## ABSTRACT

Sign language serves as a vital mode of communication for individuals with hearing and speech challenges. Yet, a notable divide exists between the hearing-impaired and the general population due to limited sign language proficiency. This review paper explores recent strides in assistive technology, particularly the development of sign language recognition and translation methods. It delves into the obstacles faced by the hearing-impaired, emphasizing the urgency of overcoming these barriers. The paper provides a comprehensive overview of diverse methodologies, encompassing machine learning, computer vision, and natural language processing, employed to interpret and translate sign language into text or speech. It stresses the need to extend these technologies to embrace the world's various sign languages, fostering inclusivity. The potential of these innovations to enhance communication, education, healthcare, and social interaction for the hearing-impaired is highlighted. This review paper offers valuable insights into the evolving field of sign language recognition and translation, working towards a world where all individuals, regardless of their hearing abilities, can engage fully in society.

### TABLE OF CONTENT

**CHAPTER NO TITLE PAGE NO**

|  |  |  |
| --- | --- | --- |
| **ABSTRACT**  **LIST OF FIGURES**  **1 INTRODUCTION** | v  v iii |  |
| **1.1** Overview | 2 |  |
| **1.2** Problem Definition | 3 |  |
| **2 LITERATURE SURVEY** | 4 |  |
| **3 SYSTEM ANALYSIS** | 11 |  |
| **3.1** Existing System | 12 |  |
| **3.2** Proposed System | 13 |  |
| **3.3** Requirement Analysis And Specification | 16 |  |
| **3.4** Technology Stack | 20 |  |
| **4 SYSTEM DESIGN** | 27 |  |
| **4.1** UML Diagrams | 28 |  |
| **4.1.1** Use Case Diagram | 28 |  |
| **4.1.2** Data Flow Diagram | 29 |  |
| **4.1.3** Flow Chart Diagram | 30 |  |
| **4..1.4** Activity Diagram | 31 |  |
| **5 SYSTEM ARCHITECTURE** | 32 |  |
| **5.1** Architecture Overview | 33 |  |
| **5.2** Module Design Specification | 34 |  |
| **6 SYSTEM IMPLEMENTATION** | 36 |  |
| **6.1** Program | 37 |  |
| **7 SYSTEM TESTING** | 49 |  |
| **7.1**Test Cases & Reports | 50 |  |
| **7.2** Unit Testing | 51 |  |

|  |  |  |
| --- | --- | --- |
| **7.3** Integration Testing | 52 |  |
| **8 CONCLUSION** | 53 |  |
| **8.1** Conclusion and Future Enhancements | 54 |  |
| **APPENDICES** | 56 |  |
| A.1 Sample Screens | 57 |  |
| **REFERENCES** | 59 |  |

### TABLE CONTENT

### LIST OF FIGURES

**FIG NO TITLE PAGE NO**

|  |  |
| --- | --- |
| **3.3.1** System Implementation | 17 |
| **3.3.2** Removal of Background | 18 |
| **3.3.3** Labelling Skin Region | 19 |
| **3.3.4** Accuracy Performance | 19 |
| **4.3.1** Use Case Diagram | 28 |
| **4.3.2** Data Flow Diagram | 29 |
| **4.3.3** Flow Chart Diagram | 30 |
| **4.3.4** Activity Diagram | 31 |
| **5.1** Architecture Diagram | 33 |
| **7.3.1** System Training Code | 50 |
| **7.3.2** Functions | 50 |
| **7.3.3** Sample Output Screenshot | 51 |
| **A.1.1** Jupyter Document Page | 57 |
| **A.1.2** Training The System | 57 |
| **A.1.3** Camera Capturing | 58 |
| **A.1.4** Captured Gesture Converted To Text | 58 |

### 

### 

### CHAPTER 1

### INTRODUCTION

* 1. **OVERVIEW**

**s**ign language interpretation through the synergy of Machine Learning (ML) and Image Processing, augmented by Sign Language Recognition (SLR), represents a pivotal advancement in the realm of accessibility and communication. This innovative approach harnesses the power of ML algorithms to transform complex sign language gestures, expressions, and movements into comprehensible text or speech, making communication seamless for the deaf and hard-of-hearing individuals. In this process, Image Processing plays a vital role in capturing and processing visual data, allowing for the recognition of intricate sign language expressions even in diverse lighting conditions. SLR technology further refines the system by accurately detecting and interpreting sign language movements, contributing to the precision and fluency of the interpretation.

The underlying concept behind this technology is the convergence of these components. Image Processing enables the system to capture and preprocess video data, eliminating noise and optimizing the visual input. SLR is the linchpin that facilitates the recognition of sign language gestures and their translation into meaningful communication. ML algorithms are the driving force that underlies the intelligence of this system, learning and adapting to the nuances and complexities of sign language as it encounters a variety of gestures and expressions.

This innovative approach not only breaks down the communication barriers faced by the deaf and hard-of-hearing community but also serves as a beacon of inclusivity and accessibility. By translating sign language into spoken language or text, it enables effective interaction between individuals who use sign language and those who do not. In education, healthcare, customer service, and various other domains, this technology has the potential to revolutionize communication and enrich the lives of millions. It is a testament to the boundless possibilities that emerge when the worlds of ML, Image Processing, and SLR converge, ultimately fostering a more inclusive and connected society.

### PROBLEM DEFINITION

### Deaf and hard-of-hearing individuals face formidable communication challenges, underscoring the need for an effective Sign Language Interpretation system employing Machine Learning and Image Processing. The primary obstacle is to ensure dependable real-time translation of sign language into spoken or written language, facilitating integration in education, healthcare, and social interactions. The project seeks to surmount key challenges, including the frustration of ineffective communication, limited access to vital information, obstacles posed by costly and inconvenient communication tools, the demand for precision and immediacy in interpretation, the necessity of a user-friendly interface, and the imperative consideration of cultural and linguistic diversity, all while upholding data privacy and ethical standards. Ultimately, the project aspires to deliver a real-time, culturally sensitive Sign Language Interpretation system that transcends these challenges, fostering inclusivity and accessibility in diverse domains.

# CHAPTER 2 LITERATURE SURVEY

### LITERATURE SURVEY

**PAPER 1: IoT Based** **Sign Language Recognition System**

This paper addresses the pressing need for effective communication between deaf and mute individuals and those who can hear and speak. It emphasizes the significance of sign language as a vital means of communication for the mute but acknowledges the challenge faced by non-mute individuals in understanding sign language gestures. With over 125 distinct sign languages used globally and various communication disorders leading to mutism, the paper proposes an innovative solution: an IoT-based sign language recognition system. This system integrates machine learning, natural language processing, and IoT principles to bridge the communication gap. It comprises a mobile application for translating sign language into digital voice and a wearable hand glove with sensors that recognize hand gestures. By combining these components, the research aims to facilitate seamless communication, ultimately enhancing understanding and reducing the barriers between these two communities.

[1]

**PAPER 2:** **Sign Language Translator and Gesture Recognition**

This research focuses on the development of a sign language translator system, with a specific emphasis on hand-spelling in American Sign Language (ASL). Sign language, a vital mode of communication for those unable to speak, presents a global challenge for effective communication, with ASL being a prominent choice. This research bridges the domains of sign language and computer engineering, highlighting the significance of facial recognition and body movement interpretation in aiding communication for individuals with hearing impairments. It delves specifically into hand-spelling intricacies within ASL, where multiple fingers are used to symbolize alphabet characters. The innovation lies in a specialized glove equipped with sensors, including flex sensors, contact sensors, and accelerometers, all processed through an Arduino microcontroller. This system translates hand gestures into corresponding alphabet characters, offering a promising solution for enhancing communication between ASL users and technology. [2]

**PAPER 3: Real-time Indian Sign Language (ISL) Recognition**

This research paper focuses on the development of a real-time system for recognizing hand poses and gestures in the Indian Sign Language (ISL). The primary objective is to bridge communication gaps between ISL users and those unfamiliar with the language, offering significant potential for improving communication for speech-impaired individuals. The paper highlights the challenges in comprehending ISL gestures and emphasizes the user-friendly design of the system, which utilizes a smartphone camera for capturing hand poses and gestures without additional hardware. The system employs a range of techniques for hand detection and tracking, along with grid-based feature extraction and classification algorithms such as k-Nearest Neighbors (k-NN) and Hidden Markov Models (HMMs). Impressively, it achieves high accuracy rates of 99.7% for static hand poses and 97.23% for gesture recognition. The research showcases keywords and techniques including ISL recognition, gesture recognition, and HMM. Future work may extend the system to recognize two-handed gestures, indicating ongoing research possibilities in the field of ISL translation. [3]

**PAPER 4: Indian Sign Language Recognition Using Machine Learning Techniques**

This paper underscores the critical need for an automatic system to facilitate communication between the hearing-impaired community and the general population, with a focus on Indian Sign Language (ISL). It recognizes sign language as a complete and distinct form of communication, highlighting its unique syntax and grammar. The paper categorizes sign languages into dynamic and static types, crucial for understanding sign language recognition. It delves into vision-based gesture recognition approaches, particularly in the context of ISL, and introduces a system designed to recognize immobile numeric signs using a standard digital camera. The system aims to convert isolated digit signs into text and relies on a database comprising 5000 ISL digit images. A comparison of classifiers reveals the superiority of the k-Nearest Neighbor classifier in achieving accurate ISL sign recognition. This research addresses the lack of automatic ISL translators and paves the way for improved communication for the hearing-impaired.[4]

**PAPER 5: Sign Language Recognition System**

This paper outlines the development of a sign language recognition system, primarily focusing on an intelligent glove as a communication bridge for the deaf-mute community and those unfamiliar with sign language. The glove incorporates various sensors, including flex sensors and an inertial measurement unit (IMU), to capture hand shape and movements. Gesture recognition relies on monitoring finger orientation and hand motions in a three-dimensional space. Machine learning algorithms, including Support Vector Machine (SVM), Naïve Bayes, and decision tree, are used for preprocessing and interpreting sensor data, achieving a 90% accuracy rate with SVM. Recognized signs are then transformed into audible speech through a speech conversion function. Additionally, the paper highlights the development of a two-way communication application that processes sensor data, translating it for further analysis, and hints at the integration of a Raspberry Pi for additional translations. The core objectives are to design an intelligent glove for ASL communication, improve accuracy through preprocessing, and create a two-way communication application. [5]

**PAPER 6: A Simple Multi-Modality Transfer Learning Baseline for Sign Language Translation**

The proposed methodology for "A Simple Multi-Modality Transfer Learning Baseline for Sign Language Translation" by Yutong Chen et al. involves several key steps. It begins with data collection, where a diverse dataset containing sign language videos and their corresponding textual translations is gathered. Preprocessing aligns video frames with text to ensure synchronization. Transfer learning techniques are then applied to leverage knowledge from related tasks. A simple baseline model is developed for sign language translation, followed by training on the multi-modal dataset. Evaluation metrics assess the model's performance, and iterative experimentation refines it further, exploring variations, data augmentation, and training strategies. Finally, a comparison with existing systems gauges effectiveness, with comprehensive documentation for future reference. In summary, this methodology aims to establish a strong baseline model for sign language translation through a systematic and iterative approach, enhancing accuracy and the translation of sign language gestures into text. [6]

