

SIGN I NET : A DEEP LEARNING BASED MODEL TO RECOGNIZE SIGN LANGUAGE

A PROJECT REPORT

Submitted by

ANNAMALAI P	[211419104013]
ASHWIN KUMAR A	[211419104027]
ABISHEK BENJAMIN RAJ D	[211419104003]

in partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING



PANIMALAR ENGINEERING COLLEGE

(An Autonomous Institution, Affiliated to Anna University, Chennai)

APRIL 2023

PANIMALAR ENGINEERING COLLEGE

(An Autonomous Institution, Affiliated to Anna University, Chennai)

BONAFIDE CERTIFICATE

Certified that this project report “**SIGN I NET : A DEEP LEARNING BASED MODEL TO RECOGNIZE SIGN LANGUAGE**” is the bonafide work of “**ANNAMALAI P (211419104013), ASHWIN KUMAR A (211419104027), ABISHEK BENJAMIN RAJ D (211419104003)**” who carried out the project work under my supervision.

SIGNATURE

**Dr.L.JABASHEELA,M.E.,Ph.D.,
HEAD OF THE DEPARTMENT**

DEPARTMENT OF CSE,
PANIMALAR ENGINEERING COLLEGE,
NAZARATHPETTAI,
POONAMALLEE,
CHENNAI-600 123.

SIGNATURE

**Dr.R.JOSPHINELEELA,M.E.,Ph.D,
SUPERVISOR
PROFESSOR**

DEPARTMENT OF CSE,
PANIMALAR ENGINEERING COLLEGE,
NAZARATHPETTAI,
POONAMALLEE,
CHENNAI-600 123.

Certified that the above candidate(s) was/ were examined in the End Semester Project Viva-Voce Examination held on.....

INTERNAL EXAMINER

EXTERNAL EXAMINER

DECLARATION BY THE STUDENT

We **ANNAMALAI P (211419104013)** ,**ASHWIN KUMAR A (211419104027)** , **ABISHEK BENJAMIN RAJ D (211419104003)** hereby declare that this project report titled “**SIGN I NET : A DEEP LEARNING BASED MODEL TO RECOGNIZE SIGN LANGUAGE**”, under the guidance of **Dr.R.JOSPHINELEELA M.E.,Ph.D.**, is the original work done by us and we have not plagiarized or submitted to any other degree in any university by us.

ANNAMALAI P

ASHWIN KUMAR A

ABISHEK BENJAMIN RAJ D

ACKNOWLEDGEMENT

We would like to express our deep gratitude to our respected Secretary and Correspondent **Dr.P.CHINNADURAI, M.A., Ph.D.** for his kind words and enthusiastic motivation, which inspired us a lot in completing this project.

We express our sincere thanks to our beloved Directors **Tmt.C.VIJAYARAJESWARI, Dr.C.SAKTHI KUMAR,M.E.,Ph.D** and **Dr.SARANYASREE SAKTHI KUMAR B.E.,M.B.A.,Ph.D.,** for providing us with the necessary facilities to undertake this project.

We also express our gratitude to our Principal **Dr.K.MANI, M.E., Ph.D.** who facilitated us in completing the project.

We thank the Head of the CSE Department, **Dr. L.JABASHEELA, M.E.,Ph.D.,** for the support extended throughout the project.

We would like to thank our parents, friends, project Guide **Dr.R.JOSPHINELEELA M.E.,Ph.D.,** and coordinator **Dr.N.PUGHAZENDI M.E., Ph.D.,** and all the faculty members of the Department of CSE for their advice and encouragement for the successful completion of the project.

ANNAMALAI P

ASHWIN KUMAR A

ABISHEK BENJAMIN RAJ D

ABSTRACT

Among the people living in the world, of whom 97.6 million are both deaf and blind. People in the age group of 15-49 years, which happens to be age group that contributes the most to science and technology, makes about 17% of this population. To communicate with other humans, deaf- blinded people use communication techniques such as sign language, and written text. Recent advancement in AI, Computer vision has made a lot of progress in helping the hearing and speech impaired people to lead a more comfortable independent life. Several solutions have been proposed to help them in navigation and in many other day-to-day tasks. Solutions to help them access the world of knowledge are comparatively less explored. The information conveyed to general masses from person of hearing-disabilities are primarily rely upon through lip reading. As there is a need for a solution for hearing impaired individuals, we integrated a AI based sign language interpreter to overcome communication hurdles among peers for online based video and real-time conversations. Through our efforts, we hope to have moved an inch closer to bridging the gap between people with hearing ,speech impairment and the world's knowledge pool.

