# HAND GESTURE ENABLED COMMANDS FOR OPERATING LAPTOPS/PC'S

A Dissertation submitted to the Jawaharlal Nehru Technological University, Hyderabad in partial fulfillment of the requirement for the award of degree of

**BACHELOR OF TECHNOLOGY**

**IN**

**COMPUTER SCIENCE AND ENGINEERING**

Submitted by


| | |
|---|---|
| **PRATHUSHA** | **20B81A05L5** |
| **SOUMYA** | **20B81A05N8** |
| **SREE LIKHITHA** | **20B81A05P2** |

Under the guidance of

**Mr. V. Veera Bhadram**
Sr. Assistant Professor , CSE Dept

Department of Computer Science and Engineering

# CVR COLLEGE OF ENGINEERING

(An UGC Autonomous Institution, Affiliated to JNTUH, Accredited by NBA, and NAAC)

Vastunagar, Mangalpalli (V), Ibrahimpatnam (M),

Ranga Reddy (Dist.) - 501510, Telangana State


**2023-24**

# CVR COLLEGE OF ENGINEERING

***(An Autonomous institution, NAAC Accredited and Affiliated to JNTUH, Hyderabad)***

Vastunagar, Mangalpalli (V), Ibrahimpatnam (M), Rangareddy

(D), Telangana- 501 510

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

## CERTIFICATE

This is to certify that the project report entitled **"Hand Gesture Enabled Commands for Operating Laptops/PC's"** bonafide record of work carried out by **Prathusha (20B81A05L5), Soumya (20B81A05N8)** and **Sree Likhitha(20B81A05P2)** submitted to **Mr. V.Veera Bhadram** for the requirement of the award of **Bachelor of Technology** in **Computer Science** to the CVR College of Engineering, affiliated to Jawaharlal Nehru Technological University, Hyderabad during the year 2022-2023.

**Project Guide**
**Mr. V Veerabhadram**
Sr. Assistant Professor
Department of CSE

**Head of the Department**
**Dr. A.Vani Vathsala**
Professor & HOD-CSE

**Project In-Charge**

**External Examiner**

## DECLARATION

I hereby declare that the project report titled **"Hand Gesture Enabled Commands for Operating Laptops/PC's"** submitted to Computer Science And Engineering Department, CVR College of Engineering, is a record of original work done by me under guidance of **Mr. V .Veerabhadram**. The information and data given in the report is authentic to the best of my knowledge. This project is not submitted to any other university for the award of any degree or published at any time before.

| | |
|---|---|
| **Prathusha** | **20B81A05L5** |
| **Soumya** | **20B81A05N8** |
| **Sree Likhitha** | **20B81A05P2** |

**Date:**

**Place:**

# ACKNOWLEDGEMENT

The success and outcome of this project require a lot of guidance and assistance from many people, and we are extremely privileged to have got this all along with the completion of our project. All that we have done is only due to such supervision and assistance and we should not forget to thank them.

We respect and thank our internal guide, **Mr. V.Veerabhadram**, for providing us an opportunity to do this project work in CVR College of Engineering and giving us all the support and guidance, which made us complete the project duly. We are extremely thankful to him for providing such a nice support and guidance.

We would like to thank the Head of the Department, Professor **Dr. Vani Vathsala** for her meticulous care and cooperation throughout the project work. We thank **Mr.K.Giri Babu**, Project Coordinator for providing us an opportunity to do this project and extending good support and guidance.

We are thankful for and fortunate enough to get constant encouragement, support, and guidance from all **Teaching staff of CSE Department** which helped us in successfully completing our project work. Also, I would like to extend our sincere esteem to all staff in the laboratory for their timely support.

**TABLE OF CONTENTS**

# LIST OF TABLES

## LIST OF FIGURES

## LIST OF ABBREVIATIONS

| | |
|---|---|
| ANN | Artificial Neural Network |
| API | Application Programming Interface |
| ASL | American Sign Language |
| CNN | Convolutional Neural Network |
| CSV | Comma Separated Values |
| CV | Computer Vision |
| GB | Giga Byte |
| HCI | Human Computer Interaction |
| | Human Computer Interface |
| OS | Operating System |
| PC | Personal Computer |
| RAM | Random Access Memory |
| RGB | Red, Green and Blue |
| TUI | Touchless User Interface |
| VGR | Vision-based Gesture Recognition |

# ABSTRACT

This project aims to develop a solution for operating Laptops/PCs using hand gestures. Hand Gesture Recognition is a widely used technique in domains like Computer Vision, Virtual Reality and Augmented Reality. Using gestures to operate electronic devices helps in enhancing the experience of Human-Computer Interaction. Even today, mouse and keyboards are most commonly used Human-Computer Interfaces. Instead, hand gestures can be used to replace these devices for controlling some frequently used operations in Laptops/PCs on daily basis like shut-down, volume increase or decrease, opening basic applications, etc. Different hand gestures can be used as commands for performing various such operations. To recognise these hand-gestures we utilise a camera which is either in-built in the device or a web-camera. It allows us to control the device without having to touch it.

# CHAPTER 1

# INTRODUCTION

## 1.1 Motivation

Computer Vision is a field of AI that enables computers and systems to derive meaningful information from digital images, videos and other visual inputs and take actions based on that information. Gesture Recognition, which is a subdiscipline of computer vision, is a computing process that attempts to recognize and interpret human gestures through the use of mathematical algorithms. Gestures can originate from any bodily motion or state, but commonly originate from the face or hand. This field includes emotion recognition from face and hand gesture recognition, since they are all expressions.

Human-Computer Interaction is the field of study that focuses on optimizing how users and computers interact by designing interactive computer interfaces that satisfy users' needs. The primary objective of HCI is to design systems that make them accessible, usable, efficient, and safe for anyone and everyone. This implies that people with a wide range of capabilities, expertise, and knowledge can easily use HCI-designed systems. Today, keyboard and mouse still play a significant role in human-computer interaction. However, owing to the rapid development of hardware and software, new types of human-computer interaction methods are being developed. Technologies such as speech recognition and gesture recognition receive great attention in this field. In particular, hand gesture recognition is one of the active research areas in the field of human-computer interface due to its flexibility and user friendliness.

Hand gesture is a mode of non-verbal interaction medium and can provide the most intuitive, originative and natural way to interact with computers. On the other hand, natural language interfaces can be difficult to use effectively due to the unpredictable and ambiguous nature of human speech. So, hand gestures can be used as a tool of communication between computer and human by allowing the machine to capture and interpret the user's intent and to respond accordingly.

Hand gesture recognition has great value in many applications such as sign language recognition, augmented reality, virtual reality, robot control, automatic television control, interactive smart home automation, smart video conferencing, interactive presentation module, virtual gaming, vehicle control, medical assistance, robotic surgery, wheelchair control, virtual classroom and so on.

Hand gestures can be static or dynamic hand movements. Static hand gestures rely on the shape of the hand gesture, in which the hand position does not change during the gesturing period, to convey the message. Whereas, dynamic hand gestures rely on the movement of the hands, in which the hand position changes continuously with respect to time, to transfer the meaning. These hand gestures can be recognised using various techniques. Broadly these techniques can be classified into two types:

sensor-based technology and vision-based technology. Sensor-based technology uses different sensors such as the accelerometer and the gyroscope which are either integrated into wearable devices such as gloves or utilized in smart devices such as smartphones, while vision-based technology uses different types of cameras such as RGB cameras and depth cameras to capture and recognise the hand gestures.

Vision-based hand gesture recognition technology is a non-contact type approach which offers development of touchless user interfaces that can amplify the user experience. It lets users give commands to computing devices without touching a screen, mouse device, trackpad, or keyboard. Moreover, touchless or empty-handed gestural input has received considerable attention during the last years because of such benefits as removing the burden of physical contact with an interactive system and making the interaction pleasurable.

We thought it would be great to use vision-based hand gesture recognition technology as a human computer interface for better human-computer interaction experience. Therefore, we have come up with this project which utilises various such computer vision capabilities so that users can now make simple gestures to control or interact with devices efficiently without physically touching them.

### 1.2 Problem Statement

Develop a solution for gesture enabled commands for operating laptops/PCs for frequently used operations on daily basis.