**PAPER 7: Sign Language Interpretation**

The proposed methodology aims to enhance communication and accessibility for the hearing-impaired community by bridging the gap between spoken language and sign language, with a focus on finger spelling. The process involves capturing audio input, converting it to text using the Google Speech API, and employing a dependency parser to analyze sentence structure. Indian Sign Language (ISL) representations are generated from the transcribed text and conveyed through a signing avatar. Challenges in speech recognition are addressed with techniques like Mel-Frequency Cepstral Coefficients (MFCCs), and the system architecture defines success and failure conditions. This project offers a smart solution to facilitate effective communication for the hearing-impaired, referencing related works in the literature survey to provide context for the proposed methodology. [7]

**PAPER 8: Sign Language Recognition and Translation: A Multidisciplined Approach From the Field of Artificial Intelligenc**e

paper delves into the realm of sign language recognition and translation, shedding light on the integration of artificial intelligence (AI) in this field. The paper outlines various methodologies, including robotics, capture gloves, computer vision, neural networks, Hidden Markov Models (HMMs), 3D animation, Natural Language Processing (NLP), notation systems, and ongoing research projects. These approaches aim to bridge the communication gap between the deaf community and the hearing world by translating sign language into written or spoken language. The paper underscores the potential applications, such as educational tools and communication services, while emphasizing the significant impact of AI-driven advancements on deaf education and accessibility. [8]

**PAPER 9: Sign Language Semantic Translation System using Ontology and Deep Learning**

The paper introduces a Sign Language Semantic Translation System that utilizes Ontology and Deep Learning to translate sign language gestures into natural language text. It combines a Multi Sign Language Ontology (MSLO) with Convolutional Neural Networks (CNNs) to achieve semantic translation. The MSLO addresses challenges in sign language recognition, while CNNs are employed for image recognition. The paper presents the system's architecture, including the CNN layers and the semantic layer that connects with the ontology. It employs a dataset of Arabic sign language gestures, achieving promising results. The study highlights the potential of combining ontology and deep learning to enhance sign language translation and discusses future prospects, such as extending the system to handle dynamic gestures and developing real-time applications. [9]

**PAPER 10: Comparative Analysis On Sign Language Recognition System**

The research paper titled "Comparative Analysis On Sign Language Recognition System" addresses the issue of communication barriers between hearing-impaired and normal individuals, with a specific focus on Indian Sign Language (ISL). The methodology involves providing an introduction to the problem, presenting background information on sign language recognition, categorizing sign language systems, discussing related work and the accuracy of various methods, exploring classification algorithms like CNN, ANN, PNN, and KNN, identifying challenges, and suggesting future directions. The paper concludes by emphasizing the importance of improving real-time sign language recognition and vision-based approaches to enhance communication for the hearing-impaired community.[10]

**PAPER 11: Indian Sign Language Recognition using Convolutional Neural Network**

The proposed methodology for Indian Sign Language (ISL) recognition using Convolutional Neural Networks (CNNs) is a comprehensive approach aimed at improving communication for individuals with hearing disabilities. It begins by capturing hand gesture images and proceeds with essential steps such as segmentation, feature extraction, and CNN-based recognition. The CNN model is meticulously trained, resulting in a high accuracy rate of approximately 95%. The paper concludes by emphasizing the significance of the Sign Language Recognition (SLR) system in bridging communication gaps and outlines future possibilities for enhancing the system's image processing capabilities and expanding its translation functionalities. This methodology showcases the potential of computer vision and deep learning to significantly contribute to sign language recognition and communication accessibility.[11]

# CHAPTER 3

# SYSTEM ANALYSIS

* 1. **EXISTING SYSTEM**

In recent decades, due to computer software and hardware technologies of continuous innovation and breakthrough, the social life and information technology have a very close relationship in the twenty-first century. In the future, especially the interfaces of consumer electronics products (e.g. smart phones, games and infota1inment systems) will have more and more functions and be complex. How to develop a convenient human-machine Interface (Human Machine Interaction/Interface, HMI) for each consumer electronics product has become an important issue. The traditional electronic input devices, such as mouse, keyboard, and joystick are still the most common interaction way. However, it does not mean that these devices are the most convenient and natural input devices for most users. Since ancient times, gestures are a major way for communication and interaction between people.

People can easily express the idea by gestures before the invention of language. Nowadays, gestures still are naturally used by many people and especially are the most major and nature interaction way for deaf people. In recent years, the gesture control technique has become a new developmental trend for many human based electronics products, such as computers, televisions, and games. This technique let people can control these classifiers selection are a major issue in most researches. The third stage is to analyze sequential gestures to identify users’ instructs or behaviors.

### PROPOSED SYSTEM

The proposed methodology for image processing using Python and TensorFlow for sign language interpretation is a detailed plan that outlines the systematic development of a system aimed at facilitating communication for deaf and hard-of-hearing individuals. It involves a series of well-defined steps to collect, process, and interpret sign language gestures effectively.

The initial phase of data collection is fundamental. It involves the acquisition of a diverse and extensive dataset that encompasses a wide range of sign language gestures, phrases, and sentences. Native sign language users are involved to ensure authenticity. The dataset should represent various sign languages and dialects to cater to linguistic diversity. Real-world scenarios, characterized by different lighting conditions and backgrounds, are considered during data collection to ensure the system's adaptability to varying environments.

Following data collection, the data preprocessing stage is essential to prepare the collected images for model training. It involves cleaning and standardizing the images to remove noise and inconsistencies. Additionally, the dataset is augmented by applying various transformations like rotation, scaling, and the introduction of noise. This augmentation aims to make the model more robust and better prepared to handle real-world variability.

The subsequent step is data splitting to segregate the dataset into training, validation, and test sets. This separation allows for rigorous model training and evaluation. The training set is used to teach the model, the validation set helps fine-tune the model, and the test set assesses the system's real-world performance.

Model development is at the core of the methodology. Model selection and training\*\* involves choosing an appropriate pre-trained convolutional neural network (CNN) architecture for image recognition. Common choices include Inception, ResNet, or MobileNet. These architectures provide a strong foundation. However, modifications are required to adapt the model to sign language gesture recognition, specifically adjusting the top layers to match the number of sign language classes. TensorFlow and Keras are the tools of choice for model training. Transfer learning, which leverages the pre-trained model's knowledge, is employed to expedite the training process.

Model evaluation is a pivotal step to ensure the model's performance is optimal. Metrics like accuracy, precision, recall, and F1 score are used to gauge the model's accuracy and effectiveness in recognizing sign language gestures. This evaluation allows for fine-tuning of hyperparameters, model architecture, or data augmentation strategies to enhance accuracy and robustness.

After model development and evaluation, real-time image capture capabilities are implemented. This includes the use of cameras or webcams as input devices, allowing the system to capture live sign language gestures. These captured frames are then processed in real-time by the trained model for sign language interpretation.

To make the system accessible, a user-friendly interface is developed using Python libraries like Tkinter or web frameworks such as Flask or Django. The interface is carefully designed to cater to both sign language users and those unfamiliar with sign language.

Continuous improvement is a key principle of the methodology. Feedback and continuous improvement involve gathering feedback from users, particularly from the deaf and hard-of-hearing community, to iteratively enhance the model's accuracy and usability. This feedback loop ensures that the system evolves to better serve its intended users.

The deployment phase brings the system to users. It involves making the image processing model and user interface accessible through platforms like desktop applications or web applications. The aim is to ensure that the system is easy to use for both sign language users and non-users.

Ethical considerations are woven throughout the methodology. It addresses privacy concerns associated with image capture and storage, emphasizing the responsible handling of user data. Furthermore, the system is designed with cultural and linguistic diversity in mind, respecting the rich tapestry of sign language users.

In summary, this proposed methodology provides a structured and comprehensive approach to developing an image processing system for sign language interpretation using Python and TensorFlow. Its ultimate goal is to break down communication barriers and make communication more accessible for the deaf and hard-of-hearing community across a wide range of settings.

### ADVANTAGES

* + - * **Reduce external Interface** The Advantage of System is to Reduce External Interface like Mouse and Keyboard.
      * **High Portability** The proposed System reduce the working of external interface

### REQUIREMENT ANALYSIS ANDSPECIFICATION

* + 1. **HARDWAREREQUIREMENTS**
       - Desktop/Laptop
       - GBRAM
       - Keyboard
       - Mouse
       - Windows 7+
       - Pentium IV+

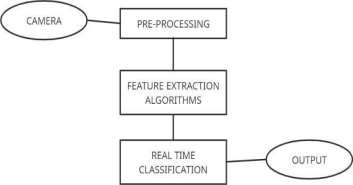
### SOFTWAREREQUIREMENTS

* + - * Python
      * IDLE
      * Anaconda
      * Jupyter Notebook

### FUNCTIONAL REQUIREMENTS

Hand gesture recognition system can be divided into following modules

* + - * Preprocessing
      * Feature extraction of the processed image
      * Real time classification



### Fig.no.3.3.1- SYSTEM IMPLEMENTATION

Value measure intensity or brightness. This is well enough to choose single color but it ignores complexity of color appearance. It trade off computation speed mean computationally expensive and perceptual relevance.

### PRE-PROCESSING

Like many other pattern recognition tasks, pre-processing is necessary for enhancing robustness and recognition accuracy. The preprocessing prepares the image sequence for the recognition, so before calculating the diagonal Sum and other algorithms, a pre-processing step is performed to get the appropriate image, which is required for real time classification. So it consists of some steps. The net effect of this processing is to extract the hand only from the given input because once the hand is detected from the given input it can be recognized easily.

So pre-processing step mainly consists of following tasks –

* + - Removal of Back ground
    - Conversion from RGB to binary
    - Hand Detection

### REMOVAL OF BACKGROUND

In identifying the background that greatly affects the results of hand detection decided to remove it. For this I have written own code in spite of using any built-in ones.



