagram:

Class diagrams model class structure and contents using design elements such as classes, packages and objects. Class diagrams 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.

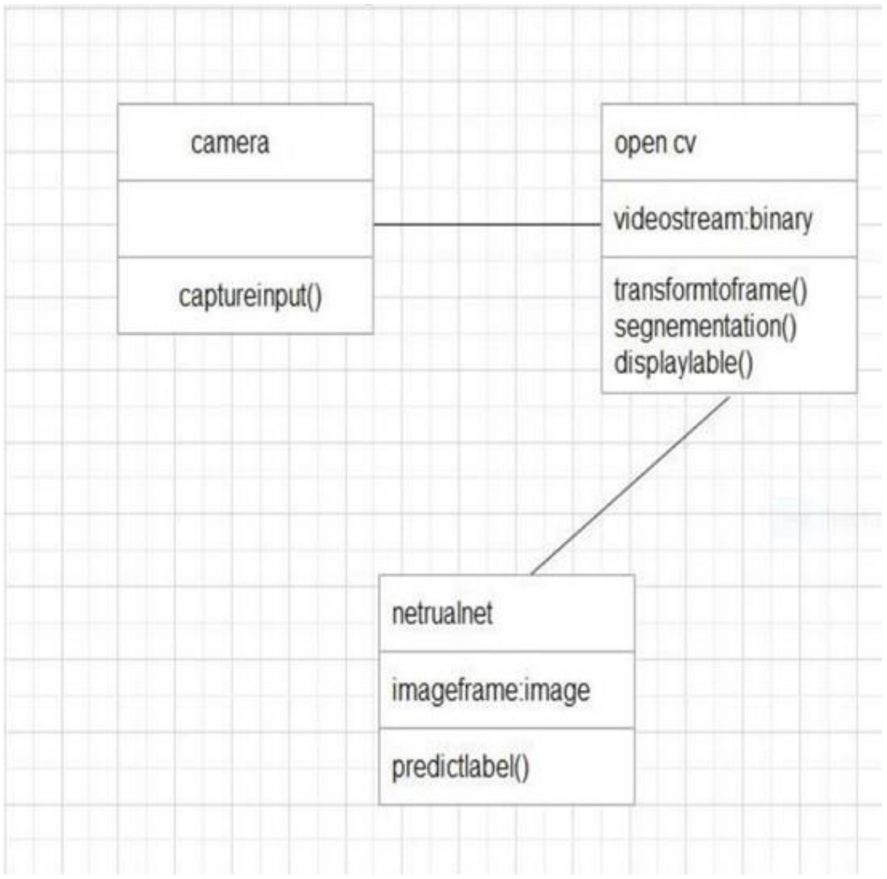