### 1.3 Project Objectives

The main objectives of this project are as follows:

- Users should be able to operate a system through appropriate hand gestures for frequently used operations on daily basis like:
    1. Shutdown the system
    2. Open system's browser
    3. Open system's media player
    4. Increase or Decrease system's volume
- Users should be able to use either the in-built system camera or an externally connected camera to the system, like a web-camera, to capture their hand gestures.
- Users should be able to view results like identified hand gesture name and image.

### 1.4 Project Report Organisation

The report is organised as follows:

- The second chapter discusses about any existing works and their limitations in this domain of the project.
- The third chapter specifies the software and hardware requirements for developing the application.
- The fourth chapter describes the concepts or methods used to build this application and the technology used to implement them. This chapter also includes various UML diagrams to provide a way to visualize the design of the application.
- It is followed by the fifth chapter with implementation details and also discusses in detail about the various steps involved in the project development process.
- The sixth chapter contains screen shots of the application for various test cases and also the results produced when the application is run.
- The final chapter concludes the report by examining the future scope of the application and possible enhancements that can be done in the future.

# CHAPTER 2

# LITERATURE SURVEY

## 2.1 Existing Work

There are existing systems that makes use of computer vision capabilities for human-computer interaction. Some gesture control devices or features that allow a human to interact with a device without any touch or audio are discussed below.

- Kinect is a line of motion sensing input devices produced by Microsoft. The devices typically include RGB cameras, infrared projectors, and detectors that map depth using time of flight or structured light calculations. These components can be utilised, among other things, for real-time gesture identification and body skeleton detection. Additionally, they have microphones that can be utilised for voice control and speech recognition.
- HP's Envy 17 Leap Motion Special Edition laptop comes with gesture recognition system built-in that can sense and react to hand gestures.
- Palm gesture can be used on Samsung mobiles to take a selfie.

Also, there are many published papers around this domain trying to use computer vison capabilities in various applications.

- Authors at [1] have built a hand gesture recognition system for recognising ASL gestures using CNN in real-time.
- Authors at [2] have developed a hand gesture recognition system to be used for human-computer interaction. They have used both static as well as dynamic gestures.

Inspired by such technologies our project aims at utilising the capabilities of computer vision and integrating existing features of gesture control models to create a touchless user interface (TUI) for operating simple and frequently used controls in our PC's.

## 2.2 Limitations of Existing Work

Vision-based recognition of hand gestures, especially dynamic hand gestures, poses challenges for three reasons:

- hand gestures are diverse, have multiple meanings, and vary spatio-temporally;
- the human hand is a complex non-rigid object making it difficult to recognise; and
- computer vision itself is an ill-posed problem.

Other challenges and limitations include:

- The processing of a large amount of image data is time consuming and thus, real-time recognition may be difficult.
- Recognizing objects and patterns that do not have a lot of texture remains difficult.
- Vision-based gesture recognition module is still in its primitive stage due to its limited functionality and gesture vocabulary.
- Unfortunately, like other image processing problems, hand tracking and segmentation in a cluttered background is a critical problem in hand gesture recognition. In most occasions, the background is not simple. Also, the background may contain other body parts captured by the camera creating occlusions.
- Kinect's poor outdoor performance and depth resolution limit its usability.
- Illumination change in the testing environment seriously hampers hand image segmentation.
- VGR systems for indoor applications such as sign-language recognition and virtual reality require uniform ambient illumination. However, VGR systems for outdoor applications such as driver monitoring, vehicular control and so on should work in an environment with a broad range of dynamically changing illuminations and cluttered backgrounds.
- VGR systems in 3D modelling-based applications such as virtual and augmented reality require accurate depth estimation with high depth resolution.
- Background light and multiple reflections create interference and cause ambiguity in time measurement, which results in inaccurate depth measurement.

# CHAPTER 3

# SOFTWARE & HARDWARE SPECIFICATIONS

## 3.1 Software Requirements

Python (v3.10.1) is the programming language used for the complete implementation of this project. The following Python libraries have been used in the application. All these libraries can be installed using the 'pip install' command. These libraries are discussed in detail below:

- OpenCV (v4.7.0.72)

  OpenCV is an open-source computer vision and machine learning software library. The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. In our project, OpenCV module is used to capture hand images and for image processing.

- MediaPipe (v0.9.1.0)

  MediaPipe offers open-source cross-platform, customizable machine learning solutions for live and streaming media. MediaPipe provides solutions for face detection, face mesh, hair segmentation, object tracking, iris, hands, pose detection and tracking, etc. In our project, MediaPipe module is used to detect hands and extract hand features. It is also used to create training data for the model. Python 3.7-3.10 versions support MediaPipe.

- TensorFlow (v2.12.0)

  TensorFlow is a free and open-source software library for machine learning and artificial intelligence. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks. In our project, TensorFlow module is used to build classification ANN model for hand gesture recognition.

- NumPy (v1.23.5)

  NumPy is a python library, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. In our project, NumPy module is used to load and hold training data into arrays to be passed as input for the model.

- CSV

  CSV module implements classes to read and write tabular data in CSV format. Comma Separated Values (CSV) is a simple file format used to store tabular data, such as a spreadsheet or database in plain text. Each line of the file is a data record. Consisting of one or more fields, separated by commas. In our project, CSV module is used to for storing training data in CSV format.

- OS

  OS module in Python provides functions for interacting with the operating system. It comes under Python's standard utility modules. This module provides a portable way of using operating system-dependent functionality. In our project, OS module is used to run system commands to perform basic system operations.

SetVol (v3.4) is a free open-source command line utility which lets us set the volume and recording levels of Windows computer's audio and recording devices [3]. In our project, SetVol utility is used for performing volume increase and decrease operations. To use this utility, its application path has to be set in the 'System Environment Variables' after successful installation.

### 3.2 Hardware Requirements

This project has been developed on a machine having the following specifications:

- Intel CORE i5 processor
- 8 GB RAM
- Windows 11 OS

The developed application requires a camera to capture hand gestures. So, the basic hardware requirement of this project is a system with an in-built camera or a web-camera that can be connected to a system externally.

# CHAPTER 4

# PROPOSED SYSTEM DESIGN

## 4.1 Proposed Methods

The proposed application design can be broadly divided into three modules:

- Camera Module

  In our project, vision-based hand gesture recognition technique is being used wherein a camera is used to capture users hand gestures in the form of images and then these images are further processed for gesture recognition. The application can either use an in-built system camera or use a web-camera to capture gestures, depending on the user requirements. For turning the camera on and capturing the hand gestures in the form of image frames, Python's OpenCV module is used in the application.

- Hand Gesture Recognition Module

  The captured images through camera are then processed for gesture recognition. Vision-based hand gesture recognition largely takes place in three stages: hand detection, feature extraction and gesture recognition.

  Hand detection and features extraction steps are carried out with the help of MediaPipe Solutions Hand module. The last step, gesture recognition, is made by passing the features extracted from an image as input through a trained ANN for classification. Python's TensorFlow library is used to build and train the ANN.

  In our project, we have used only static gestures as commands to control the system. These gestures are: thumb, closed palm, one, two and five finger formations. A dataset of images of all these 5 gestures is created and features from these images are extracted using Mediapipe. The features extracted are labelled and stored in CSV format, which is used as training data for training the ANN model created.

- Action Module

  After the neural network classifies the image as a particular hand gesture, the corresponding system operation is executed. To control the system, we have used Python's OS module to run system's shell commands.

  The below table maps the corresponding system operation that is executed when user provides a particular gesture:

| Hand Gesture | System Operation |
|---|---|
| Closed palm | Shutdown |
| Five | Open web browser |
| Two | Open media player |
| One | Increase volume |
| Thumb | Decrease volume |

**Table 4.1.1:** Hand Gesture Enabled Commands

### 4.2 Technology Description

For hand gesture recognition, the proposed solution is to use MediaPipe's Hand Landmark Model [4]. The hand landmark model bundle detects the keypoint localization of 21 hand-knuckle coordinates within the detected hand regions. The model was trained on approximately 30000 real-world images, as well as several rendered synthetic hand models imposed over various backgrounds. Using this model, we can detect the landmarks of the hands in an image. We can also use it to localize key points of the hands and render visual effects over the hands. This model operates on image data with a machine learning model as static data or a continuous stream and outputs hand landmarks in image coordinates.