### Before After

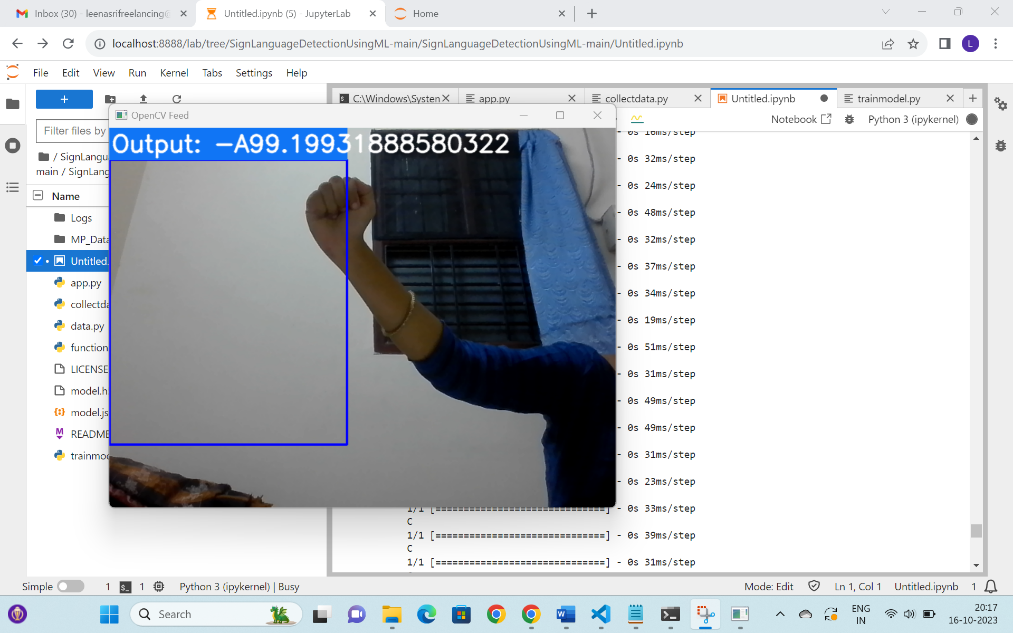
**Fig.no.3.3.2 - REMOVAL OF BACKGROUND**

### CONVERSION FROM RGB TO BINARY

All algorithms accept an input in RGB form and then convert it into binary format in order to provide ease in recognizing any gesture and also retaining the luminance factor in an image.

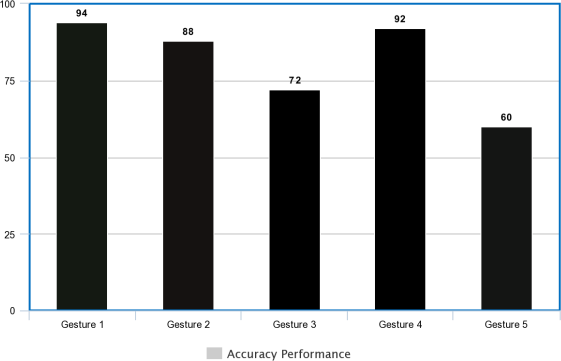
### HAND DETECTION

Image could have more than one skin area but in requirement only hand for further process. For this i choose criteria image labeling which is following- **Labeling** To define how many skin regions that we have in image is by labelling all skin regions. Label is basically an integer value have 8 connecting objects in order to label all skin area pixel. If object had label then mark current pixel with label if not then use new label with new integer value. After counting all labelled region (segmented image) I sort all them into ascending order with maximum value and choose the area have maximum value which I interested because I assume that hand region in bigger part of image. To separate that region which looked for, create new image that have one in positions where the label occurs and others set to zero.



### Before After

**Fig.no.3.3.3 - LABELING SKIN REGION**



### Fig.no.3.3.4 – ACCURACY PERFORMANCE

* 1. **TECHNOLOGY STACK**

### PYTHON

Python is a general-purpose interpreted, interactive, object-oriented, and high- level programming language. It was created by Guido van Rossum during 1985- 1990. Like Perl, Python source code is also available under the General Public License (GPL). This tutorial gives enough understanding on Python programming language.

Python is a high-level, interpreted, interactive and object- oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages.

**Python is Interpreted** − Python is processed at runtime by the interpreter. No do not need to compile the program before executing it. This is similar to PERL and PHP.

**Python is Object-Oriented** − Python supports Object-Oriented style or technique of programming that encapsulates code within objects.

**Python is a Beginner Language** − Python is a great language for the beginner- level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

### HISTORY OF PYTHON

Python was developed by Guido van Rossum in the late eighties and early nineties at the National Research Institute for Mathematics and Computer Science in the Netherlands.

Python is derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, Small Talk, and UNIX shell and other scripting languages.

Python is copyrighted. Like Perl, Python source code is now available under the GNU General Public License(GPL).

Python is now maintained by a core development team at the institute, although Guido van Rossum still holds a vital role in directing its progress.

### PYTHON FEATURES

Python features include–

**Easy-to-learn** − Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.

**Easy-to-read** − Python code is more clearly defined and visible to the yes.

**Easy-to-maintain** − Python source code is fairly easy-to-maintain.

**A broad standard library** − Python bulk of the library is very portable and

cross-platform compatible on UNIX, Windows, and Macintosh.

**Interactive Mode** − Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.

**Portable** − Python can run on a wide variety of hardware platforms and has the same interface on all platforms.

**Extendable** –The low-level modules can be added to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.

**Databases** − Python provides interfaces to all major commercial databases.

**GUI Programming** − Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.

**Scalable** − Python provides a better structure and support for large programs than shell scripting.

Apart from the above-mentioned features, Python has a big list of good features, few are listed below −

It supports functional and structured programming methods as well as OOP.

It can be used as a scripting language or can be compiled to byte-code for building large applications.

It provides very high-level dynamic data types and supports dynamic type

checking.

It supports automatic garbage collection.

It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.

### ANACONDA TOOL

Anaconda is a FREE enterprise-ready Python distribution for data analytics, processing, and scientific computing. Anaconda comes with Python 2.7 or Python3.4 and 100+ cross-platforms tested and optimized Python packages. All of the usual Python ecosystem tools work with Anaconda. AdditionallY Anaconda can create custom environments that mix and match different Python versions (2.6, 2.7,3.3 or 3.4) and other packages into isolated environments and easily switch between them using conda, in this project with innovative multi-platform package manager for Python and other languages.

### Anaconda Navigator

Anaconda Navigator is a desktop **graphical user interface (GUI)** included in Anaconda® distribution that allows to launch applications and easily manage conda packages, environments and channels without using

command-line commands. Navigator can search for packages on Anaconda Cloud

It is available for Windows, macOS and Linux Using Python in Anaconda Many people write Python code using a text editor like Emacs or Vim. Others prefer to use an IDE like Spyder, Wing IDE, PyCharm or Python Tools for Visual Studio. Spyder is a great free IDE that is included with Anaconda. To start Spyder, type the name spyder in a terminal or at the Command Prompt.

The Python 2.7 version of Anaconda also includes a graphical Launcher application that enables to start IPython Notebook, IPythonQTConsole, and

Spyder with a single click. On Mac, double click the Launcher.app, found in the system ~/anaconda directory (or wherever the Anaconda installed). On Windows, on Start Menu there will be a Launcher. The Start Menu also has an Anaconda Command Prompt that, regardless of system and install settings, will launch the Python interpreter installed via Anaconda. This is particularly useful for troubleshooting, if the user have multiple Python installations on the system.

### CONDA

Conda is an open source package management system and environment management system that runs on Windows, macOS and Linux. Conda quickly installs, runs and updates packages and their dependencies. Conda easily creates, saves, loads and switches between environments on the local computer. It was created for Python programs, but it can package and distribute software for any language.

Conda as a package manager helps to find and install packages. If in need a package that requires a different version of Python, do not need to switch to a different environment manager, because conda is also an environment manager. With just a few commands, it can set up a totally separate environment to run that different version of Python, while continuing to run the usual version of Python in user system normal environment.

### SPYDER

Spyder’s text editor is a multi-language editor with features such as syntax coloring, code analysis (real-time code analysis powered by pyflakes and advanced code analysis using pylint), introspection capabilities such as code completion, calltips and go-to-definition features (powered by rope), function/class browser, horizontal/vertical splitting features, etc.Spyder is the Scientific Python Development Environment.

Spyder is a powerful interactive development environment for the Python language with advanced editing, interactive etesting, debugging and Introspection features; and a numerical computing environment thanks to the support of IPython (enhanced interactive Python interpreter) and popular Python libraries such as NumPy (linear algebra), SciPy (signal and image processing) ormatplotlib (interactive 2D/3Dplotting).

### JUPYTER NOTEBOOK

Notebook documents (or “notebooks”, all lower case) are documents produced by the Jupyter Notebook App, which contain both computer code (e.g. python)and rich text elements (paragraph, equations, figures, links, etc…). Notebook documents are both human-readable documents containing the analysis description and the results (figures, tables, etc..) as well as executable documents which can be run to perform data analysis.

### JUPYTER NOTEBOOK APP

The Jupyter Notebook App is a server-client application that allows editing and running notebook documents via a web browser. The Jupyter Notebook App can be executed on a local desktop requiring no internet access (as described in this document) or can be installed on a remote server and accessed through the internet.

In addition to displaying/editing/running notebook documents, the Jupyter Notebook App has a “Dashboard” (Notebook Dashboard), a “control panel”showing local files and allowing to open notebook documents or shutting down their kernels.

### KERNEL

A notebook kernel is a “computational engine” that executes the code contained in a Notebook document. The ipython kernel, referenced in this guide, executes python code.

Kernels for many other languages exist (official kernels).When Notebook document is opened, the associated kernel is automatically launched. When the notebook is executed (either cell-by-cell or with menu Cell Depending on the type of computations, the kernel may consume significant CPU and RAM. Note that the RAM is not released until the kernel is shut-down Notebook Dashboard The Notebook Dashboard is the component which is shown first when Jupyter Notebook App launched.

The Notebook Dashboard is mainly used to open notebook documents, and to manage the running kernels (visualize and shutdown).

The Notebook Dashboard has other features similar to a file manager, namely navigating folders and renaming/deleting files.

In this case, notebook or notebook documents denote documents that contain both code and rich text elements, such as figures, links, and equations. Because of the mix of code and text elements, these documents are the ideal place to bring together an analysis description and its results as well as they can be executed perform the data analysis in real time. These documents are produced by the Jupyter Notebook App.

For now, it is should be know that Jupyter is a loose acronym meaning Julia, Python, and R. These programming languages were the first target languages of the Jupyter application, but nowadays, the notebook technology also supports many other languages.

# CHAPTER 4

# SYSTEM DESIGN

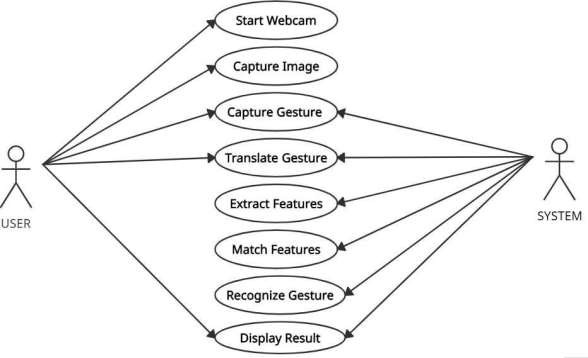
* 1. **UML DIAGRAMS**

Unified Modeling Language (UML) is a general purpose modelling language. The main aim of UML is to define a standard way to visualize the way a system has been designed. It is quite similar to blueprints used in other fields of engineering.