TABLE OF CONTENTS

CHAPTER NO.	TITLE	PAGE NO.
	ABSTRACT	v
	LIST OF TABLES	ix
	LIST OF FIGURES	x
	LIST OF SYMBOLS, ABBREVIATIONS	xi
1.	INTRODUCTION	1
	1.1 Overview	1
	1.2 Problem Definition	2
2.	LITERATURE SURVEY	3
3.	SYSTEM ANALYSIS	7
	3.1 Existing System	7
	3.1.1 Disadvantages of Existing System	7
	3.2 Proposed system	7
	3.2.1 Advantages of Proposed System	8
	3.3 Software Requirements	8
	3.4 Hardware Requirements	8
	3.5 Technologies Used	8
4.	SYSTEM DESIGN	11
	4.1 Data Flow Diagrams(DFD)	11
	4.2 UML Diagrams	13
	4.2.1 Use Case Diagram	14

CHAPTER NO.	TITLE	PAGE NO.
	4.2.2 Class Diagram	15
	4.2.3 Sequence Diagram	17
	4.2.4 Activity Diagram	18
	4.2.5 State Chart Diagram	19
5.	SYSTEM ARCHITECTURE	20
	5.1 Architecture Diagram	20
	5.2 Algorithms	21
	5.2.1 Random Forest	21
	5.2.2 Convolutional neural networks(CNN)	22
6.	SYSTEM IMPLEMENTATION	24
	6.1 Data Pre-Processing	24
	6.1.1 Classification	24
	6.1.2 Normalization	25
	6.1.3 Autocorrect Module	25
	6.1.4 Detection Module	26
7.	SYSTEM TESTING	27
	7.1 White Box Testing	27
	7.2 Black Box Testing	27
	7.3 Test Cases	28
8.	CONCLUSION & FUTURE ENHANCEMENTS	30
	8.1 Conclusion	30
	8.2 Future Enhancements	30

CHAPTER NO.	TITLE	PAGE NO.
	APPENDICES	31
	A.1 Coding	31
	A.2 Sample Screens	47
	REFERENCES	50

LIST OF TABLES

TABLE NO.	TABLE DESCRIPTION	PAGE NO.
7.3.1	Test Case For Hand Signs	28
7.3.2	Test Case For Hand Signs	29

LIST OF FIGURES

FIGURE NO.	FIGURE DESCRIPTION	PAGE NO.
4.1.1	Level 0 of Data Flow Diagram	11
4.1.2	Level 1 of Data Flow Diagram	12
4.1.3	Level 2 of Data Flow Diagram	13
4.2.1	Use Case Diagram	15
4.2.2	Class Diagram	16
4.2.3	Sequence Diagram	17
4.2.4	Activity Diagram	18
4.2.5	State Chart Diagram	19
5.1	Architecture Diagram	20
5.2.1	Random Forest Classifier	22
5.2.2	Layers Architecture	23
6.1.1	Random Forest Classifier	25
6.1.4	Detection Module	26
A2.1	Home Screen Of The Application	47
A2.2	Overlay Landmarks On Hand Signs	48
A2.3	Generated Word From Hand Sign	49

LIST OF SYMBOLS, ABBREVIATIONS

AI	Artificial Intelligence
CNN	Convolutional Neural Network
OpenCV	Open Source Computer Vision
RCNN	Regions Convolutional Neural Networks
UML	Unified Modeling Language
FPS	Frame Per Second
GPU	Graphical Processing Unit
OMG	Object Management Group

CHAPTER 1

INTRODUCTION

1. INTRODUCTION

1.1 OVERVIEW

Communication is very crucial to human beings, as it enables us to express ourselves. We communicate through speech, gestures, body language, reading, writing or through visual aids, speech being one of the most commonly used among them. However, unfortunately, for the speaking and hearing impaired minority, there is a communication gap. Visual aids, or an interpreter, are used for communicating with them. However, these methods are rather cumbersome and expensive, and can't be used in an emergency. Sign Language chiefly uses manual communication to convey meaning. This involves simultaneously combining hand shapes, orientations and movement of the hands, arms or body to express the speaker's thoughts.

Sign Language consists of fingerspelling, which spells out words character by character, and word level association which involves hand gestures that convey the word meaning. Fingerspelling is a vital tool in sign language, as it enables the communication of names, addresses and other words that do not carry a meaning in word level association. In spite of this, fingerspelling is not widely used as it is challenging to understand and difficult to use. Moreover, there is no universal sign language and very few people know it, which makes it an inadequate alternative for communication.

A system for sign language recognition that classifies finger spelling can solve this problem. Various machine learning algorithms are used and their accuracies are recorded and compared in this report.