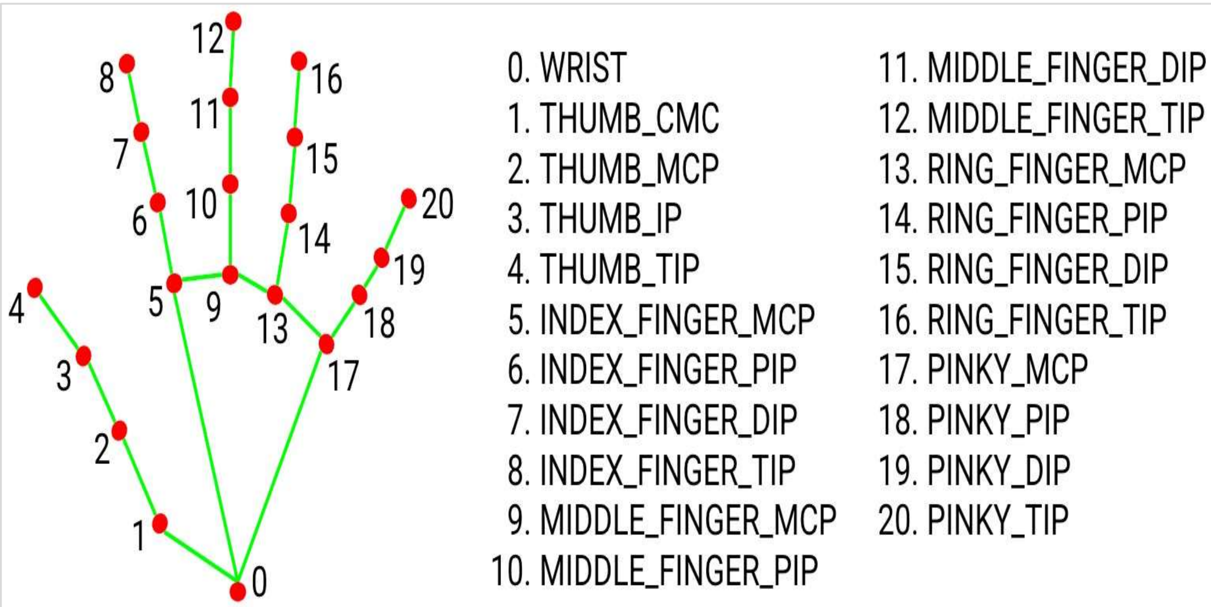


**Fig. 4.2.1:** MediaPipe's Hand Landmark Model

### 4.3 System Architecture

Figure 4.3.1 shows the architectural design of the proposed 'Hand Gesture Enable Commands for Operating Laptops/PC's' application. As discussed in previous 'proposed methods' section, the application is divided into three modules. It clearly depicts the flow of steps involved in the process of controlling a laptop/PC using hand gestures through this application.

- User provides hand gestures to operate the system.
- Camera captures these gestures in the form of images.
- Images captured are passed through MediaPipe.
- MediaPipe detects the hand in the images and extracts hand features from the images.
- The extracted hand features, i.e. hand landmarks, are passed as input to the trained neural network model.
- The trained model performs classification by observing patterns in the input data and predicts the gesture in the image captured.
- Based on the gesture recognised or identified, the corresponding system operation is executed through shell commands.

The 'gesture dataset' shown in this figure is the dataset collected and prepared to train the neural network.