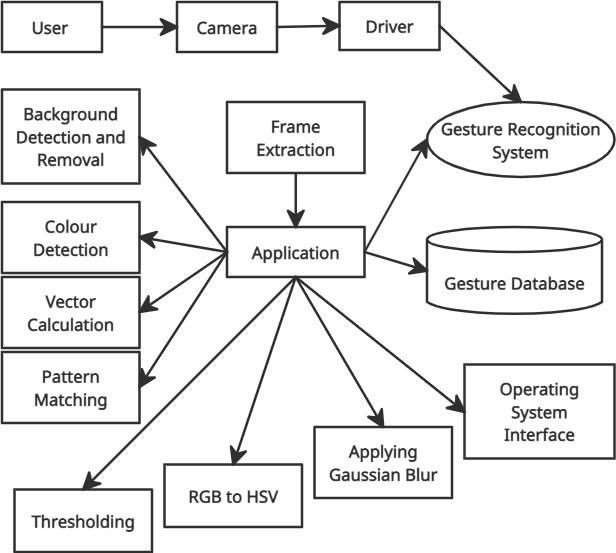
### USE CASE DIAGRAM

A use case diagram at its simplest is a representation of a user's interaction

### Fig.no.4.1.1 - USE CASE DIAGRAM

with the system that shows the relationship between the user and the different use cases in which the user is involved. A use case diagram can identify the different types of users of a system and the different use cases and will often be accompanied by other types of diagrams as well.

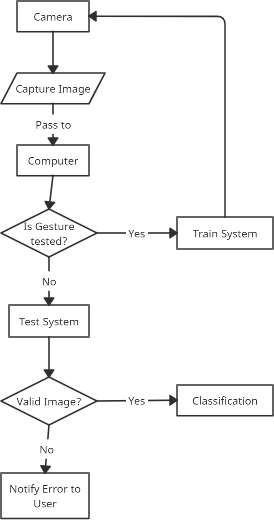
**4.1.2 DATA FLOW DIAGRAM**



### Fig.no.4.1.2 - DATA FLOWDIAGRAM

* + 1. **FLOWCHART DIAGRAM**

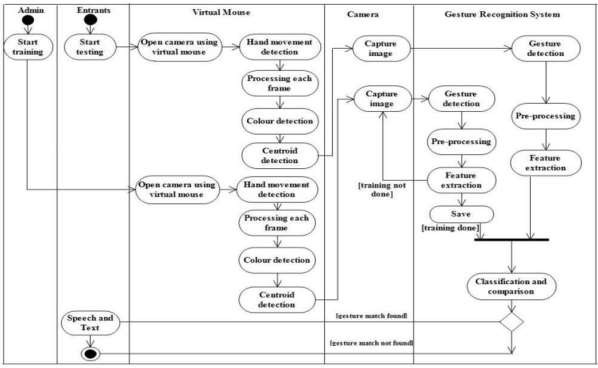
A diagram of the sequence of movements or actions of people or things involved in a complex system or activity.



### Fig.no.4.1.3 - FLOWCHART DIAGRAM

* + 1. **ACTIVITY DIAGRAM**

A graphical representation of an executed set of procedural system activities and considered a state chart diagram variation. Activity diagrams describe parallel and conditional activities, use cases and system functions at a detailed level.

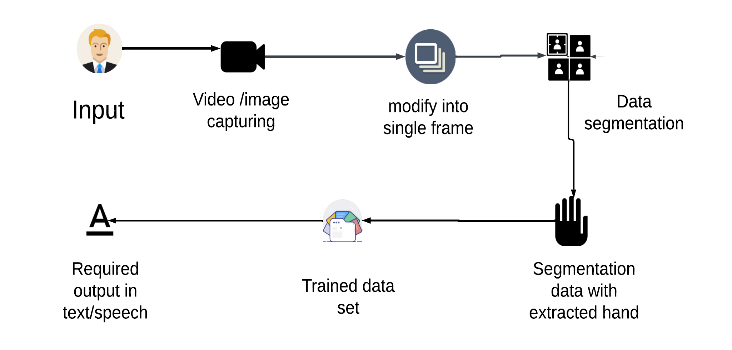


**Fig.no.4.1.4 - ACTIVITY DIAGRAM**

# CHAPTER 5

# SYSTEM ARCHITECTURE

* 1. **ARCHITECTURE OVERVIEW**



### Fig.no.5.1 ARCHITECTURE DIAGRAM

The figure above represents the System Architecture of the project that basically show each component of the system , how the system works, and the flow of the system and so on. Images that are taken from the web camera goes under pre- processing stages to enhance the feature of an image. Then there is a removal of object and background from the images which later convert into binary form. Feature extraction and reorganization helps to match the images that is stored in database and get the desired output in the form of text and converts that text to speech.

### MODULES DESIGN SPECIFICATION

* + - Dataset collection
    - Dataset pre-processing
    - Machine Learning Training
    - Capture Real time video
    - Eliminate Background
    - Identify Gesture

### Dataset Collection

Dataset collection involves collecting or creating test dataset for gesture and respective terminology. This is very exhaustive task as in need to collect more samples for gesture. Dataset are collected from kaggle website for the project in scope.

### Dataset Pre-Processing

Dataset preprocessing involves removing noise in images and converting to numpy arrays

### Machine Learning

Machine learning involves training the system and classification using CNN algorithms.

### Capture Real Time Video & Gesture Detection

By using the open cv to capture and process the video.

After capturing video then pre-process the frames and fit it in model to get the gesture right. **Background Elimination**

And by finding the contours to detect edges of the main object and create a mask with numpy zeros for the background and then combine the mask and the image using bitwise and operator.

**5.2 PROGRAM DESIGN LANGUAGE**

**ALGORITHM**

Step 1 - Reading image from camera and applying pre-processing techniques like gamma correction, blurring.

Step 2 - Hand Segmentation using background subtraction algorithm.

Step 3 - Hand detection using thresholding and dilation

Step 4 - Finding contours of hand for getting shape of hand.

Step 5 - Finding contour area, convex hull, hull area, solidity.

Step 6 - Also find the angle between two fingers and aspect ratio of hand.

Step 7 - Finding the defects of hand using convex hull.

Step 8 - Finally classifying using solidity, aspect ratio, convex defects and angle.

Step 9 - if image (sign) ==database image, speak the meaning of that sign Repeat.

Step 10 - END.