1.2 PROBLEM DEFINITION

Communication is one of the basic requirement for survival in society. Deaf and dumb people communicate among themselves using sign language but normal people find it difficult to understand their language. Extensive work has been done on American sign language recognition but Indian sign language differs significantly from American sign language. ISL uses two hands for communicating whereas ASL uses single hand for communicating. Using both hands often leads to security of features due to overlapping of hands. In addition to this, lack of datasets along with variance in sign language with locality has resulted in restrained efforts in ISL gesture detection. Our project aims at attacking the basic step in bridging the communication gap between normal people and deaf and dumb people using Indian sign language. Effective extension of this project to words and common expressions may not only make the deaf and dumb people communicate faster and easier with outer world, but also provide a boost. Developing autonomous systems for understanding and aiding them.

CHAPTER 2

LITERATURE SURVEY

2. LITERATURE SURVEY

2.1 Sign language recognition using deep learning on custom processed static gesture images

Author Name : A. Das, S. Gawde, K. Suratwala and D. Kalbande

Year of Publish 2018

International Conference on Smart City and Emerging Technology (ICSCET) have researched on deep learning-based sign language recognition system with processed static images implemented on American Sign Language gestures. The dataset consisted of 24 labels of static gestures of alphabets from A to Z, excluding J. There were approximately 100 images per class in the dataset captured on an RGB sensor. The methodology included using of Inception V3 convolutional neural network model to train the model. After training and testing the model, the average accuracy rate of validation was over 90%, with the most outstanding validation accuracy being 98%. They concluded that the relatively new InceptionV3 model could be an appropriate model for static sign language detection when provided with a dataset of properly cropped images.

2.2 Indian sign language recognition using machine learning techniques

Author Name : A. K. Sahoo

Year of Publish 2021

This research focused on the static hand gestures in Indian sign language for the numeric values (0-9). A digital RGB sensor was used to capture the images of the signs to build the dataset. The dataset contained 5000 total images, with 500 images for each digit from 0 to 9. Two classifiers were used based on the supervised learning technique to train the model: Naïve Bayes and kNearest Neighbor. K-Nearest Neighbor technique slightly performed better than the Naïve Bayes classifier in this research as the average accuracy rates of k-NN and Naïve Bayes were 98.36% and 97.79%, respectively.

2.3 Real-time vernacular sign language recognition using mediapipe and machine learning

Author Name : A. Halder and A. Tayad

Year of Publish 2021

Used the MediaPipe framework to get multi-hand landmarks and used Support Vector Machine (SVM) for Real-time detection of hand signs. The average accuracy achieved was about 99%. Work adopts the hand landmarks detection approach at the core of the complete model. Using MediaPipe's hand tracking model, 21 hand landmarks in each image containing hand signs

2.4 Shape, texture and local movement hand gesture features for Indian sign language recognition

Author Name : J. Rekha, J. Bhattacharya and S. Majumder

Year of Publish 2011

Trends in information sciences & computing (TISC2011) used a threshold-based technique for segmentation of hands from Kinect captured images. They had ten classes with 1000 images in total in their dataset. The accuracy rate achieved was about 93%.

2.5 Real time hand pose estimation using depth sensors

Author Name : C. Keskin, F. Kırac, Y. E. Kara and L. Akarun

Year of Publish 2013

Used an approach involving object recognition by parts to recognize signs indicating digits of American Sign Language. Their dataset had ten classes with a total of 30,000 3d depth images captured using a Kinect sensor, and they achieved about 99% accuracy rate.

2.6 Spelling it out: Real-time ASL fingerspelling recognition

Author Name : N. Pugeault and R. Bowden

Year of Publish 2011

They worked on a real-time recognition system for recognition of the alphabets in American Sign Language. A dataset consisting of 24 classes with 48,000 3D depth images captured using a Kinect sensor was used. Gabor filters and multi-class random forests were used and highly accurate classification rates were achieved.

2.7 Hand pose identification from monocular image for sign languagerecognition

Author Name : M. K. Bhuyan, M. K. Kar and D. R. Neog

Year of Publish 2011

Image Processing Applications (ICSIPA) used a dataset of 8 gestures from Indian Sign Language consisting of 400 images. They used a skin color based segmentation technique for hand detection, then used nearest neighbor classification, and finally achieved a recognition rate above 90%.

2.8 Nearest neighbour classification of Indian sign language gestures usingkinect camera

Author Name : Z. A. Ansari and G. Harit

Year of Publish 2016

Classifying Indian sign language static gestures using images with 3d depth data. The images were captured using Microsoft Kinect, which enables 3d depth information capturing along with the 2D image. The dataset totaled 5041 images of static hand gestures and was labeled with 140 classes. K-means clustering was used to train the model. In the research, they were able to score a 90% accuracy rate for 13 signs and 100% accuracy for three signs making it up to 16 alphabets (A, B, D, E, F, G, H, K, P, R, T, U, W, X, Y, Z) recognition with an average accuracy rate of 90.68%.