Fig.7.4. Class Diagram

CHAPTER 8

SCREEN SHOTS

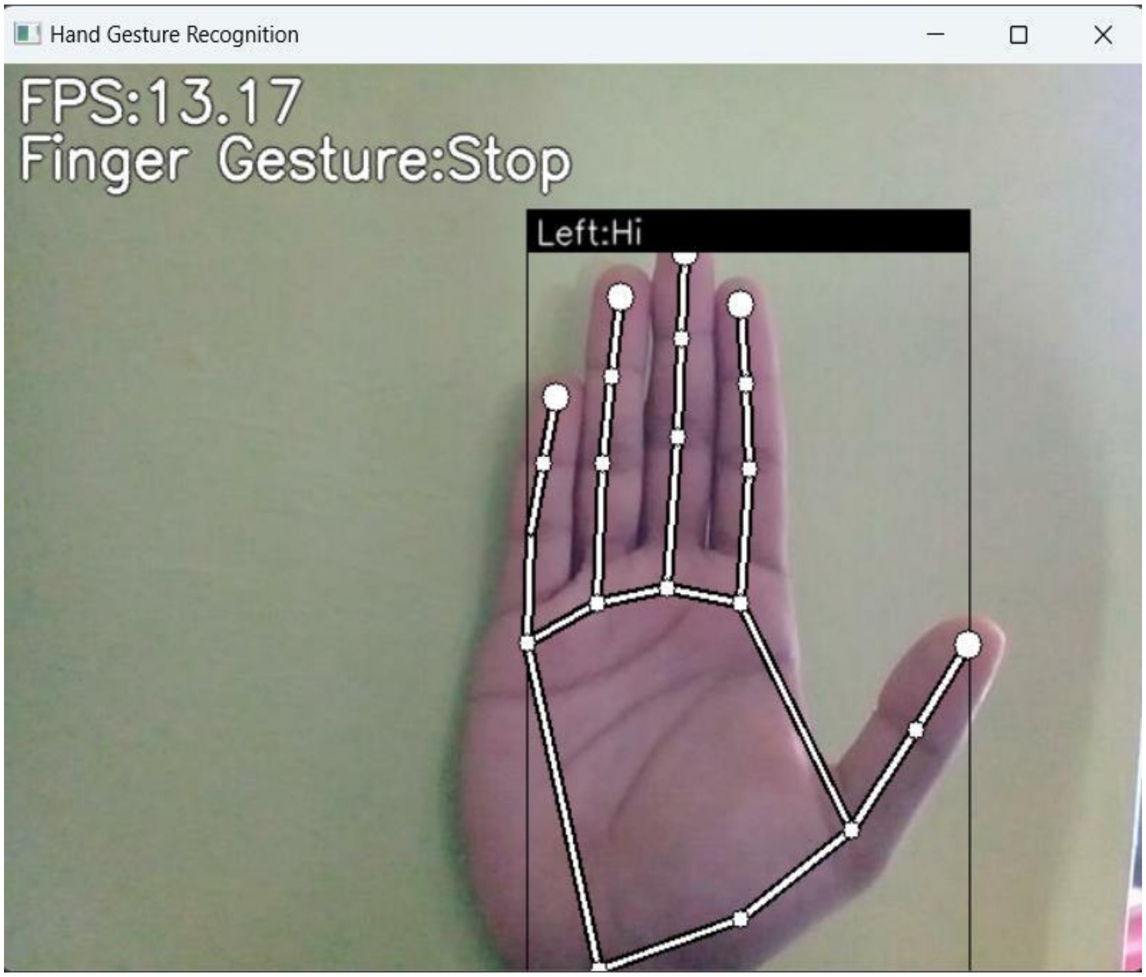

Fig.8.1. Sign Language detection for the word "Hi".

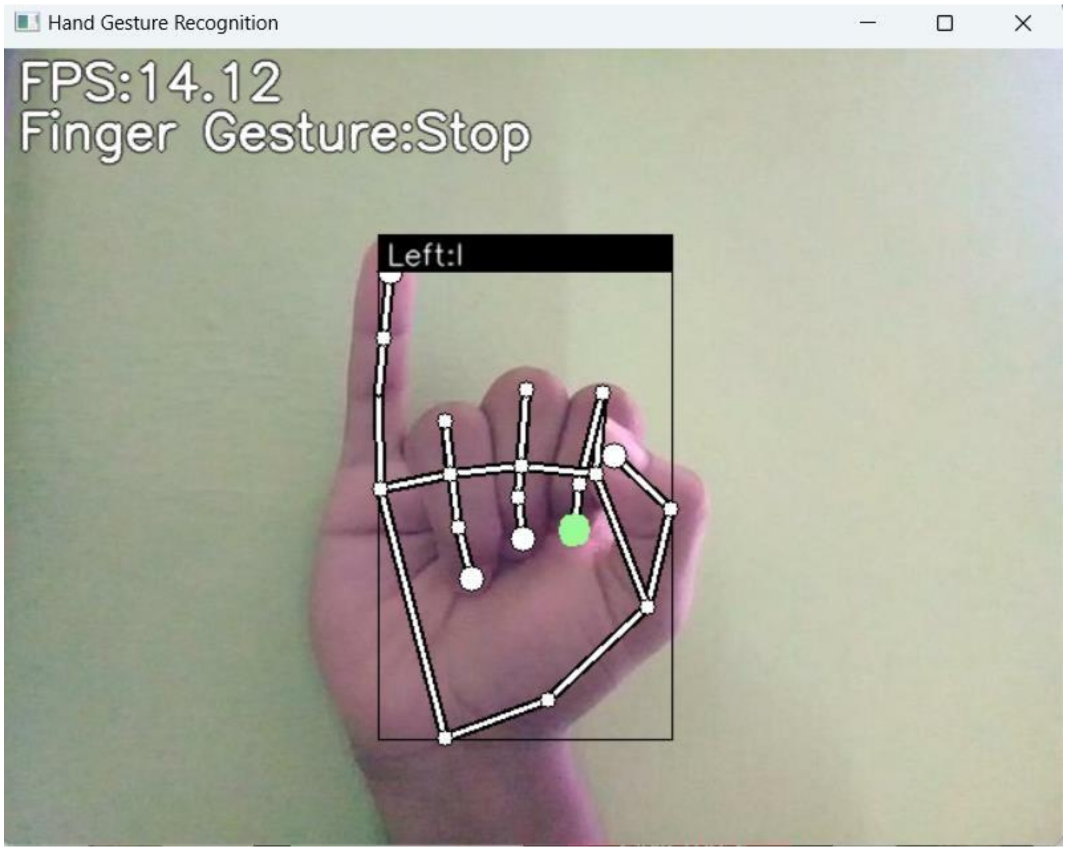

Fig. 8.2. Sign Language detection for the word "I".