**CHAPTER 6**

**SYSTEM IMPLEMENTATION**

# Program

# Collect.py

# import os

# import cv2

# cap=cv2.VideoCapture(0)

# directory='Image/'

# while True:

# \_,frame=cap.read()

# count = {

# 'a': len(os.listdir(directory+"/A")),

# 'b': len(os.listdir(directory+"/B")),

# 'c': len(os.listdir(directory+"/C")),

# 'd': len(os.listdir(directory+"/D")),

# 'e': len(os.listdir(directory+"/E")),

# 'f': len(os.listdir(directory+"/F")),

# 'g': len(os.listdir(directory+"/G")),

# 'h': len(os.listdir(directory+"/H")),

# 'i': len(os.listdir(directory+"/I")),

# 'j': len(os.listdir(directory+"/J")),

# 'k': len(os.listdir(directory+"/K")),

# 'l': len(os.listdir(directory+"/L")),

# 'm': len(os.listdir(directory+"/M")),

# 'n': len(os.listdir(directory+"/N")),

# 'o': len(os.listdir(directory+"/O")),

# 'p': len(os.listdir(directory+"/P")),

# 'q': len(os.listdir(directory+"/Q")),

# 'r': len(os.listdir(directory+"/R")),

# 's': len(os.listdir(directory+"/S")),

# 't': len(os.listdir(directory+"/T")),

# 'u': len(os.listdir(directory+"/U")),

# 'v': len(os.listdir(directory+"/V")),

# 'w': len(os.listdir(directory+"/W")),

# 'x': len(os.listdir(directory+"/X")),

# 'y': len(os.listdir(directory+"/Y")),

# 'z': len(os.listdir(directory+"/Z"))

# }

# # cv2.putText(frame, "a : "+str(count['a']), (10, 100), cv2.FONT\_HERSHEY\_PLAIN, 1, (0,255,255), 1)

# # cv2.putText(frame, "b : "+str(count['b']), (10, 110), cv2.FONT\_HERSHEY\_PLAIN, 1, (0,255,255), 1)

# # cv2.putText(frame, "c : "+str(count['c']), (10, 120), cv2.FONT\_HERSHEY\_PLAIN, 1, (0,255,255), 1)

# # cv2.putText(frame, "d : "+str(count['d']), (10, 130), cv2.FONT\_HERSHEY\_PLAIN, 1, (0,255,255), 1)

# # cv2.putText(frame, "e : "+str(count['e']), (10, 140), cv2.FONT\_HERSHEY\_PLAIN, 1, (0,255,255), 1)

# # cv2.putText(frame, "f : "+str(count['f']), (10, 150), cv2.FONT\_HERSHEY\_PLAIN, 1, (0,255,255), 1)

# # cv2.putText(frame, "g : "+str(count['g']), (10, 160), cv2.FONT\_HERSHEY\_PLAIN, 1, (0,255,255), 1)

# # cv2.putText(frame, "h : "+str(count['h']), (10, 170), cv2.FONT\_HERSHEY\_PLAIN, 1, (0,255,255), 1)

# # cv2.putText(frame, "i : "+str(count['i']), (10, 180), cv2.FONT\_HERSHEY\_PLAIN, 1, (0,255,255), 1)

# # cv2.putText(frame, "k : "+str(count['k']), (10, 190), cv2.FONT\_HERSHEY\_PLAIN, 1, (0,255,255), 1)

# # cv2.putText(frame, "l : "+str(count['l']), (10, 200), cv2.FONT\_HERSHEY\_PLAIN, 1, (0,255,255), 1)

# # cv2.putText(frame, "m : "+str(count['m']), (10, 210), cv2.FONT\_HERSHEY\_PLAIN, 1, (0,255,255), 1)

# # cv2.putText(frame, "n : "+str(count['n']), (10, 220), cv2.FONT\_HERSHEY\_PLAIN, 1, (0,255,255), 1)

# # cv2.putText(frame, "o : "+str(count['o']), (10, 230), cv2.FONT\_HERSHEY\_PLAIN, 1, (0,255,255), 1)

# # cv2.putText(frame, "p : "+str(count['p']), (10, 240), cv2.FONT\_HERSHEY\_PLAIN, 1, (0,255,255), 1)

# # cv2.putText(frame, "q : "+str(count['q']), (10, 250), cv2.FONT\_HERSHEY\_PLAIN, 1, (0,255,255), 1)

# # cv2.putText(frame, "r : "+str(count['r']), (10, 260), cv2.FONT\_HERSHEY\_PLAIN, 1, (0,255,255), 1)

# # cv2.putText(frame, "s : "+str(count['s']), (10, 270), cv2.FONT\_HERSHEY\_PLAIN, 1, (0,255,255), 1)

# # cv2.putText(frame, "t : "+str(count['t']), (10, 280), cv2.FONT\_HERSHEY\_PLAIN, 1, (0,255,255), 1)

# # cv2.putText(frame, "u : "+str(count['u']), (10, 290), cv2.FONT\_HERSHEY\_PLAIN, 1, (0,255,255), 1)

# # cv2.putText(frame, "v : "+str(count['v']), (10, 300), cv2.FONT\_HERSHEY\_PLAIN, 1, (0,255,255), 1)

# # cv2.putText(frame, "w : "+str(count['w']), (10, 310), cv2.FONT\_HERSHEY\_PLAIN, 1, (0,255,255), 1)

# # cv2.putText(frame, "x : "+str(count['x']), (10, 320), cv2.FONT\_HERSHEY\_PLAIN, 1, (0,255,255), 1)

# # cv2.putText(frame, "y : "+str(count['y']), (10, 330), cv2.FONT\_HERSHEY\_PLAIN, 1, (0,255,255), 1)

# # cv2.putText(frame, "z : "+str(count['z']), (10, 340), cv2.FONT\_HERSHEY\_PLAIN, 1, (0,255,255), 1)

# row = frame.shape[1]

# col = frame.shape[0] cv2.rectangle(frame,(0,40),(300,400),(255,5,255),2)

# cv2.imshow("data",frame)

# cv2.imshow("ROI",frame[40:400,0:300])

# frame=frame[40:400,0:300]

# interrupt = cv2.waitKey(10)

# if interrupt & 0xFF == ord('a'): cv2.imwrite(directory+'A/'+str(count['a'])+'.png',frame)

# if interrupt & 0xFF == ord('b'): cv2.imwrite(directory+'B/'+str(count['b'])+'.png',frame)

# if interrupt & 0xFF == ord('c'): cv2.imwrite(directory+'C/'+str(count['c'])+'.png',frame)

# if interrupt & 0xFF == ord('d'): cv2.imwrite(directory+'D/'+str(count['d'])+'.png',frame)

# if interrupt & 0xFF == ord('e'): cv2.imwrite(directory+'E/'+str(count['e'])+'.png',frame)

# if interrupt & 0xFF == ord('f'): cv2.imwrite(directory+'F/'+str(count['f'])+'.png',frame)

# if interrupt & 0xFF == ord('g'): cv2.imwrite(directory+'G/'+str(count['g'])+'.png',frame)

# if interrupt & 0xFF == ord('h'): cv2.imwrite(directory+'H/'+str(count['h'])+'.png',frame)

# if interrupt & 0xFF == ord('i'): cv2.imwrite(directory+'I/'+str(count['i'])+'.png',frame)

# if interrupt & 0xFF == ord('j'): cv2.imwrite(directory+'J/'+str(count['j'])+'.png',frame)

# if interrupt & 0xFF == ord('k'): cv2.imwrite(directory+'K/'+str(count['k'])+'.png',frame)

# if interrupt & 0xFF == ord('l'): cv2.imwrite(directory+'L/'+str(count['l'])+'.png',frame)

# if interrupt & 0xFF == ord('m'): cv2.imwrite(directory+'M/'+str(count['m'])+'.png',frame)

# if interrupt & 0xFF == ord('n'): cv2.imwrite(directory+'N/'+str(count['n'])+'.png',frame)

# if interrupt & 0xFF == ord('o'): cv2.imwrite(directory+'O/'+str(count['o'])+'.png',frame)

# if interrupt & 0xFF == ord('p'): cv2.imwrite(directory+'P/'+str(count['p'])+'.png',frame)

# if interrupt & 0xFF == ord('q'):

# cv2.imwrite(directory+'Q/'+str(count['q'])+'.png',frame)

# if interrupt & 0xFF == ord('r'): cv2.imwrite(directory+'R/'+str(count['r'])+'.png',frame)

# if interrupt & 0xFF == ord('s'): cv2.imwrite(directory+'S/'+str(count['s'])+'.png',frame)

# if interrupt & 0xFF == ord('t'): cv2.imwrite(directory+'T/'+str(count['t'])+'.png',frame)

# if interrupt & 0xFF == ord('u'): cv2.imwrite(directory+'U/'+str(count['u'])+'.png',frame)

# if interrupt & 0xFF == ord('v'): cv2.imwrite(directory+'V/'+str(count['v'])+'.png',frame)

# if interrupt & 0xFF == ord('w'): cv2.imwrite(directory+'W/'+str(count['w'])+'.png',frame)

# if interrupt & 0xFF == ord('x'): cv2.imwrite(directory+'X/'+str(count['x'])+'.png',frame)

# if interrupt & 0xFF == ord('y'):

# cv2.imwrite(directory+'Y/'+str(count['y'])+'.png',frame)

# if interrupt & 0xFF == ord('z'): cv2.imwrite(directory+'Z/'+str(count['z'])+'.png',frame)

# cap.release()

# cv2.destroyAllWindows()

# Data.py

# from function import \*

# from time import sleep

# for action in actions:

# for sequence in range(no\_sequences):

# try:

# os.makedirs(os.path.join(DATA\_PATH, action, str(sequence)))

# except:

# pass

# # cap = cv2.VideoCapture(0)

# # Set mediapipe model

# with mp\_hands.Hands(

# model\_complexity=0,

# min\_detection\_confidence=0.5,

# min\_tracking\_confidence=0.5) as hands:

# # NEW LOOP

# # Loop through actions

# for action in actions:

# # Loop through sequences aka videos

# for sequence in range(no\_sequences):

# # Loop through video length aka sequence length

# for frame\_num in range(sequence\_length):

# # Read feed

# # ret, frame = cap.read()

# frame=cv2.imread('Image/{}/{}.png'.format(action,sequence))

# # frame=cv2.imread('{}{}.png'.format(action,sequence))

# # frame=cv2.cvtColor(frame,cv2.COLOR\_BGR2GRAY)

# # Make detections

# image, results = mediapipe\_detection(frame, hands)

# print(results)

# # Draw landmarks

# draw\_styled\_landmarks(image, results)

# # NEW Apply wait logic

# if frame\_num == 0:

# cv2.putText(image, 'STARTING COLLECTION', (120,200),

# cv2.FONT\_HERSHEY\_SIMPLEX, 1, (0,255, 0), 4, cv2.LINE\_AA)

# cv2.putText(image, 'Collecting frames for {} Video Number {}'.format(action, sequence), (15,12),

# cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 0, 255), 1, cv2.LINE\_AA)

# # Show to screen

# cv2.imshow('OpenCV Feed', image)

# cv2.waitKey(200)

# else:

# cv2.putText(image, 'Collecting frames for {} Video Number {}'.format(action, sequence), (15,12),

# cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 0, 255), 1, cv2.LINE\_AA)

# # Show to screen

# cv2.imshow('OpenCV Feed', image)

# # NEW Export keypoints

# keypoints = extract\_keypoints(results)

# npy\_path = os.path.join(DATA\_PATH, action, str(sequence), str(frame\_num))

# np.save(npy\_path, keypoints)

# # Break gracefully

# if cv2.waitKey(10) & 0xFF == ord('q'):

# break

# # cap.release()

# cv2.destroyAllWindows()

# Function.py

# #import dependency

# import cv2

# import numpy as np

# import os

# import mediapipe as mp

# mp\_drawing = mp.solutions.drawing\_utils

# mp\_drawing\_styles = mp.solutions.drawing\_styles

# mp\_hands = mp.solutions.hands

# def mediapipe\_detection(image, model):

# image = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB) # COLOR

# image.flags.writeable = False # Image is no longer writeable

# results = model.process(image) # Make prediction

# image.flags.writeable = True # Image is now writeable

# image = cv2.cvtColor(image, cv2.COLOR\_RGB2BGR) # COLOR COVERSION RGB 2 BGR

# return image, results

# def draw\_styled\_landmarks(image, results):

# if results.multi\_hand\_landmarks:

# for hand\_landmarks in results.multi\_hand\_landmarks:

# mp\_drawing.draw\_landmarks(

# image,

# hand\_landmarks,

# mp\_hands.HAND\_CONNECTIONS,

# mp\_drawing\_styles.get\_default\_hand\_landmarks\_style(),

# mp\_drawing\_styles.get\_default\_hand\_connections\_style())

# def extract\_keypoints(results):

# if results.multi\_hand\_landmarks:

# for hand\_landmarks in results.multi\_hand\_landmarks:

# rh = np.array([[res.x, res.y, res.z] for res in hand\_landmarks.landmark]).flatten() if hand\_landmarks else np.zeros(21\*3)

# return(np.concatenate([rh]))

# # Path for exported data, numpy arrays

# DATA\_PATH = os.path.join('MP\_Data')

# actions = np.array(['A','B','C'])

# no\_sequences = 30

# sequence\_length = 30

# app.py

# from function import \*

# from keras.utils import to\_categorical

# from keras.models import model\_from\_json

# from keras.layers import LSTM, Dense

# from keras.callbacks import TensorBoard

# json\_file = open("model.json", "r")

# model\_json = json\_file.read()

# json\_file.close()

# model = model\_from\_json(model\_json)

# model.load\_weights("model.h5")

# colors = []

# for i in range(0,20):

# colors.append((245,117,16))

# print(len(colors))

# def prob\_viz(res, actions, input\_frame, colors,threshold):

# output\_frame = input\_frame.copy()

# for num, prob in enumerate(res):

# cv2.rectangle(output\_frame, (0,60+num\*40), (int(prob\*100), 90+num\*40), colors[num], -1)

# cv2.putText(output\_frame, actions[num], (0, 85+num\*40), cv2.FONT\_HERSHEY\_SIMPLEX, 1, (255,255,255), 2, cv2.LINE\_AA)

# return output\_frame

# # 1. New detection variables

# sequence = []

# sentence = []

# accuracy=[]

# predictions = []

# threshold = 0.8

# cap = cv2.VideoCapture(0)

# # cap = cv2.VideoCapture("https://192.168.43.41:8080/video")

# # Set mediapipe model

# with mp\_hands.Hands(

# model\_complexity=0,

# min\_detection\_confidence=0.5,

# min\_tracking\_confidence=0.5) as hands:

# while cap.isOpened():

# # Read feed

# ret, frame = cap.read()

# # Make detections

# cropframe=frame[40:400,0:300]

# # print(frame.shape)

# frame=cv2.rectangle(frame,(0,40),(300,400),255,2)

# # frame=cv2.putText(frame,"Active Region",(75,25),cv2.FONT\_HERSHEY\_COMPLEX\_SMALL,2,255,2)

# image, results = mediapipe\_detection(cropframe, hands)

# # print(results)

# # Draw landmarks

# # draw\_styled\_landmarks(image, results)

# # 2. Prediction logic

# keypoints = extract\_keypoints(results)

# sequence.append(keypoints)

# sequence = sequence[-30:]

# try:

# if len(sequence) == 30:

# res = model.predict(np.expand\_dims(sequence, axis=0))[0]

# print(actions[np.argmax(res)])

# predictions.append(np.argmax(res))

# #3. Viz logic

# if np.unique(predictions[-10:])[0]==np.argmax(res):

# if res[np.argmax(res)] > threshold:

# if len(sentence) > 0:

# if actions[np.argmax(res)] != sentence[-1]: sentence.append(actions[np.argmax(res)]) accuracy.append(str(res[np.argmax(res)]\*100))