2.9 A Comprehensive Study on Deep Learning-based Methods for Sign Language Recognition

Author Name : Nikolas Adaloglou, Theodoris Chatzis

Year of Publish 2021

In this research, computer vision-based approaches for sign language detection are experimentally compared and evaluated. The most current deep neural network techniques in this area are used to conduct a complete analysis on several publicly accessible datasets. The current study's objective is to provide light on the identification of sign language while concentrating on the mapping of non-segmented video streams to glosses. Two novel sequence training criteria, which are well-known in the disciplines of voice and scene text recognition, are presented for this job. Additionally, a wide range of pre-training programs are in-depth covered. Finally, a fresh RGB+D dataset is produced for Greek sign language. As far as we are aware, this is the first sign language dataset where three levels of annotation (individual gloss, sentence, and spoken language) are supplied for the same batch of video recordings.

2.10 Attention is All You Sign: Sign Language Translation with Transformers

Author Name : Kayo Yin and Jesse Read

Year of Publish 2019

By using transformers, this study enhances the translation process for sign language translation (SLT). We present a unique end-to-end SLT system that combines Transformer networks with Spatial-Temporal Multi-Cue (STMC) and a variety of experimental findings for different Transformer setups. On the ground truth (GT) glosses and anticipated glosses of the PHOENIX-Weather 2014T dataset, our technique outperforms the state-of-the-art by more than 5 and 7 BLEU, respectively. We report an improvement of over 16 BLEU on the ASLG-PC12 corpus. Additionally, our results show that translation using predicted glosses works better than translation using GT glosses. As a result, it is possible to further enhance SLT by either reworking the gloss annotation technique or concurrently training the SLR and translation systems.

CHAPTER 3

SYSTEM ANALYSIS

3. SYSTEM ANALYSIS

3.1 EXISTING SYSTEM

Sign Language Recognition (SLR) system, which is required to recognize sign languages, has been widely studied for years. The studies are based on various input sensors, gesture segmentation, extraction of features and classification methods. This paper aims to analyze and compare the methods employed in the SLR systems, classification methods that have been used, and suggests the most promising method for future research. Due to recent advancement in classification methods, many of the recent proposed works mainly contribute to the classification methods, such as hybrid method and Deep Learning. This project focuses on the classification methods used in prior Sign Language Recognition system. Based on our review, HMM based approaches have been explored extensively in prior research, including its modifications.

3.2 DISADVANTAGES OF EXISTING SYSTEM

- Lack of overlay landmarks for hand gesture recognition
- Lesser Accuracy and comparative analysis of Machine learning Algorithms
- Significant dataset, data preparation and normalization are not handled
- CNN based Neural Network are not fine tuned, Overfitting cases are to be addressed

3.3 PROPOSED SYSTEM

Our proposed system is sign language recognition system using convolution neural networks which recognizes various hand gestures by capturing video and converting it into frames. Then the hand pixels are segmented and the image is obtained and sent for comparison to the trained model. Thus our system is more robust in getting exact text labels of letters. This work contributes to a deep learning powered American Sign Language Recognition system focused on recognizing a few fundamental letters of the American Alphabet. The approach includes a methodology that involves implementation of a hand tracking solution provided by Google's open-source project, MediaPipe. Along with it, a deep learning algorithm is implemented on this solution to produce a fast, inexpensive, lightweight, and easy-to-deploy system that can be used as the core of a complete sign language recognition system.

3.4 ADVANTAGES OF PROPOSED SYSTEM

- Manipulating a Virtual Camera
- ASL Alphabet Recognition
- Output to Virtual Camera
- Autocorrect
- Showcase on the Web
- Detecting Hands

3.5 FEASIBILITY STUDY

This work adopts the hand landmarks detection approach at the core of the complete model. Using MediaPipe's hand tracking model, 21 hand landmarks in each image containing hand signs from American Sign Language have been detected. The landmarks are then collected as coordinate points, normalized, and saved in a .csv file as datapoints. A neural network model then takes these data-points as input, and after training the model, real time hand sign recognition with OpenCV has been implemented using the trained model. Fig. 1 gives an overview of the complete project. The prepared model for American alphabets is highly efficient, accurate, portable, and lightweight. This project discusses the entire procedure of developing our model, along with the results and analysis of performance.

3.6 HARDWARE REQUIREMENT

- Hard Disk : 500GB and Above
- RAM : 4GB and Above
- Processor : I3 and Above

3.7 SOFTWARE REQUIREMENT

- Operating System : Windows 10 (64 bit)
- Software : Python
- Tools : Anaconda

3.8 TECHNOLOGIES USED

- Python
- Deep Learning

3.8.1 Python

Python is a high-level, interpreted programming language that is widely used in various domains such as web development, data science, artificial intelligence, scientific computing, and more. It was first released in 1991 and has since become one of the most popular programming languages in the world. Some key features of Python include:

- **Easy to Learn:** Python has a simple and easy-to-learn syntax, which makes it an ideal language for beginners.
- **Interpreted Language:** Python is an interpreted language, which means that the code is executed line by line, making it easier to test and debug.
- **Cross-Platform:** Python can be run on various platforms, including Windows, macOS, and Linux.
- **Large Standard Library:** Python has a large standard library that provides a wide range of built-in modules for various tasks, such as file I/O, regular expressions, networking, and more.
- **Open Source:** Python is open-source software, which means that the source code is freely available to anyone and can be modified and redistributed.
- **Object-Oriented:** Python is an object-oriented language, which means that it supports object-oriented programming concepts such as encapsulation, inheritance, and polymorphism.