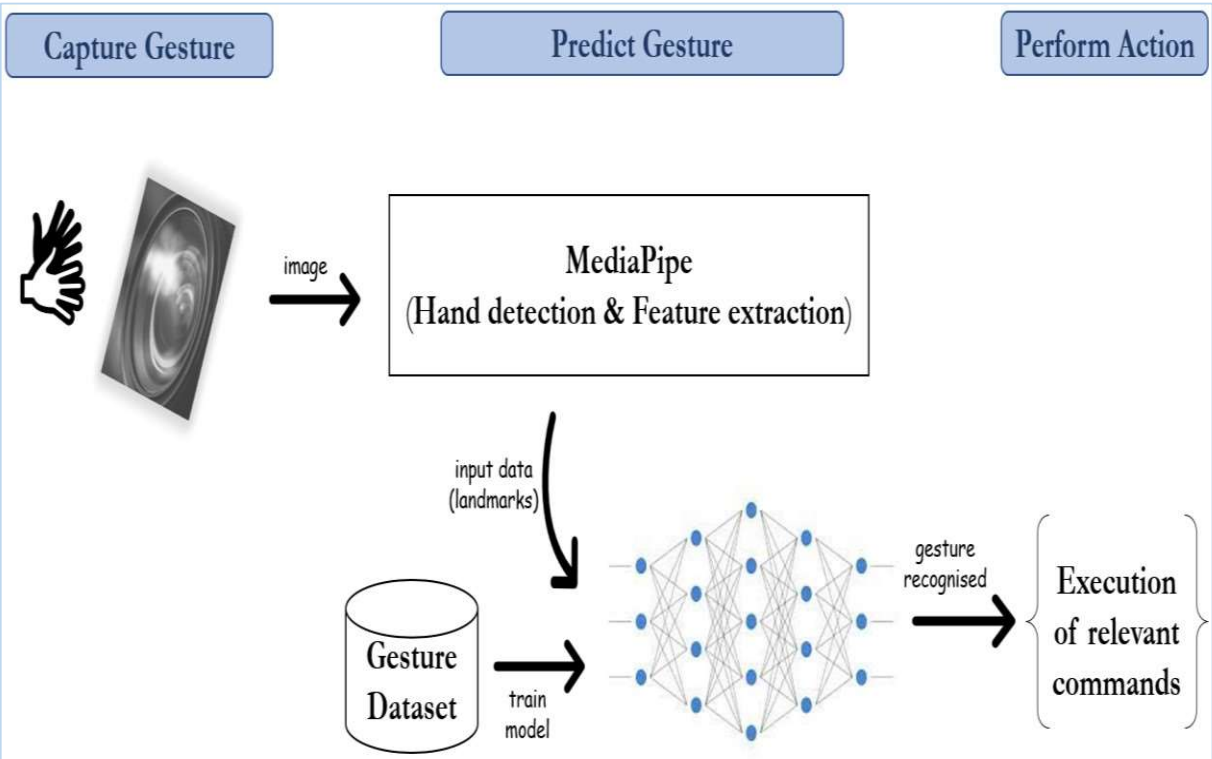


**Fig. 4.3.1:** Proposed System Architecture

### 4.4 Use Case Diagram

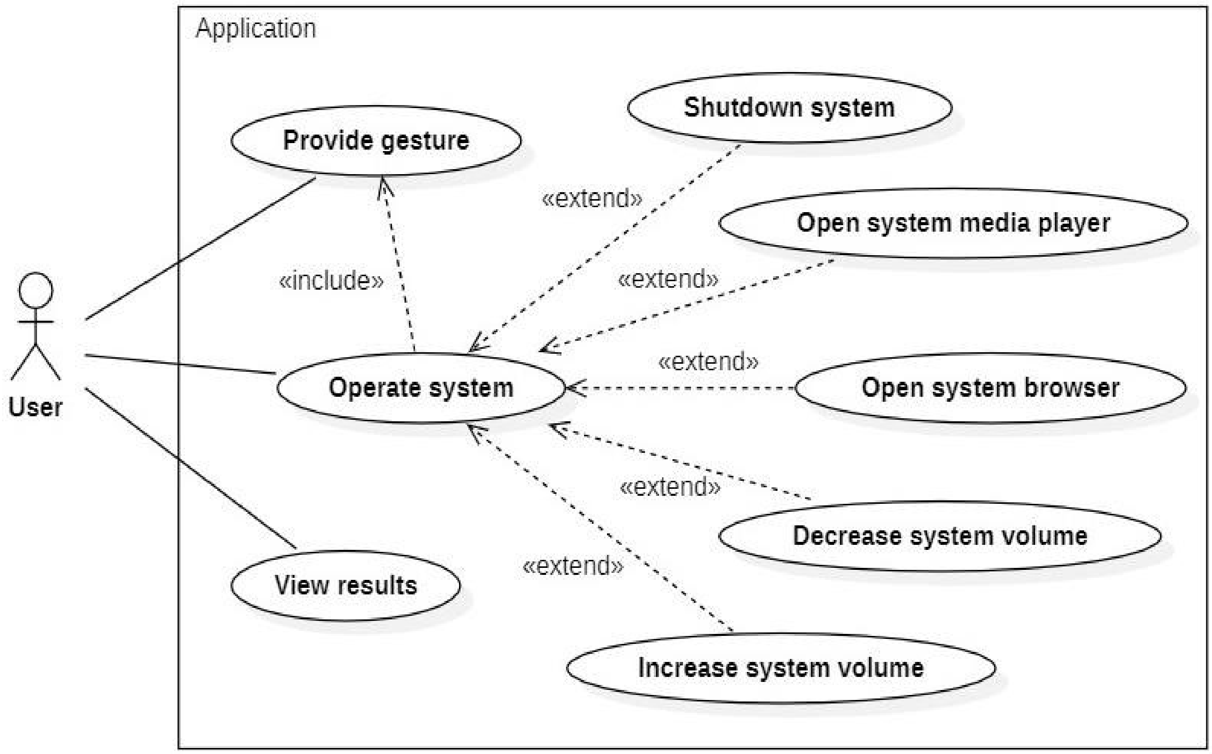


**Fig. 4.4.1:** Use Case Diagram

From the above use case diagram, it is clear that there is only one actor interacting with this application. That actor is identified as the user of the application.

The diagram clearly lists the functional requirements of this application.

- The user should be able to operate the system through this application, but for doing that the user should provide the appropriate hand gesture.
- The system operations that the user can perform through hand gestures include the following:
    - Shutdown the laptop/PC
    - Open the system's web browser
    - Open the system's media player
    - Increase the system's volume
    - Decrease the system's volume

The user should also be able to view the results of hand gesture recognition. This is achieved by displaying the captured image with hand gesture to the user labelled with the corresponding gesture name. Also, through our application the user can see the landmarks (hand gesture features) marked on the hand in the image that are identified by MediaPipe.

## 4.5 Class Diagram

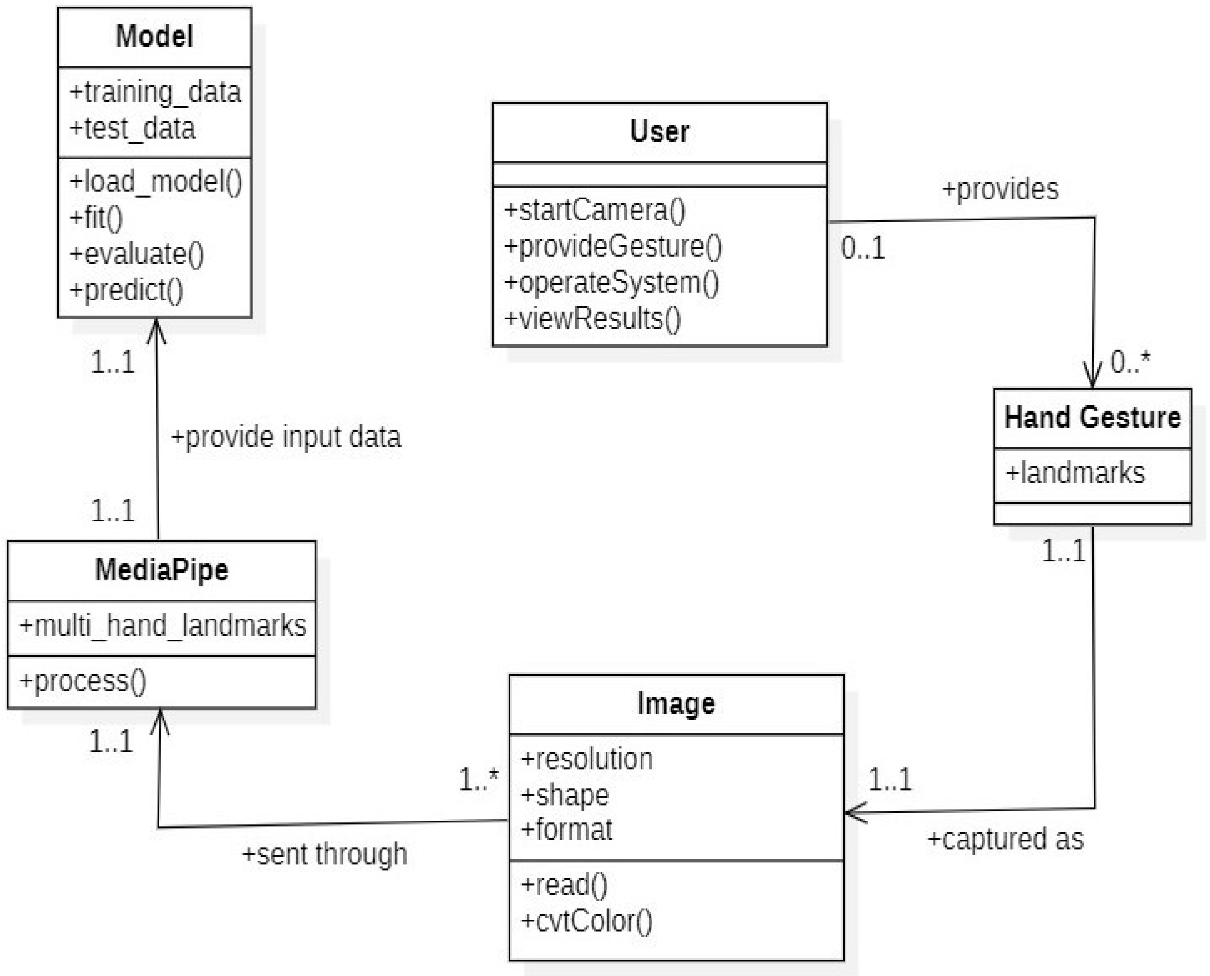


**Fig. 4.5.1:** Class Diagram

The above class diagram shows that there are five classes identified in the application. The associations between them tells us how these classes interact with each other.

- User

  The user provides hand gestures to the application. To do so, he can either start the in-built camera or a web-camera connected to the system.

- Hand Gesture

  Each of these hand gestures have 21 key features (landmarks).

- Image

  The hand gestures are captured as images through camera. These images can have different shapes, resolutions and formats depending on the system used for image acquisition. So, these images are read and converted to required format by the application and passed through MediaPipe.

- MediaPipe

  Now, this component processes the images and detects the presence of hand in the images. After detection, it extracts the 21 landmarks for each hand detected and passes this data as input to the Model class.

- Model

  The ANN model, which has been fitted on training data and evaluated based on testing data, in the application predicts the hand gesture provided by the user based on the input data from MediaPipe.

Based on the identified hand gesture, the application runs appropriate system operation. In this way, finally the user will be able to operate the system through this application and also view the outcome of this application.

### 4.6 Activity Diagram

The below activity diagram clearly depicts the flow of hand gesture recognition process in the application.

- When the application is run, the camera is started to capture user gestures.
- Captured images are passed through Mediapipe to identify if the user has provided any gesture or not.
  - When the user does not provide any gesture, the captured image will have no hand in it. Hence, MediaPipe will not be able to detect any hand. In such a case, the application will continue to capture images.
- Only when the user provides any hand gesture and the camera captures it, the MediaPipe will be able to detect the hand within the image and extract that particular hand gesture features. In such cases, these features are passed as input to a trained ANN model for gesture recognition.
- When the model predicts the class of the gesture, the corresponding system action is performed. The application will continue to capture images to track the user, waiting for the next hand gesture enabled command to be provided.

This process keeps continuing until the user closes the application or stops this continuous process by clicking on the Esc button.