# else:

# sentence.append(actions[np.argmax(res)])

# accuracy.append(str(res[np.argmax(res)]\*100))

# if len(sentence) > 1:

# sentence = sentence[-1:]

# accuracy=accuracy[-1:]

# # Viz probabilities

# # frame = prob\_viz(res, actions, frame, colors,threshold)

# except Exception as e:

# # print(e)

# pass

# cv2.rectangle(frame, (0,0), (300, 40), (245, 117, 16), -1)

# cv2.putText(frame,"Output: -"+' '.join(sentence)+''.join(accuracy), (3,30),

# cv2.FONT\_HERSHEY\_SIMPLEX, 1, (255, 255, 255), 2, cv2.LINE\_AA)

# # Show to screen

# cv2.imshow('OpenCV Feed', frame)

# # Break gracefully

# if cv2.waitKey(10) & 0xFF == ord('q'):

# break

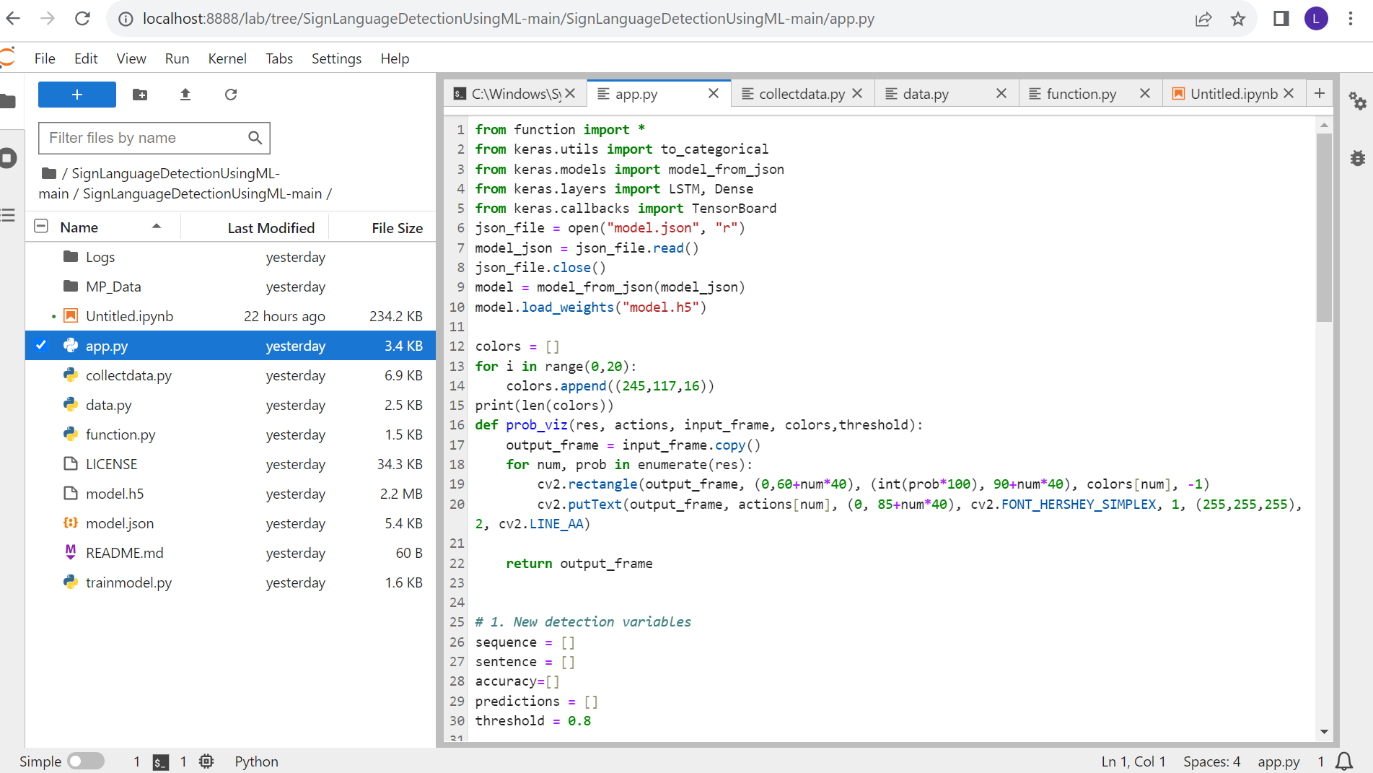
# cap.release()

# cv2.destroyAllWindows()

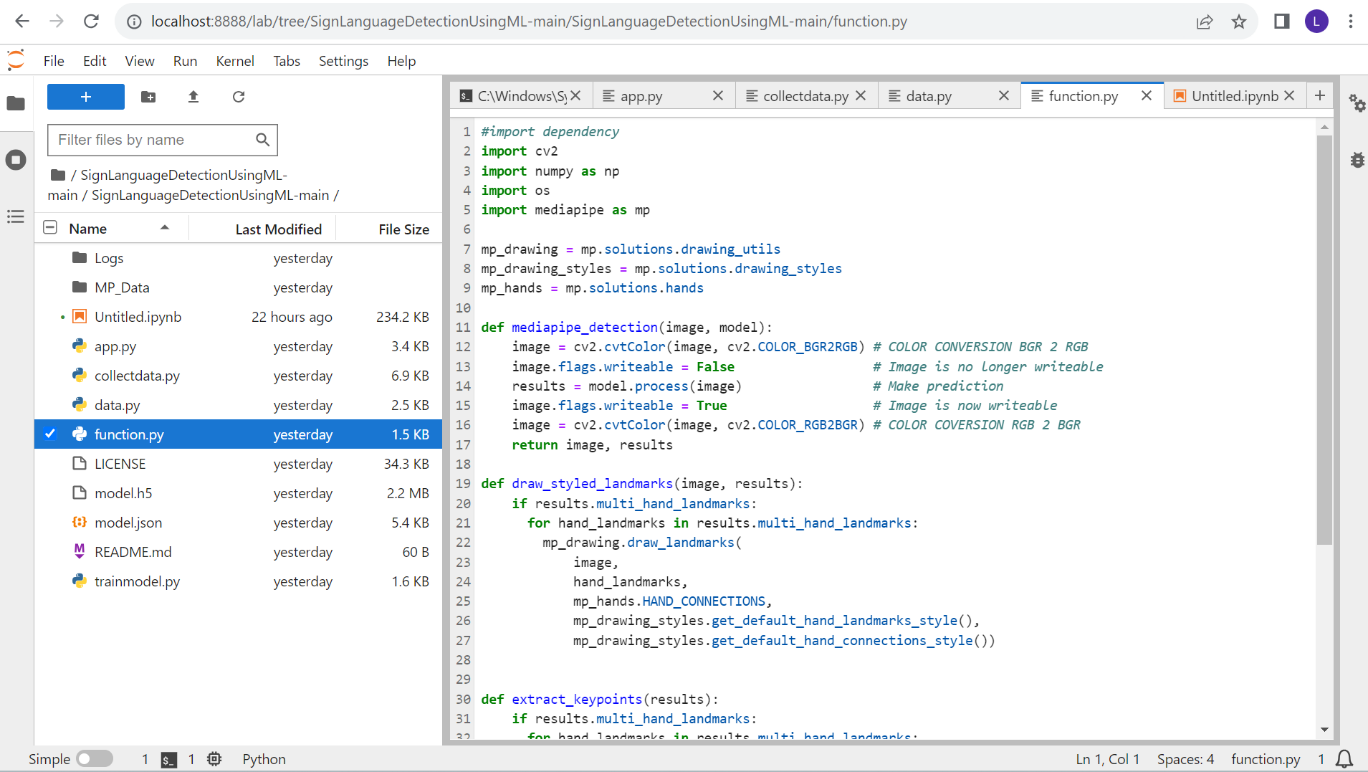
# CHAPTER 7

# SYSTEM TESTING

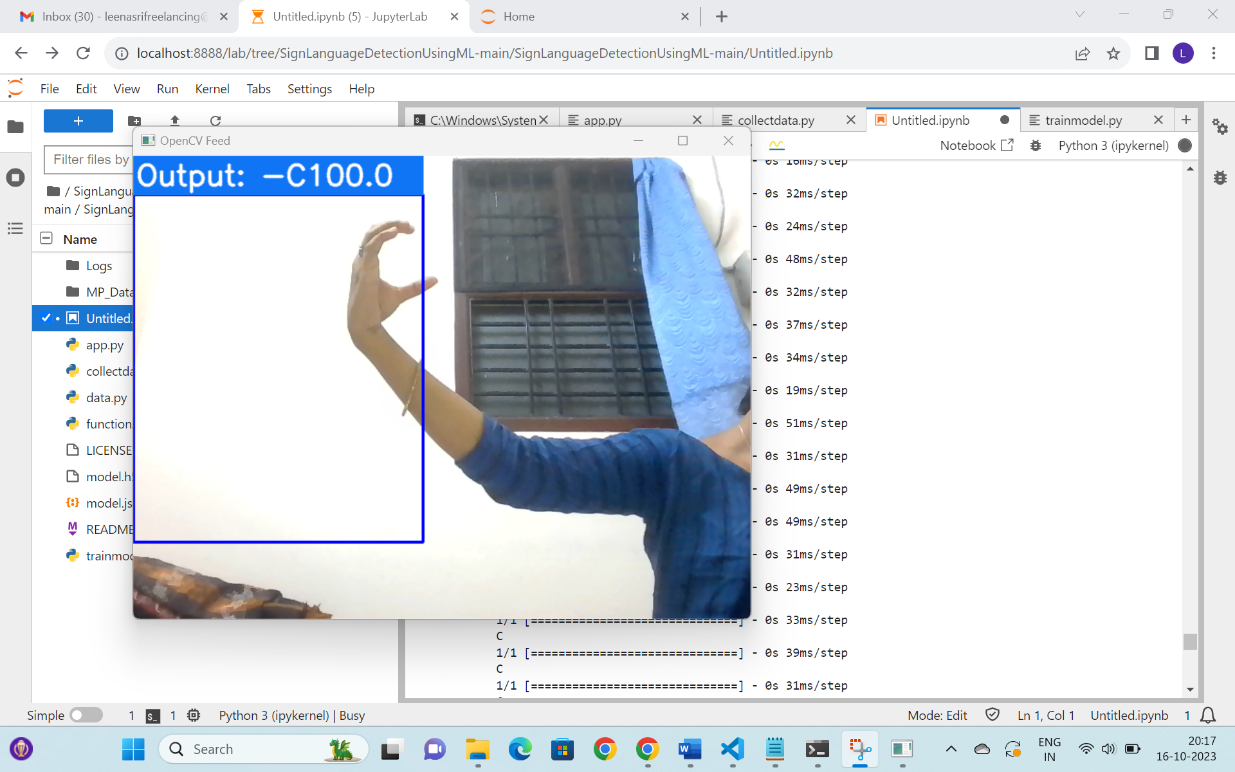
### TEST CASES & REPORTS / PERFORMANCE ANALYSIS



**Fig.no.7.3.1 - System Training Coding**



### Fig.no.7.3.2 - Functions



**Fig.no.7.3.3 - Sample Output Screen Shot**

**7.1 UNIT TESTING**

Unit testing is a type of software testing where individual units or components of a software are tested. The purpose is to validate that each unit of the software code performs as expected. Unit Testing is done during the development (coding phase) of an application by the developers. Unit Tests isolate a section of code and verify its correctness. A unit may be an individual function, method, procedure, module, or object. In SDLC, STLC, V Model, Unit testing is first level of testing done before integration testing. Unit testing is a White Box testing technique that is usually performed by the developer. Though, in a practical world due to time crunch or reluctance of developers to tests, QA engineers also do unit testing.