Python has a vast community of developers and users, which has contributed to its popularity and growth. It has a large number of third-party libraries and frameworks that make it easier to develop complex applications quickly. Some popular Python frameworks include Django, Flask, and Pyramid for web development, and NumPy, Pandas, and SciPy for scientific computing and data analysis. In summary, Python's simplicity, ease of use, and versatility have made it a popular programming language in various industries, from web development to scientific computing and artificial intelligence.

3.8.2 Deep Learning

Deep learning is a subfield of machine learning that uses neural networks to model and solve complex problems. Neural networks are a type of artificial intelligence modeled after the structure of the human brain. They consist of layers of interconnected nodes, or artificial neurons, that are trained to recognize patterns in data. In deep learning, neural networks are organized in layers, with each layer learning to identify and extract different features from the data. The layers are stacked one on top of the other, creating a deep neural network that can learn complex representations of the data. Deep learning has shown remarkable success in many areas of artificial intelligence, such as image recognition, natural language processing, speech recognition, and robotics. It has enabled breakthroughs in areas such as self-driving cars, virtual assistants, and medical diagnosis. Some of the popular deep learning architectures include Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and Generative Adversarial Networks (GANs). CNNs are commonly used in image recognition tasks, RNNs are used for sequential data such as natural language processing, and GANs are used for image generation. Deep learning requires large amounts of labeled data and computational power to train the neural networks. With the availability of big data and advancements in hardware, deep learning has become a widely used and powerful tool for solving complex problems in various industries.

CHAPTER 4

SYSTEM DESIGN

4. SYSTEM DESIGN

4.1 DATA FLOW DIAGRAM (DFD)

A data flow diagram (DFD) is a graphical or visual representation using a standardized set of symbols and notations to describe a business's operations through data movement. They are often elements of a formal methodology such as Structured Systems Analysis and Design Method (SSADM). Superficially, DFDs can resemble flow charts or Unified Modeling Language (UML), but they are not meant to represent details of software logic. DFDs make it easy to depict the business requirements of applications by representing the sequence of process steps and flow of information using a graphical representation or visual representation rather than a textual description. When used through an entire development process, they first document the results of business analysis. Then, they refine the representation to show how information moves through, and is changed by, application flows. Both automated and manual processes are represented.