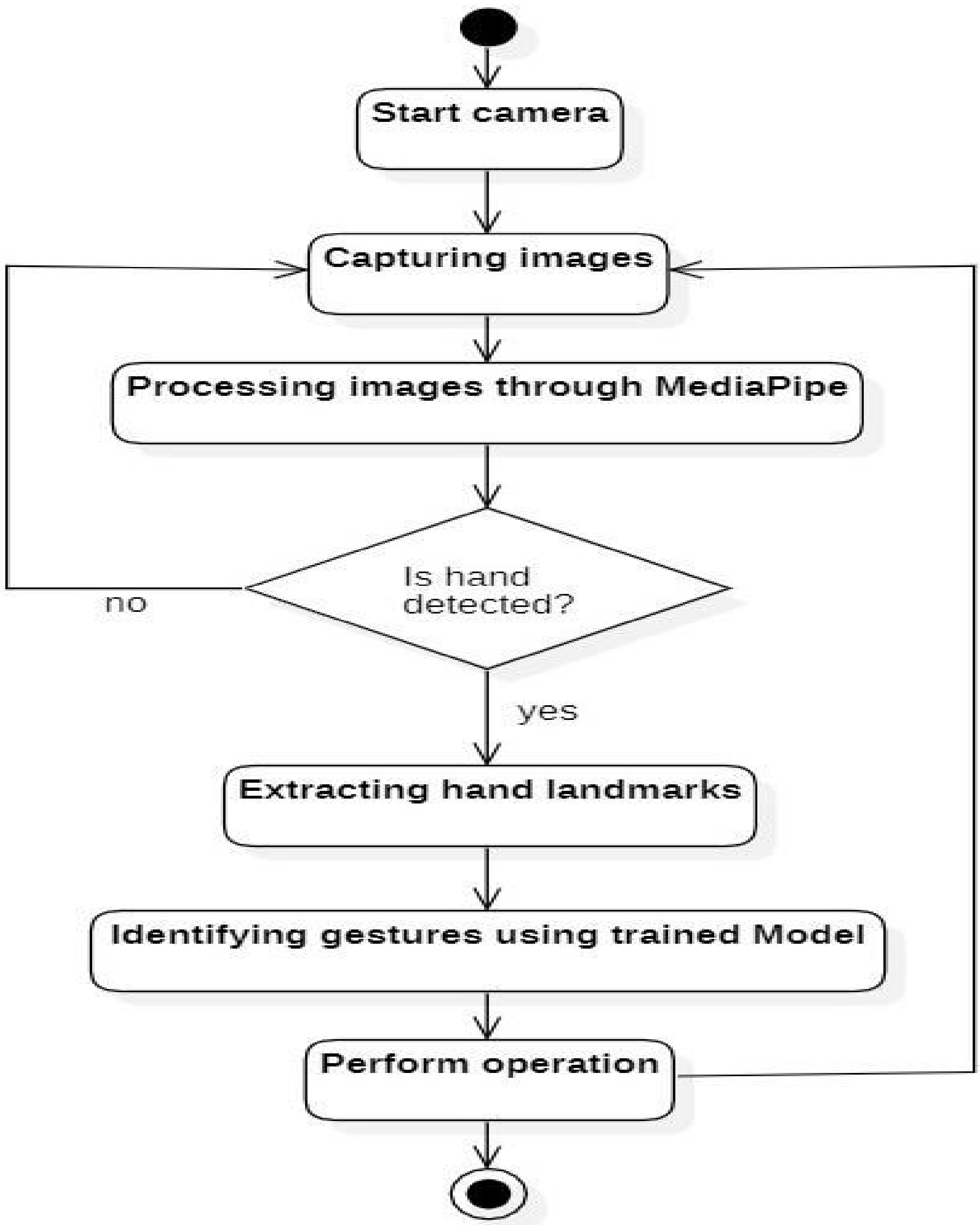


**Fig. 4.6.1:** Activity Diagram

### 4.7 Sequence Diagram

Figure 4.7.1 shows the processes involved and their sequence of execution in the application in order to help the user control the system through hand gestures. There are five lifelines identified in the application and the sequence of messages between them depicts the flow of execution.

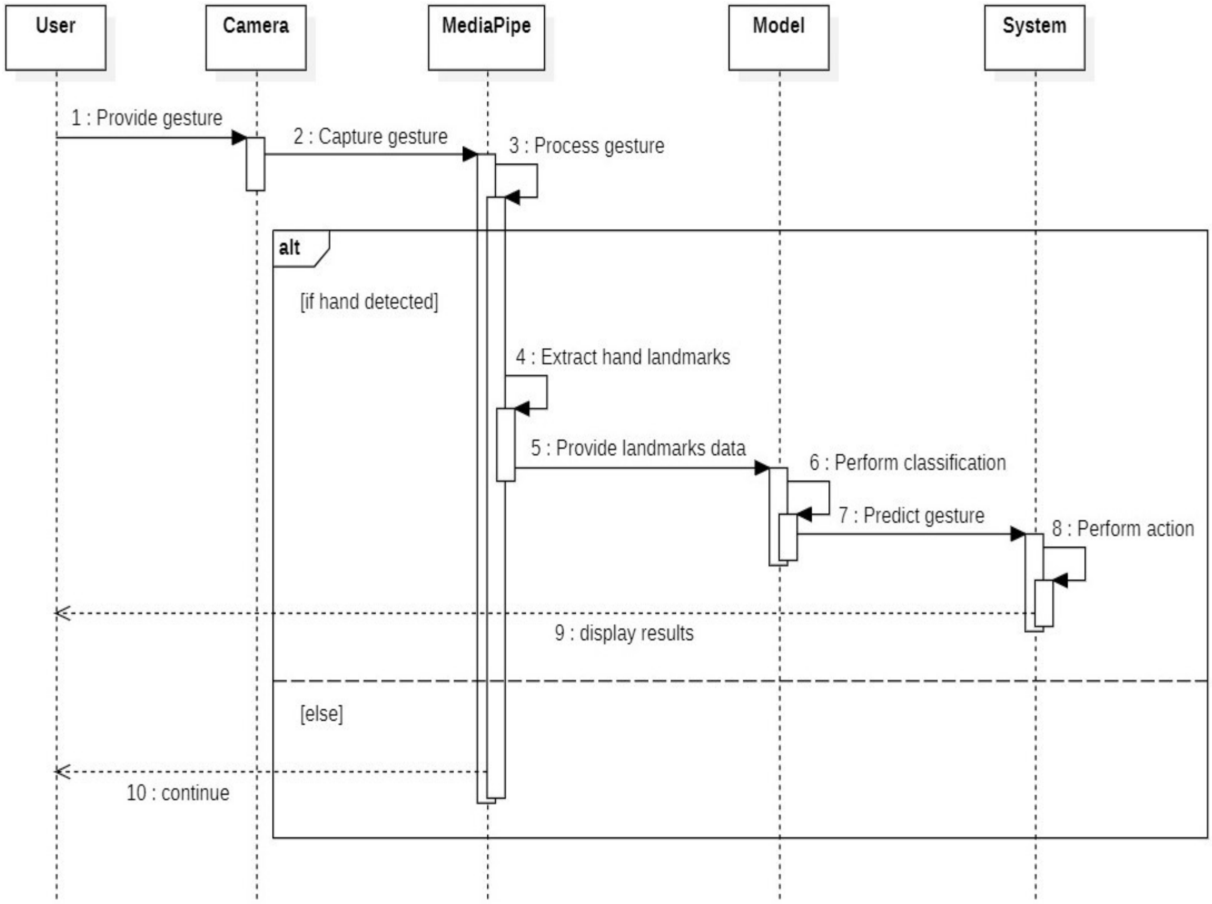


**Fig. 4.7.1:** Sequence Diagram

## CHAPTER 5

## IMPLEMENTATION

### 5.1 Dataset Creation

We have collected over 300 hand gesture images which includes all 5 static gestures: five, one two, thumb and closed palm finger formations. These hand gesture images are collected from the following Kaggle datasets:

- HG14 (HandGesture14) dataset [5]
- ASL Alphabet [6]
- Hand Gestures Dataset [7]

The following figures shows us the hand gesture images collected for all 5 classes of static hand gestures considered for this application. Each class includes hand gesture images taken in different backgrounds from different angles using different hands. Including such diversity in dataset is crucial for our model to work effectively in real-time.