**7.2 INTEGRATION TESTING**

Integration testing is a technique for constructing the software architecture and conducting tests to uncover errors with interface. The objective of testing was to crosscheck for components fully functional or not according to design. Thus, by integrating all the unit components and if the system worked as a whole properly or not. The information flows between the components were checked once again.

# CHAPTER 8

# CONCLUSION

* 1. **CONCLUSION AND FUTURE ENHANCEMENTS**

### CONCLUSION

This chapter summarizes my work at every stage of the project. At the time I started my thesis, I had a brief idea of how I will bring it from a topic on the paper to a real product. Due to knowledge of Computer Vision and Biometric subjects I had background in the image-processing field but not at expert level but my constant effort helped me to go through and succeed eventually.

As required in every project, research is of utmost importance. So, by spending pretty much time in going through the background literature. And looked at various approaches of doing this thesis and developed four different methods: Row vector algorithm, Edging and row vector passing algorithm, Mean and standard deviation of edged image and Diagonal sum algorithm.

Each of these algorithms was tried with neural networks and have higher performance rate in the ascending order respectively.

The first limitation that was discovered in all the algorithms used with neural networks was that their performance depended on the amount of training dataset provided. The system worked efficiently after being trained by a larger dataset as compared to a smaller dataset.

The Row vector algorithm used initially was a very vague approach adopted for classification as it was found through experiments that the row vectors of two different images could happen to be the same.

In the edging and row vector-passing algorithm, the edging parameter was introduced in addition to the row vector to improve the gesture classification

accuracy but it was found that due to self-shadowing effect found in edges, the detection rate was not sufficiently improved.

The next parameters tried for classification were mean and standard deviation. They also failed to give satisfactory results (i.e. above 60%) but still they were among the best parameters used for detection with neural networks.

Due to the unusual behavior of neural network with all the mentioned parameters, the diagonal sum algorithm was finally implemented in real time. The system was tested with 60 pictures and the detection rate was found to be 86%. The strengths and weaknesses of gesture recognition using diagonal sum have been presented and discussed. With the implemented system serving as an extendible foundation for future research, extensions to the current system have been proposed

### FUTUREENHANCEMENTS

The system could also be made smart to be trained for only one or two gestures rather than all and then made ready for testing. This will require only a few changes in the current interface code, which were not performed due to the shortage of time.

One time training constraint for real time system can be removed if the algorithm is made efficient to work with all skin types and light conditions which seems impossible by now altogether. Framing with COG (Centre of gravity) to control orientation factor could make this system more perfect for real application.

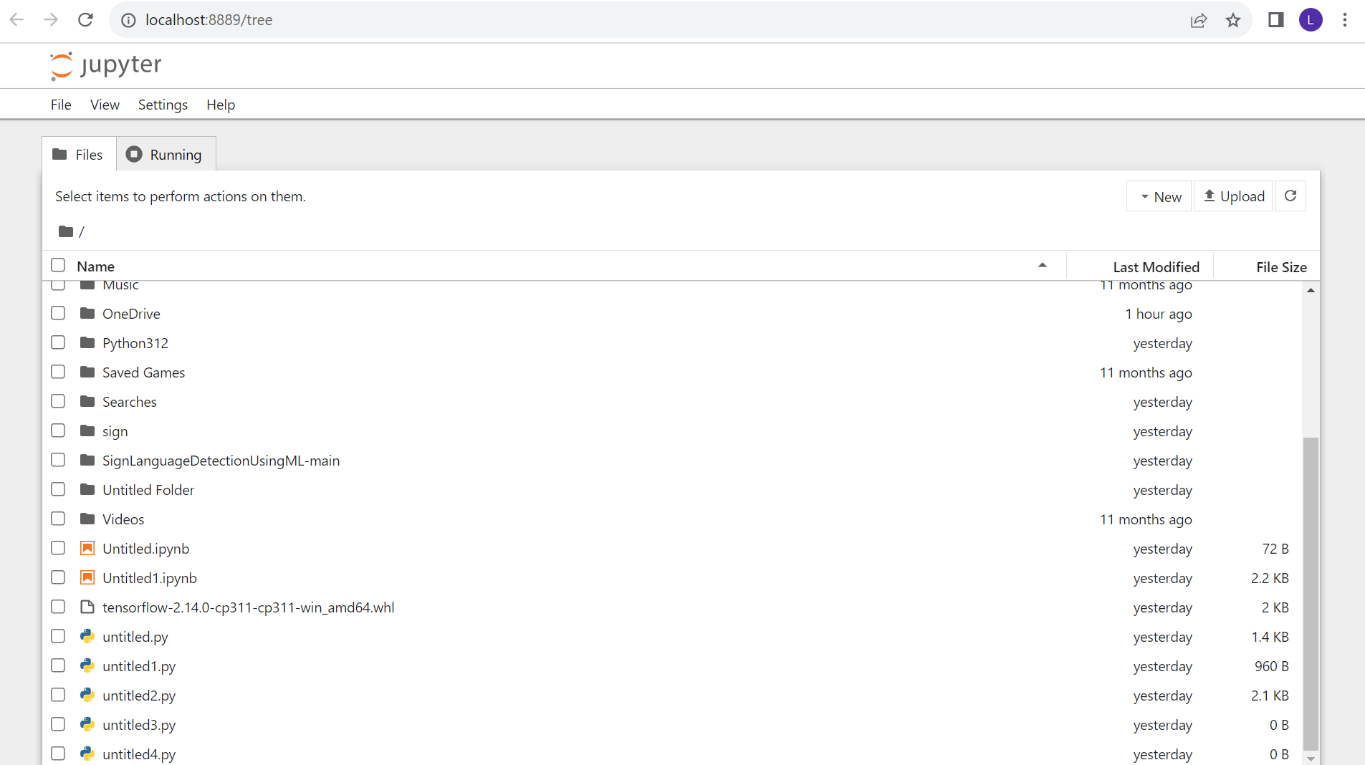
The system’s speed for preprocessing could be improved if the code is developedinVC/VC.Net