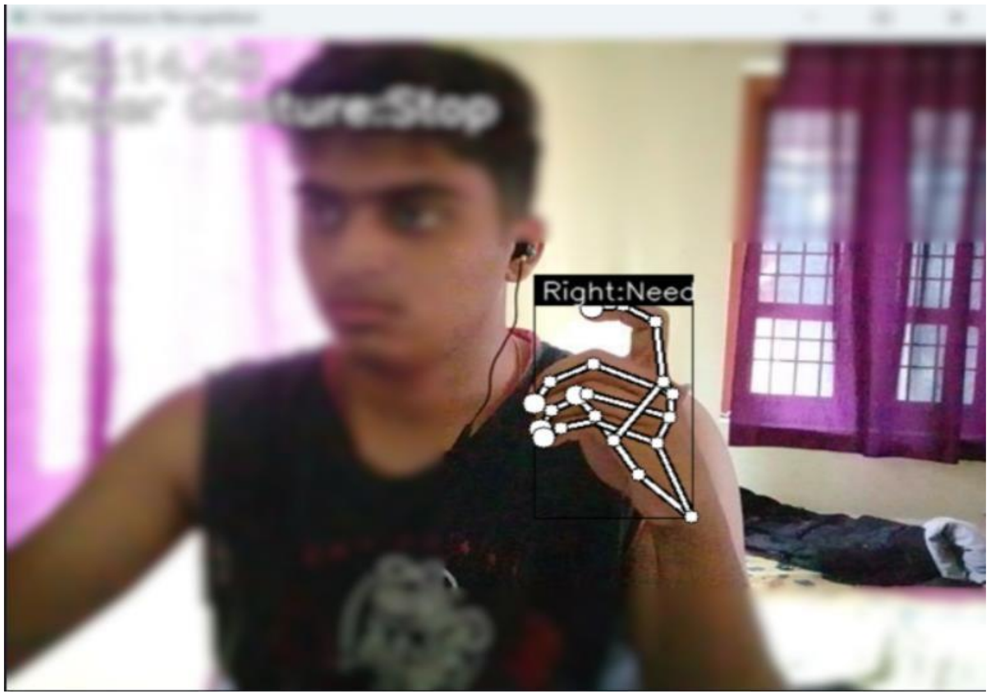

Fig.8.3. Sign Language detection for the word "Need"

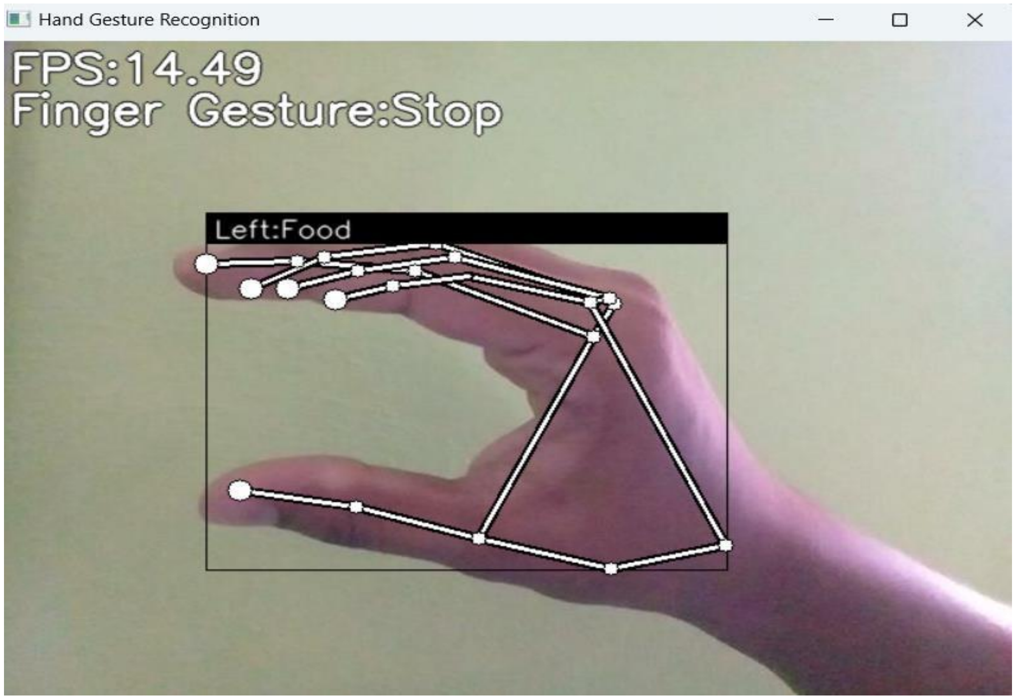

Fig.8.4. Sign Language detection for the word Food

CHAPTER 9

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 a system which will help convert sign language to English text. 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. The project can be future enhanced with the development of application or website for easy access by the people.

REFERENCES

[1] 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.

[2] 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.

[3] 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.

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

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.