Figure 5.1.1 shows 63 hand gesture images that contain 'five' gesture formation. Some images shows that the user can either keep fingers apart to indicate a five fingers formation or keep them close to each other like an open palm still indicating a five fingers formation.

**Fig. 5.1.1:** 'Five' Hand Gesture Images

Figure 5.1.2 shows 56 hand gesture images that contain ‘one’ gesture formation.


**Fig. 5.1.2:** ‘One’ Hand Gesture Images

Figure 5.1.3 shows 63 hand gesture images that contain ‘two’ gesture formation.


**Fig. 5.1.3:** ‘Two’ Hand Gesture Images

Figure 5.1.4 shows 52 hand gesture images that contain ‘thumb’ gesture formation.

**Fig. 5.1.4:** ‘Thumb’ Hand Gesture Images

Figure 5.1.5 shows 66 hand gesture images that contain ‘closed palm’ gesture formation.

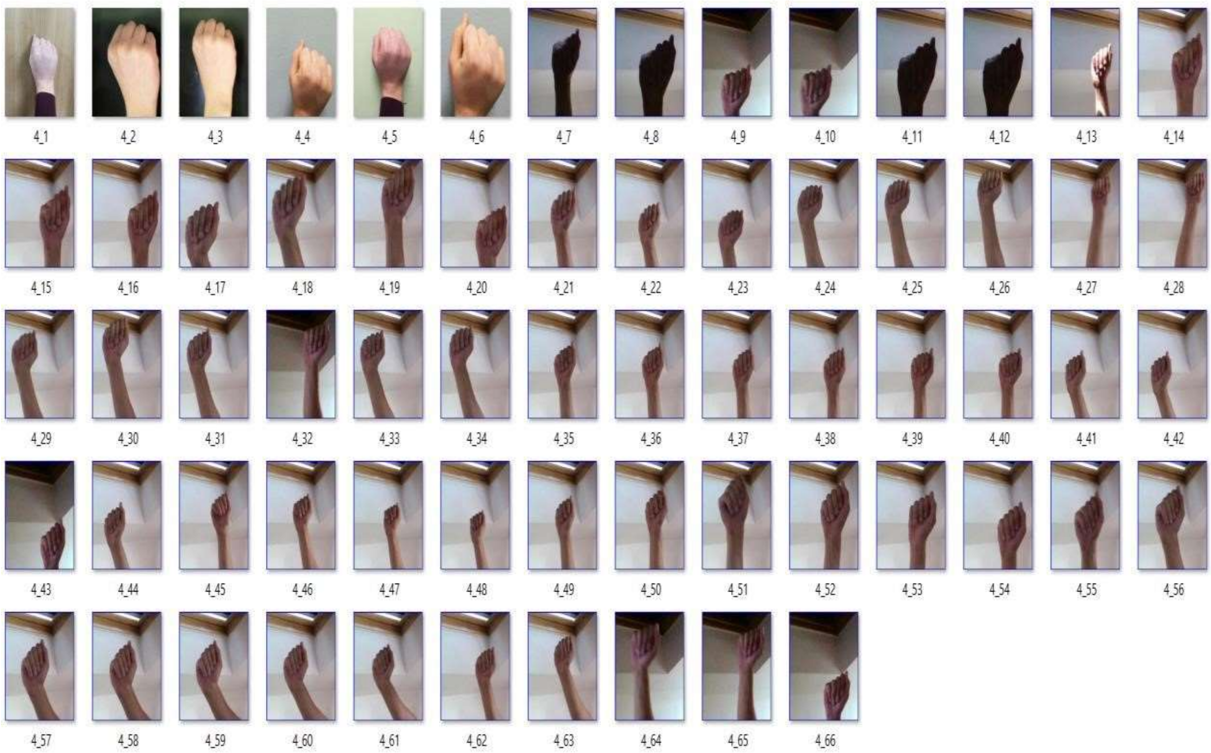

**Fig. 5.1.5:** ‘Closed Palm’ Hand Gesture Images

Hand gesture images dataset is ready. But, as discussed in 'proposed methods' section, our ANN model takes as input labelled hand gesture landmarks and not the hand gesture image itself. So, from each hand gesture image we have extracted the required hand key-features and created a training dataset. To do so, we have used MediaPipe's Hand Landmark Model to extract the 21 keypoints (landmarks) for each hand in the dataset.

Figure 5.1.7 shows us the code written to extract landmarks from 300 hand gesture images and label them with the corresponding hand gesture label, as shown in Table 5.1.1, to create a labelled training dataset for the ANN model. Labelled dataset is required for the model to learn and classify images with more accuracy.

| Hand Gesture | Label |
|---|---|
| Five | 0 |
| One | 1 |
| Two | 2 |
| Thumb | 3 |
| Closed Palm | 4 |

**Table 5.1.1:** Hand Gestures & their Labels

The 'Dataset_Creation_CSV.py' code is explained as follows:

- Fisrt, all the required libraries like os, cv2, mediapipe and csv have to be imported.
- MediaPipe's 'Hands' solution object instance is created, which will be used to detect hand gesture and extract its features. Using this object instance, we set the maximum number of hands that it should detect from an image as 1 and the minimum hand detection confidence to 0.5. If any image which is not clear and MediaPipe is not able to detect hand in an image, then the detection confidence falls. In such a case, we will not be able to extract data.
- As seen from the code, the extracted features of a hand gesture along with its label are stored in CSV format in a CSV file named 'gesture_dataset_new.csv'. This file, if not exists, is created when opened and a csv writer object is created on it.
- The paths to all the 5 hand gesture images dataset are stored in a list. Each image of each class of gestures has to be processed through MediaPipe to extract landmarks and have to be labelled accordingly. To accomplish this repetitive process, here a for loop is used which for each loop calls a function (load_images_from_folder()) on each gesture class folder along with its label.
- So, initially the label counter (g_c) is set to one. As the loop moves forward to next class of gesture images, the label counter also gets incremented accordingly.

- For every gesture class, the function first tries to access each of its images from the provided folder path with the help os library functions listdir() and join().
  - Every image in the folder is read from the folder using openCV library function imread(). Since, openCV processes images exceptionally in BGR format, we used its cvtColor() function to convert the image format to RGB.
  - Then the read image is processed by MediaPipe with the help of its hands.process() function which returns an object with landmarks details.
  - If this object's multi_hand_landmarks attribute is empty, indicates that no hand is detected. In such case we proceed to next image.
  - If it is not empty, then it contains landmarks of each hand MediaPipe has detected.
  - The landmarks are a list of (x,y) normalised coordinates of the 21 handknuckles of the detected hand.
  - These 21 coordinates (42 datapoints) along with the label are written into the CSV file as one data row.
- Finally, after all images are processed the CSV file is closed.

In this way, features from 300 hand gesture images are extracted and labelled to form the required training dataset. Figure 5.1.6 shows the 21 landmarks (42 datapoint) of a 'five' hand gesture image. The last record '0' appended at the end is the label given to this gesture class. All the data is comma separated and stored as a row in the CSV file.

```
0.4749143123626709,0.8473268747329712,0.33529534935951233,0.7689113020896912,0.2723129093
647003,0.6488218307495117,0.2586408257484436,0.519392728805542,0.2262246012687683,0.422132
0152282715,0.42999187111854553,0.5407705307006836,0.4216950535774231,0.35506874322891235,
0.42109084129333496,0.24440070986747742,0.42540279030799866,0.15602847933769226,0.5290554
761886597,0.5557278394699097,0.5603717565536499,0.3616945147514343,0.5830839276313782,0.23
681756854057312,0.6052291393280029,0.1418623924255371,0.6023471355438232,0.59331583976745
6,0.6442290544509888,0.42973122000694275,0.667263388633728,0.31628113985061646,0.686616361
1412048,0.2321104109287262,0.6586114168167114,0.6393271088600159,0.7424126863479614,0.5352
187156677246,0.7948282361030579,0.46998274326324463,0.8401288986206055,0.415347695350647,
0
```