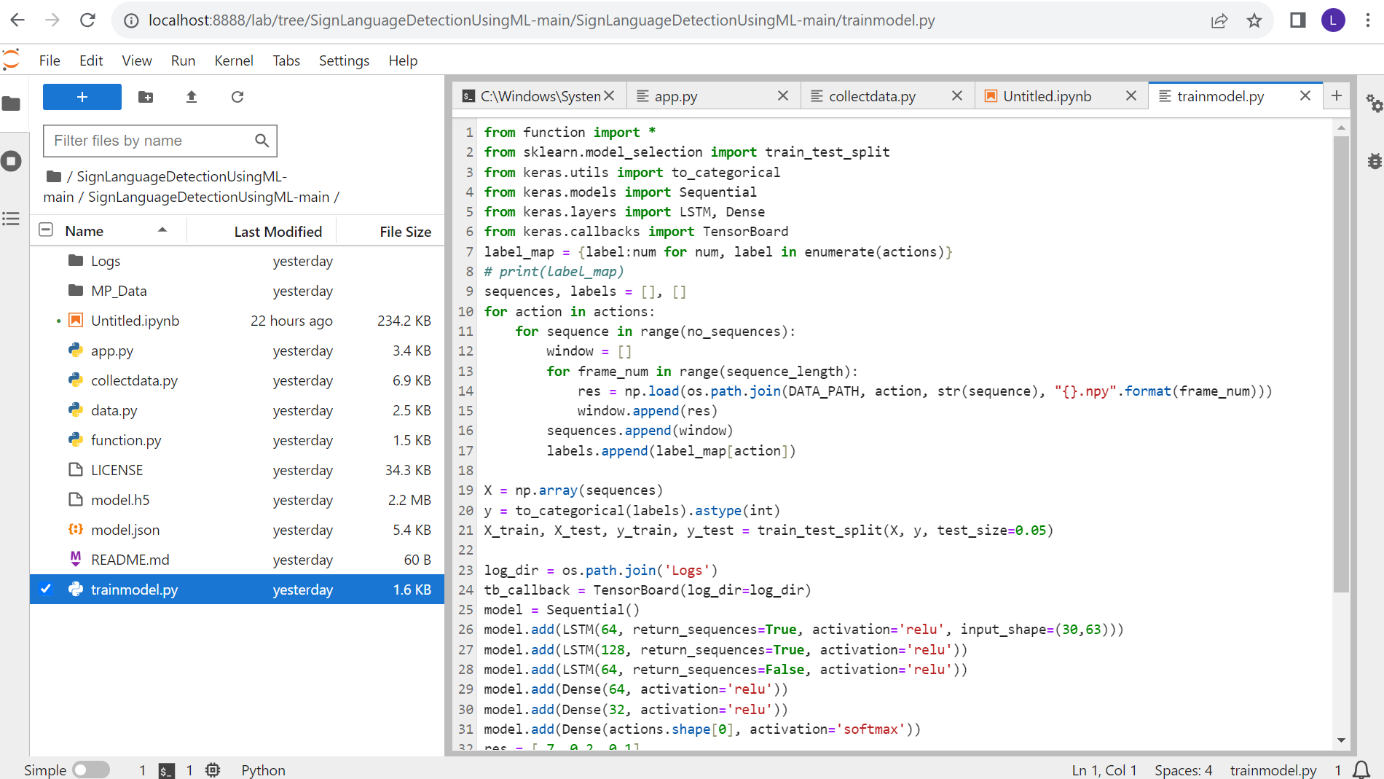
# APPENDICES

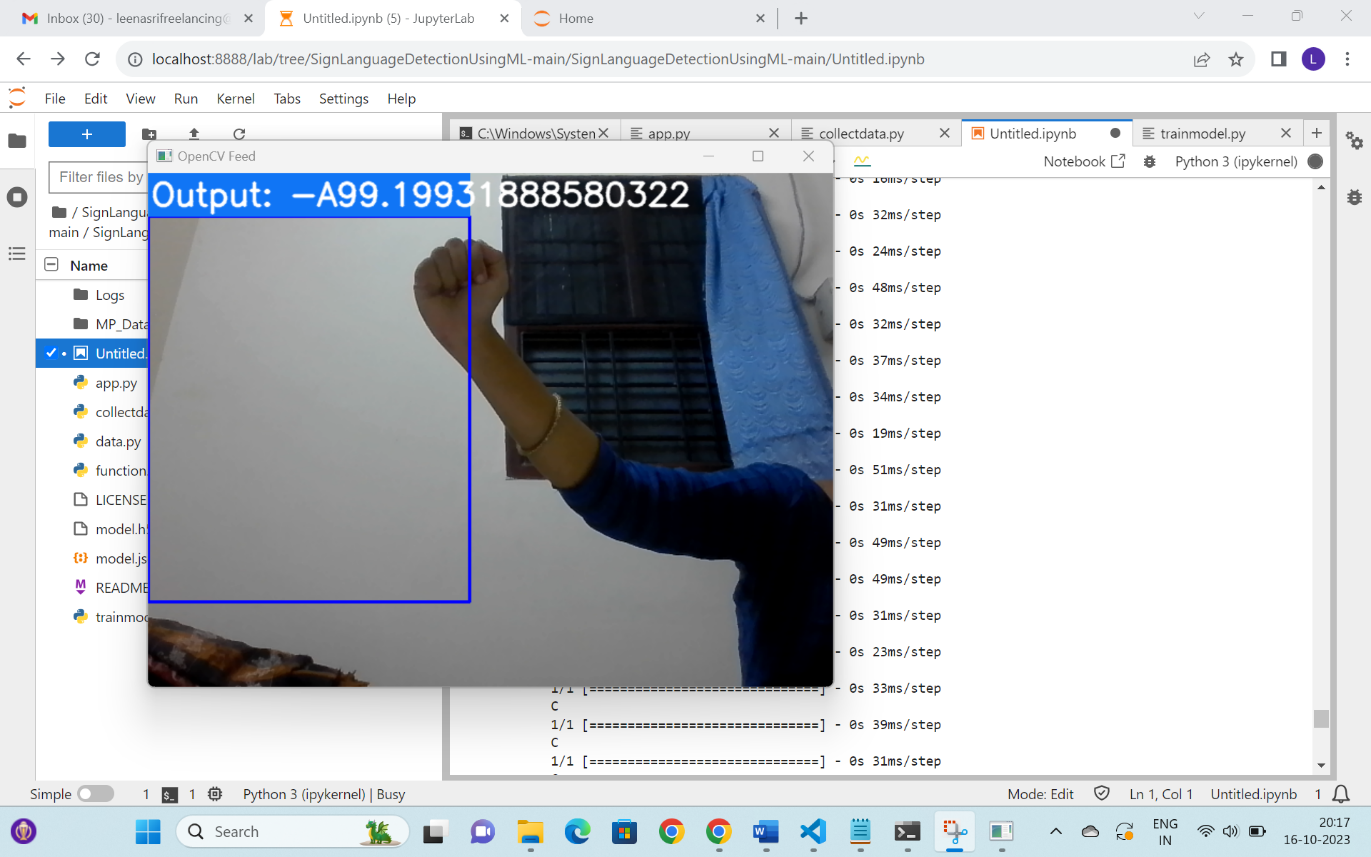
**A.1 SAMPLE**

**APPENDICES**

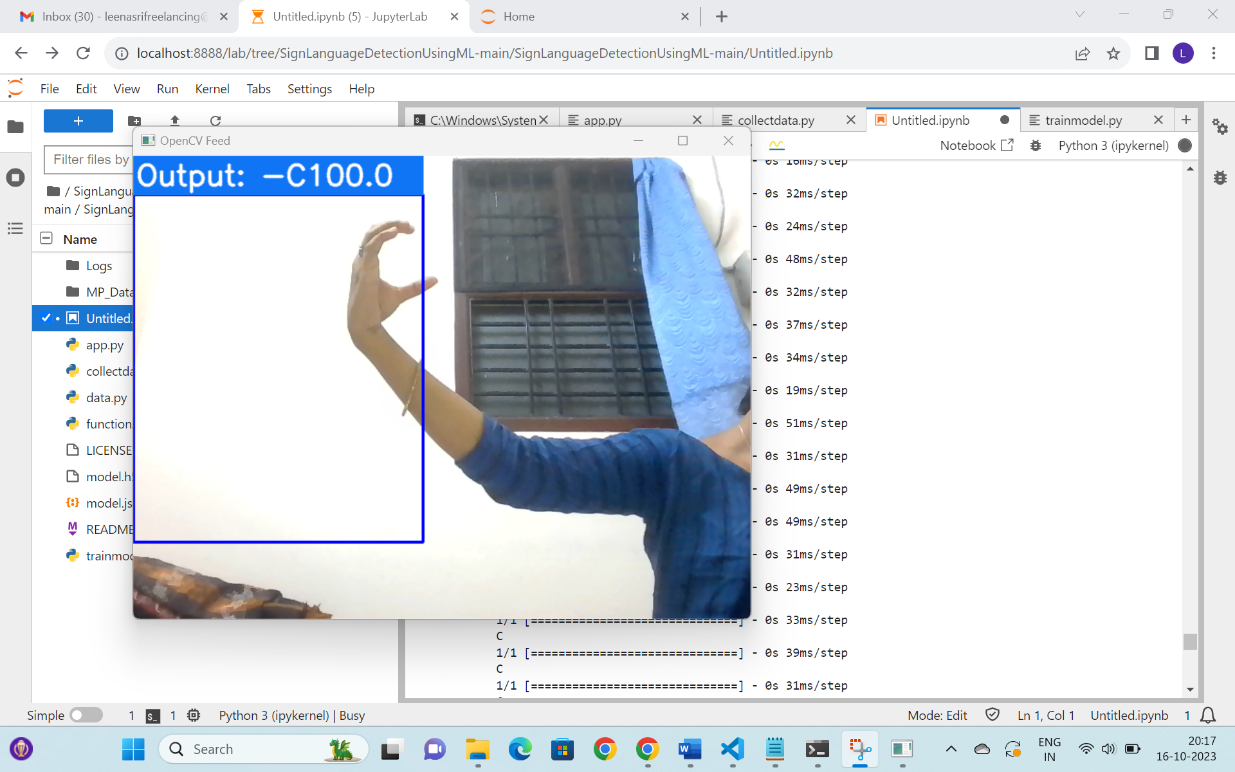
### Fig A.1.1. Jupyter Document Page



**FigA.1.2. Training The System**



### FigA.1.3. Camera Capturing



**FigA.1.4. Captured Gesture Converted To Text Message**

# REFERENCES

# REFERENCES

**REFERENCES:**

[1] Punsara K.K.T- [thilanpunsara@gmail.com](mailto:thilanpunsara@gmail.com),

Premachandra - ravipremachandra@gmail.com,

Chanaka A.W.A.D- achinthachanaka97@gmail.com,

Wijayawickrama [R.V-wijayawick@gmail.com](mailto:R.V-wijayawick@gmail.com),

Nimsiri Abhayasinghe- [nimsiri.a@sliit.lk](mailto:nimsiri.a@sliit.lk),

Rajitha de [Silva-rajitha.d@sliit.lk-](mailto:Silva-rajitha.d@sliit.lk-)

Sri Lanka Institute of Information Technology Malabe, Sri

Lanka: “**IoT Based** **Sign Language Recognition System**”

DOI : [10.1109/ICAC51239.2020.9357267](https://doi.org/10.1109/ICAC51239.2020.9357267)

[2] Mohammed Elmahgiubi-School of Engineering University

of Guelph Ontario Canada,Mohamed Ennajar

-National Information Security & Safety Assurance Libya,

Nabil Drawil, and Mohamed Samir Elbuni-University

of Tripoli Libya: “**Sign Language Translator**

**and Gesture Recognition**”

DOI: [10.1109/GSCIT.2015.7353332](https://doi.org/10.1109/GSCIT.2015.7353332)

[3] Kartik Shenoy, Tejas Dastane, Varun Rao,

Devendra Vyavaharkar-K. J. Somaiya College of

Engineering, University of Mumbai India: **“Real-time**

**Indian Sign Language (ISL) Recognition**”

DOI : [10.1109/ICCCNT.2018.8493808](https://doi.org/10.1109/ICCCNT.2018.8493808)

[4] Ashok Kumar Sahoo-“ **Indian Sign Language**

**Recognition Using Machine Learning Techniques**”

DOI: 10.1002/masy.202000241

[5] Dhirendra Kumar Choudhary, Rameet Singh ,

Deepali Kamathania -Vivekananda Institute of

Professional Studies, Delhi, India:” **Sign**

**Language Recognition System**”.

https://ssrn.com/abstract=3832151

[6] Yutong Chen, Fangyun Wei, Xiao Sun,Zhirong

Wu,Stephen Lin,Tsinghua University ,Microsoft Research Asia ,

**”A Simple Multi-Modality Transfer Learning Baseline for Sign Language Translation”**

<https://doi.org/10.48550/arXiv.2203.04287>

 [7] Prof. D.S.Shingate, ,Rutika Bajaj, Anshu Singh, Gayatri Walzade, Yogita Bhavar  SIEM Computer Department, Nashik, India

” **Sign Language Interpretation**

ISSN: 2455-2631

© October 2019 IJSDR | Volume 4, Issue 10

[8] Becky Sue Parton University of North **Texas**

**Sign Language Recognition and Translation: A Multidisciplined Approach From the Field of Artificial Intelligenc**e

The Journal of Deaf Studies and Deaf Education, Volume 11, Issue 1, Winter 2006, , <https://doi.org/10.1093/deafed/enj003>

 [9]  Eman K. Elsayed, Doaa R. Fathy Mathematical and computer science Al-Azhar University Cairo, Egypt

**Sign Language Semantic Translation System using Ontology and Deep Learning ,** (IJACSA) International Journal of Advanced Computer Science and Applications

,Vol. 11, No. 1, 2020

[10] Swati Sharma, Varun Prakash Saxena, Kandukuru Satish

**Comparative Analysis On Sign Language Recognition System** i

nternational journal of scientific & technology research volume 8, issue 08, august 2019 issn 2277-8616

[11] Rachana Patil1, Vivek Patil1, Abhishek Bahuguna1, and

Mr. Gaurav Datkhile1,Ramrao Adik Institute Of Technology,Nerul, India

**Indian Sign Language Recognition using Convolutional Neural Network**

ITM Web of Conferences 40, 03004 (2021) ICACC-2021