LEVEL 0

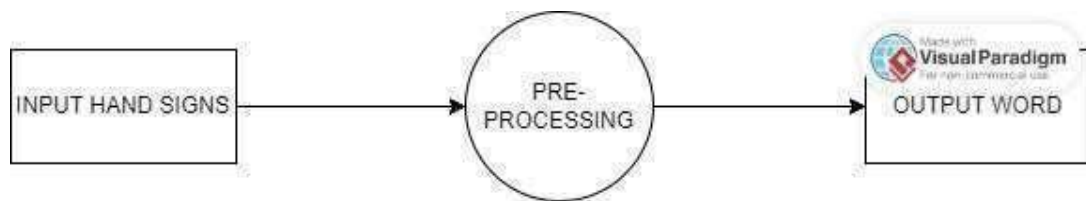


Fig 4.1.1 Level 0 of Data Flow Diagram

LEVEL 1

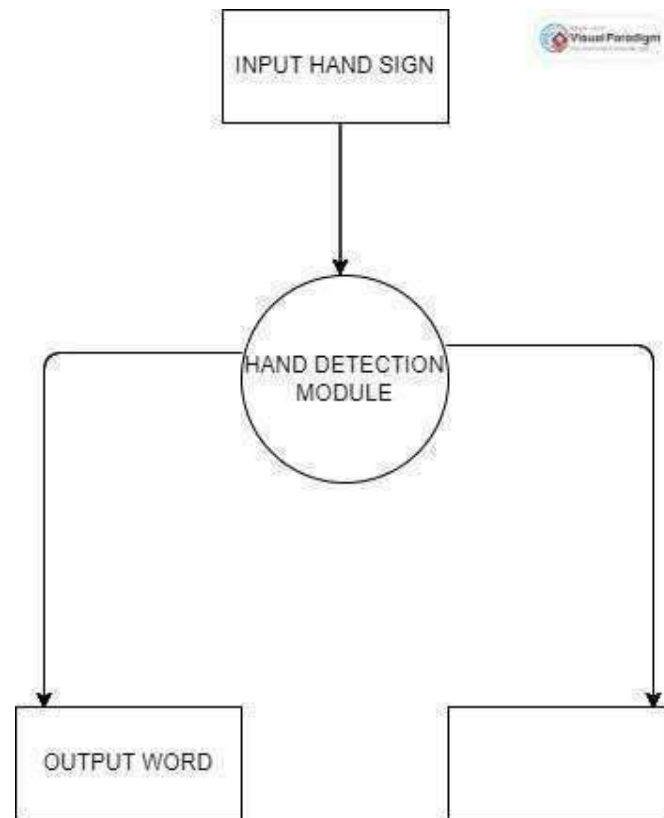


Fig 4.1.2 Level 1 of Data Flow Diagram

LEVEL 2

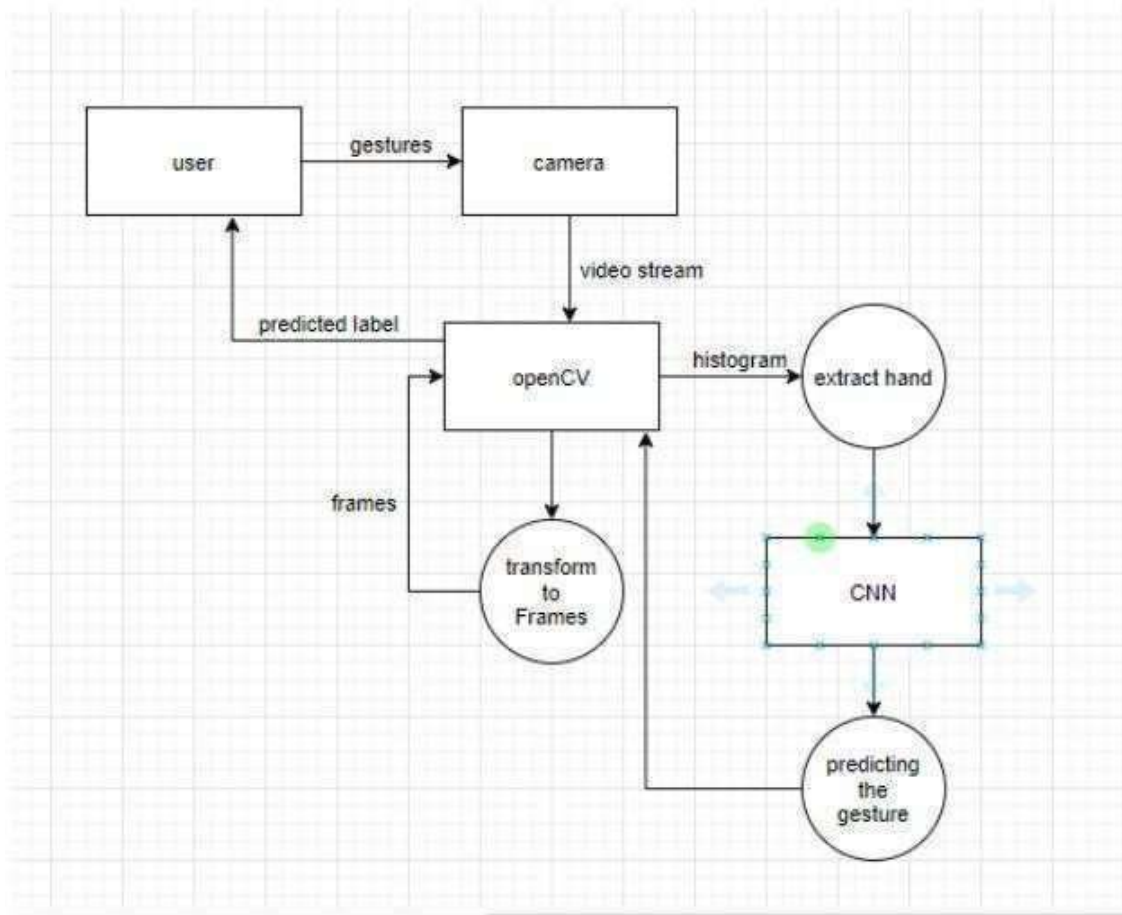


Fig 4.1.3 Level 2 of Data Flow Diagram

4.2 UML DIAGRAMS

UML is a standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems. UML was created by the Object Management Group (OMG) and UML 1.0 specification draft was proposed to the OMG in January 1997. OMG is continuously making efforts to create a truly industry standard. UML stands for Unified Modeling Language. UML is different from the other common programming languages such as C++, Java, COBOL, etc. UML is a pictorial language used to make software blueprints. UML can be described as a general purpose visual modeling language to visualize, specify, construct, and document software system. Although UML is generally used to model software systems, it is not limited within this boundary. It is also used to model non-software systems as well.