**Fig. 5.1.6:** Labelled Training Data Sample

```
import os from os
import listdir import
cv2 import mediapipe
as mp
import csv

mp_hands = mp.solutions.hands hands =
mp_hands.Hands(max_num_hands=1, min_detection_confidence=0.5)

file = open('gesture_dataset_new.csv', 'w', newline='') writer
= csv.writer(file)

def load_images_from_folder(folder,label):
for imagename in os.listdir(folder):
    #print(imagename)       image_frame =
cv2.imread(os.path.join(folder, imagename))       image =
cv2.cvtColor(image_frame, cv2.COLOR_BGR2RGB)       result =
hands.process(image)       if result.multi_hand_landmarks:
for handslms in result.multi_hand_landmarks:
        landmarks = []            for
lm in handslms.landmark:
landmarks.append(lm.x)
landmarks.append(lm.y)
landmarks.append(label)
writer.writerow(landmarks)

gestures=["C:/MajorProject/Dataset/Gesture0/","C:/MajorProject/Dataset/Gesture
1/","C:/MajorProject/Dataset/Gesture2/","C:/MajorProject/Dataset/Gesture3/","C:
/MajorProject/Dataset/Gesture4/"]
g_c = 0 for ges_class in gestures:
  load_images_from_folder(ges_class, g_c)
g_c = g_c+1

file.close()
```

**Fig. 5.1.7:** 'Dataset_Creation_CSV.py' Code

### 5.2 Model Creation & Training

In order to process the hand gesture image captured by the user's camera, the application is making use of MediaPipe to extract the features of the hand gesture in the image. These extracted features have hidden patterns in them which can be used to identify the hand gesture or classify the hand gesture as one among the five gesture classes. These hidden patterns can be better explored using artificial neural networks.

So, we have used Python's Keras Models API to build a neural network model for our application to recognise hand gestures. Seqential API is used in our application which simply arranges the Keras layers in a sequential order and the data flows from one layer to another layer in the given order until the data finally reaches the output layer. We have used only Dense Keras layer, where every neuron in a dense layer takes the input from all the other neurons of the previous layer.

Since Keras models takes input data in array format, our training data stored in CSV format in the 'gesture_dataset_new.csv' file is loaded and converted into numpy array using NumPy library function loadtxt(). This training data is then split into two parts:

- train_X

  It is the input data to the model consisting of a list of all the 21 landmarks (42 datapoints) of each hand gesture in the training dataset. The model tries to find patterns from this data.

- train_Y

  It is the target data to the model. It consists of a list of labels in an order corresponding to the data in input data (train_X). The model tries to classify similar patterns under one label.

The model we built is used to classify the input hand gesture image into one of the 5 labelled gesture classes based on the extracted landmarks from that image. The application's 'Gesture Classification Model' is built in the following stages:

- Model Creation (Sequential() & add() methods)

  Our model has an input layer of size 42 (42 datapoints). There are two other dense layers. The first dense layer has 21 nodes and each node receives 42 data inputs from the input layer. The activation function used for this layer is 'ReLU' which is the most common function used for hidden layers. The second dense layer is the output layer with 5 nodes (each node representing one of the 5 labelled gesture classes) and each node receives 21 data inputs from the previous hidden dense layer. The activation function used for this layer is 'Sigmoid' that maps input values to a value between 0 and 1, making it useful for classification problems. Therefore, our model comprises of one input layer, one hidden layer and one output layer

- Model Compilation (compile() method)

  Model is compiled to configure it for training. The compile method takes a loss function and an optimizer function as parameters. The loss function measures how far the model's output are from the desired output. Here, 'sparse_categorical_crossentropy' is used, which is a loss function for multi-class classification model where the output label is assigned integer value (0, 1, 2, 3,). The optimizer function tries to adjust the inner parameters of the model in order to minimize loss and increase accuracy of the model. Here, 'adam' is used which is a best optimizer function.

- Model Training (fit() method)

  Once the model is configured for training, it trained on the train_X (input data) and train_Y (target data). Here, batch_size is initialised to 20 meaning while training after every batch 20 hand gestures and labels the model has to update its inner variables. Also, 50 epochs means that the model has to train for a total number of 50 full iterations on the training dataset. That is, 300x50 = 15000 examples in total.

After the model is built, it is saved in a folder named 'trained_model_new'. Figure 5.2.1 shows us the code written for creating and training the application's gesture classification model.

```
From tensorflow.keras.models import
Sequential from tensorflow.keras.layers import
Dense from numpy import loadtxt

dataset = loadtxt('gesture_dataset_new.csv', delimiter=',')
train_X = dataset[:, 0:42]
train_Y = dataset[:, 42]

model = Sequential() model.add(Dense(21,
input_shape=(42, ), activation='relu')) model.add(Dense(5,
activation='sigmoid'))

model.compile(loss='sparse_categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])

model.fit(train_X, train_Y, epochs=50, batch_size=20)

model.save('trained_model_new')
```

**Fig. 5.2.1:** 'Model_training.py' Code

Figure 5.2.2 shows the summery of the model created for our application. We can clearly see the output shape and number of weights in each layer.

- The first dense layer with 21 nodes produces 21 datapoints as output, which are input to each of the 5 nodes of the second (output) dense layer.
- The output dense layer's output shape is 5, where each output value falls between 0 and 1 indicating the confidence score (probability) of the classification into a particular label (class) being correct.

The parameters are the total number of inner variables in that corresponding layer.

- dense layer: 21x42+21 = 903 parameters
- dense_1 layer: 5x21+5 = 110 parameters
- Total params: 903+110 = 1013 parameters

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 21)                903

 dense_1 (Dense)             (None, 5)                 110

=================================================================
Total params: 1,013
Trainable params: 1,013
Non-trainable params: 0
_________________________________________________________________
None
```

**Fig. 5.2.2:** Sequential Model Summary

Figure 5.2.3 shows the output in the terminal, when the 'Model_training.py' is run. A total of 50 epochs are run. As the training move forward by every epoch, the loss value keeps decreasing while the accuracy value keeps increasing gradually. This happens because for each epoch, about 15 times (300/20) backpropagation takes place where loss is calculated and necessary optimization is made by updating the model's parameter values to maximize the accuracy by minimizing the loss.

```
 1/11 [=>............................] - ETA: 0s - loss: 0.6089 - accuracy: 0.9000
11/11 [==============================] - 0s 2ms/step - loss: 0.7087 - accuracy: 0.8372
Epoch 41/50

 1/11 [=>............................] - ETA: 0s - loss: 0.8137 - accuracy: 0.7500
11/11 [==============================] - 0s 1ms/step - loss: 0.6969 - accuracy: 0.8512
Epoch 42/50

 1/11 [=>............................] - ETA: 0s - loss: 0.7055 - accuracy: 0.8500
11/11 [==============================] - 0s 2ms/step - loss: 0.6811 - accuracy: 0.8558
Epoch 43/50

 1/11 [=>............................] - ETA: 0s - loss: 0.8176 - accuracy: 0.7500
11/11 [==============================] - 0s 2ms/step - loss: 0.6693 - accuracy: 0.8651
Epoch 44/50

 1/11 [=>............................] - ETA: 0s - loss: 0.6021 - accuracy: 0.8500
11/11 [==============================] - 0s 2ms/step - loss: 0.6554 - accuracy: 0.8605
Epoch 45/50

 1/11 [=>............................] - ETA: 0s - loss: 0.7521 - accuracy: 0.8000
11/11 [==============================] - 0s 2ms/step - loss: 0.6435 - accuracy: 0.8465
Epoch 46/50

 1/11 [=>............................] - ETA: 0s - loss: 0.5989 - accuracy: 0.8500
11/11 [==============================] - 0s 2ms/step - loss: 0.6363 - accuracy: 0.8512
Epoch 47/50

 1/11 [=>............................] - ETA: 0s - loss: 0.7091 - accuracy: 0.9000
11/11 [==============================] - 0s 2ms/step - loss: 0.6255 - accuracy: 0.8744
Epoch 48/50

 1/11 [=>............................] - ETA: 0s - loss: 0.4885 - accuracy: 0.9000
11/11 [==============================] - 0s 2ms/step - loss: 0.6124 - accuracy: 0.8698
Epoch 49/50