For example, the process flow in a manufacturing unit, etc. UML is not a programming language but tools can be used to generate code in various languages using UML diagrams. UML has a direct relation with object oriented analysis and design. After some standardization, UML has become an OMG standard.

4.2.1 Use Case Diagram

A use case diagram is a type of Unified Modeling Language (UML) diagram that represents the interactions between a system and its actors, and the various use cases that the system supports. It is a visual representation of the functional requirements of the system and the actors that interact with it. Use case diagrams typically include the following elements:

- **Actors:** Actors are external entities that interact with the system. They can be human users, other systems, or devices.
- **Use Cases:** Use cases are the specific functions or tasks that the system can perform. Each use case represents a specific interaction between an actor and the system.
- **Relationships:** Relationships are used to indicate how the actors and use cases are related to each other. The two main relationships in a use case diagram are "uses" and "extends". "Uses" relationship indicates that an actor uses a specific use case, while "extends" relationship indicates that a use case extends or adds functionality to another use case.
- **System Boundary:** The system boundary is a box that contains all the actors and use cases in the system. It represents the physical or logical boundary of the system being modeled.

Use case diagrams are useful for identifying the functional requirements of a system, and for communicating these requirements to stakeholders. They can be used in the requirements gathering phase of software development, as well as in the design and testing phases.