 1/11 [=>............................] - ETA: 0s - loss: 0.7201 - accuracy: 0.8000
11/11 [==============================] - 0s 2ms/step - loss: 0.5952 - accuracy: 0.8744
Epoch 50/50

 1/11 [=>............................] - ETA: 0s - loss: 0.7393 - accuracy: 0.8000
11/11 [==============================] - 0s 2ms/step - loss: 0.5855 - accuracy: 0.8791
```

**Fig. 5.2.3:** Model Training Output

### 5.3 Hand Gesture Classification

The trained model can now be used to recognise hand gestures provided by the user in real-time. Figure 5.3.1 shows the main code written for this application. When this application is run, it starts the camera and keeps capturing hand gestures provided by the user and performs relevant system operations for the user in real-time.

The ‘Gesture_classification.py’ code is explained as follows:

- First, all the required libraries like os, sys, cv2, numPy, mediapipe and tensorflow have to be imported.
- MediaPipe’s ‘Hands’ solution object instance is created and configured. Also, MediaPipe’s ‘drawing_utils’ object is initialised, which can be used to draw landmarks on the hand gesture image captured by the application and display it to the user.
- The save ‘trained_model_new’ is loaded into the application using load_model() method of tensorflow.keras.models module. A list of names of all 5 gesture classes is created indexed in the order of their corresponding labels.
- OpenCV is used for capturing images through camera. VideoCapture() method of cv2 library is used to start a camera which is passed as a parameter to it. User can decide if he/she wants to access the in-built camera or web-camera. By default, the application will access the inbuilt

camera as the source is set to 0 (s=0). If the user wants to access web-camera, he has to provide 1 as the argument while running the application. In such a case, the value of source will be updated to 1 (s=1).

- We have used namedWindow() method to create a window with a suitable name to display images and videos on the screen.
- Using waitKey() method allows us to display the window for given milliseconds or until any key is pressed. Since we want to keep displaying the window to the user, it is used in a while loop. In our code, the window keeps changing display for every 1 sec. We can both stop displaying the window and exit the loop by pressing on 'Esc' button (27).
- As we continue to loop, the code inside while loop is executed infinitely until 'Esc' is pressed:
  - Using read() method we read image frames at that particular time from the video coming from the camera source in real-time.
  - We convert the frame from BGR to RGB format and pass it through MediaPipe for processing.
  - If hand is not detected, then the frame is displayed to the user as it is.
  - If any hand is detected in the frame, we continue to extract landmarks from it. We also draw those landmarks on the frame using draw_landmarks() method of MediaPipe's drawing_utils module.
  - The extracted landmarks are appended to an empty list and passed through our model for classification using predict() method.
  - This method returns a numPy array of predictions. Since we have 5 nodes (labelled classes) in outer layer of our model, we get 5 confidence scores.
  - The gesture class with highest confidence score is identified as the hand gesture that the user provided. The name of the gesture identified is then put on the frame (along with landmarks) using cv2 library putText() method.
  - Simultaniously, depending on the recognised gesture, corresponding system operation is performed. To do so, we have used os library system() method to run the corresponding system shell command.
  - At last, the updated frame with all details is displayed to the user through the window using imshow() method of openCV.
- Once the user presses 'Esc' button the video source is closed and display window is also destroyed. The application stops running.

Hence, we are able to implement a solution wherein a user can use hand gestures to control basic system operations like shutdown, increase or decrease volume, open web browser and media player.

```
Import os import
sys import numpy
as np import cv2
import mediapipe as mp from
tensorflow.keras.models import load_model

mpHands = mp.solutions.hands hands =
mpHands.Hands(max_num_hands=1, min_detection_confidence=0.5) mpDraw
= mp.solutions.drawing_utils

model = load_model('trained_model_new') gesture_classes
= ['five','one','two','thumb','closed palm']

s = 0
if len(sys.argv)>1:
   s = int(sys.argv[1])

source = cv2.VideoCapture(s)
win_name = "Camera Preview"
cv2.namedWindow(win_name, cv2.WINDOW_NORMAL)

while cv2.waitKey(1000)!=27:
   has_frame,frame = source.read()
if not has_frame:
      break
   framergb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
   result = hands.process(framergb)
   if result.multi_hand_landmarks:
     for handsIms in result.multi_hand_landmarks:
mpDraw.draw_landmarks(frame, handsIms,
mpHands.HAND_CONNECTIONS)
        landmarks = []
        for Im in handsIms.landmark:
landmarks.append(Im.x)
landmarks.append(Im.y)
  prediction = model.predict([landmarks])
print(prediction)
         index = np.argmax(prediction)
```

```
        gesture = gesture_classes[index]
         print(gesture)
         cv2.putText(frame, gesture, (10, 50), cv2.FONT_HERSHEY_SIMPLEX, 1,
(0,0,255), 2, cv2.LINE_AA)
if index==0:
     os.system('start Chrome.exe')
elif index==2:
     os.system('start wmplayer')
elif index==4:
     os.system('shutdown /s /t 60')
print('Enter 0 to abort shutdown')
x = int(input())
if x==0:
     os.system('shutdown /a')
elif index==1:
     os.system('setvol +10')
 elif index==3:
     os.system('setvol -10')
  cv2.imshow(win_name, frame)

source.release()
cv2.destroyWindow(win_name)
```

**Fig. 5.3.1:** 'Gesture_classification.py' Code

Figure 5.3.2 shows the output of the two print statements in the above code. The model after prediction returns a numPy array of predictions as probabilities.

- In the first array, second index element has the highest value. Hence, the gesture corresponding to label 2 is the recognised gesture. The gesture that is assigned label 2 is 'Two' gesture formation. As a result, the media player will be opened.
- In the second array, first index element has highest score. Operation corresponding to 'One' gesture is performed, increasing the system volume.
- In the fourth array, zeroth index element has highest score. Zero is the label assigned to gesture 'Five'. As a result, browser is opened.
- In the fifth array, third index element has highest value. 'Thumb' gesture is assigned a label of 3, so as a result system volume is decreased.

Similarly, if 'closed palm' is provided by the user, then the system will shutdown. But before the system shutdowns, Windows will display a prompt saying that it will shutdown in a minute.

```
1/1 [==============================] - ETA: 0s
1/1 [==============================] - 0s 139ms/step
[[0.37488082 0.7518031  0.8153728  0.19826555 0.34390298]]
two

1/1 [==============================] - ETA: 0s
1/1 [==============================] - 0s 88ms/step
[[0.30601412 0.8295198  0.5920369  0.5520349  0.12309445]]
one

1/1 [==============================] - ETA: 0s
1/1 [==============================] - 0s 30ms/step
[[0.20736703 0.88090765 0.8114505  0.22816224 0.41834936]]
one

1/1 [==============================] - ETA: 0s
1/1 [==============================] - 0s 30ms/step
[[0.8731525  0.64658606 0.5783752  0.40114972 0.04172346]]
five

1/1 [==============================] - ETA: 0s
1/1 [==============================] - 0s 81ms/step
[[0.11510202 0.6285637  0.5242618  0.68153256 0.35146797]]
thumb
```

**Fig. 5.3.2:** Sample Model Predictions

# CHAPTER 6

# TESTING

## 6.1 Test Cases

We saw in the previous section, that our gesture classification model is able to recognise user hand gestures. Now, we will see some test cases where each gesture is provided by the user to the application in different forms like using right hand, using left hand, showing the gesture from front or backwards, and from various angles. And see if our model can recognise the hand gestures correctly or not. The landmarks that you see on the hands are detected by MediaPipe. Based on those landmarks, the hand gestures are recognised by our 'trained_model_new' gesture classification model, which is created using Keras Sequential API and trained on our 'gesture_dataset_new.csv' training dataset.

The test cases are distributed as follows:

- Figures 6.1.1 to 6.1.5 shows test cases when user provides 'Five' hand gesture.
- Figures 6.1.6 to 6.1.10 shows test cases when user provides 'One' hand gesture.
- Figures 6.1.11 to 6.1.15 shows test cases when user provides 'Two' hand gesture.
- Figures 6.1.16 to 6.1.20 shows test cases when user provides 'Thumb' hand gesture.
- Figures 6.1.21 to 6.1.25 shows test cases when user provides 'Closed Palm' hand gesture.