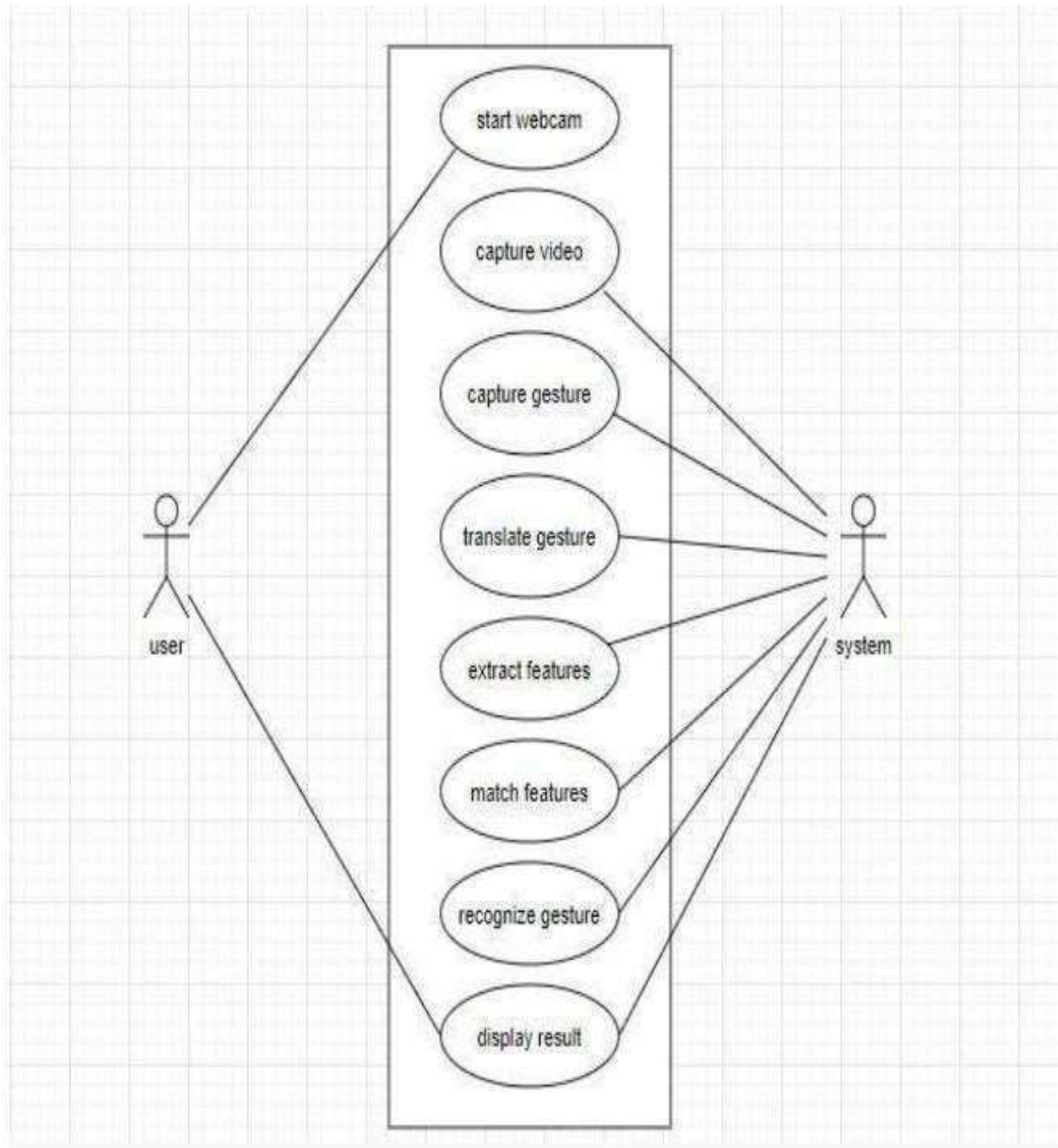


Fig 4.2.1 Use Case Diagram

4.2.2 Class Diagram

The class diagram depicts a static view of an application. It represents the types of objects residing in the system and the relationships between them. A class consists of its objects, and also it may inherit from other classes. A class diagram is used to visualize, describe, document various different aspects of the system, and also construct executable software code. It shows the attributes, classes, functions,

and relationships to give an overview of the software system. It constitutes class names, attributes, and functions in a separate compartment that helps in software development. Since it is a collection of classes, interfaces, associations, collaborations, and constraints, it is termed as a structural diagram. The main purpose of class diagrams is to build a static view of an application. It is the only diagram that is widely used for construction, and it can be mapped with object-oriented languages.

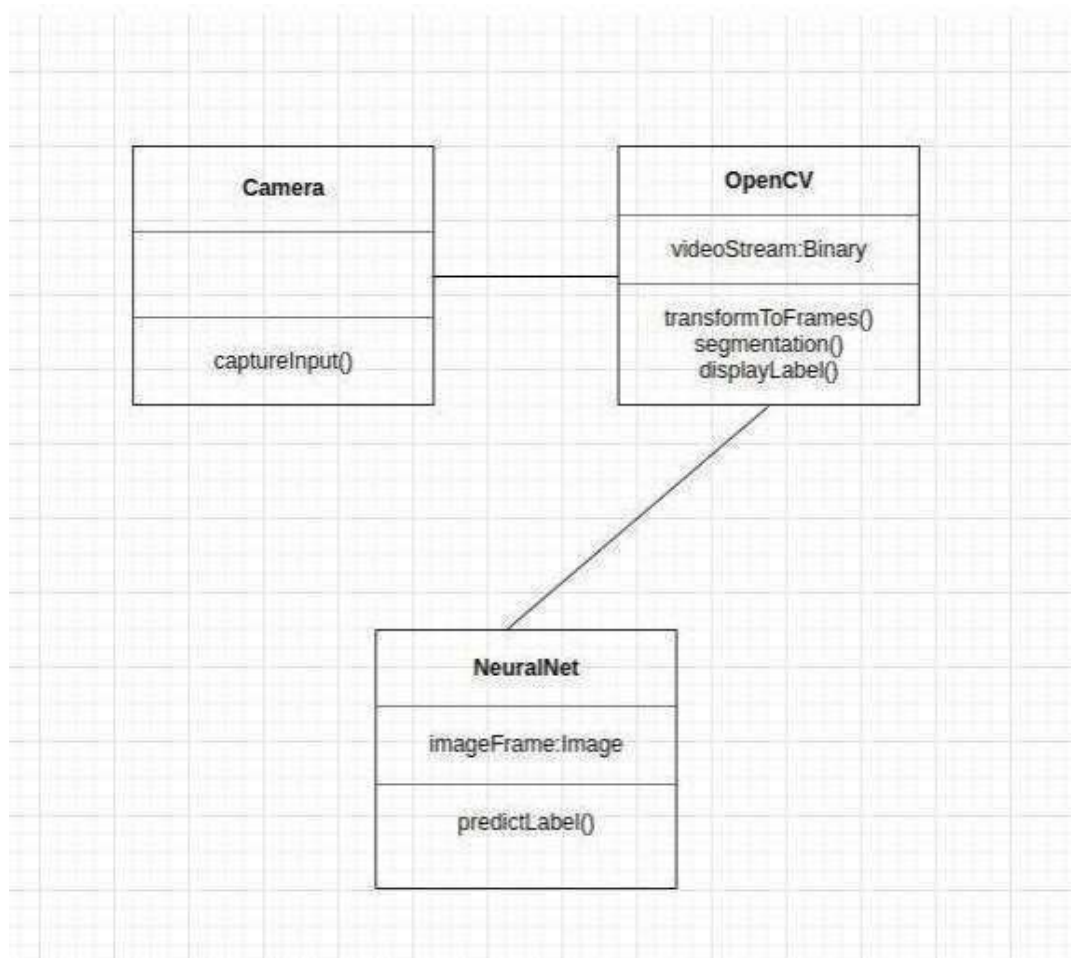


Fig 4.2.2 Class Diagram

4.2.3 Sequence Diagram

A Sequence diagram is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of Message Sequence diagrams are sometimes called event diagrams, event sceneries and timing diagram.

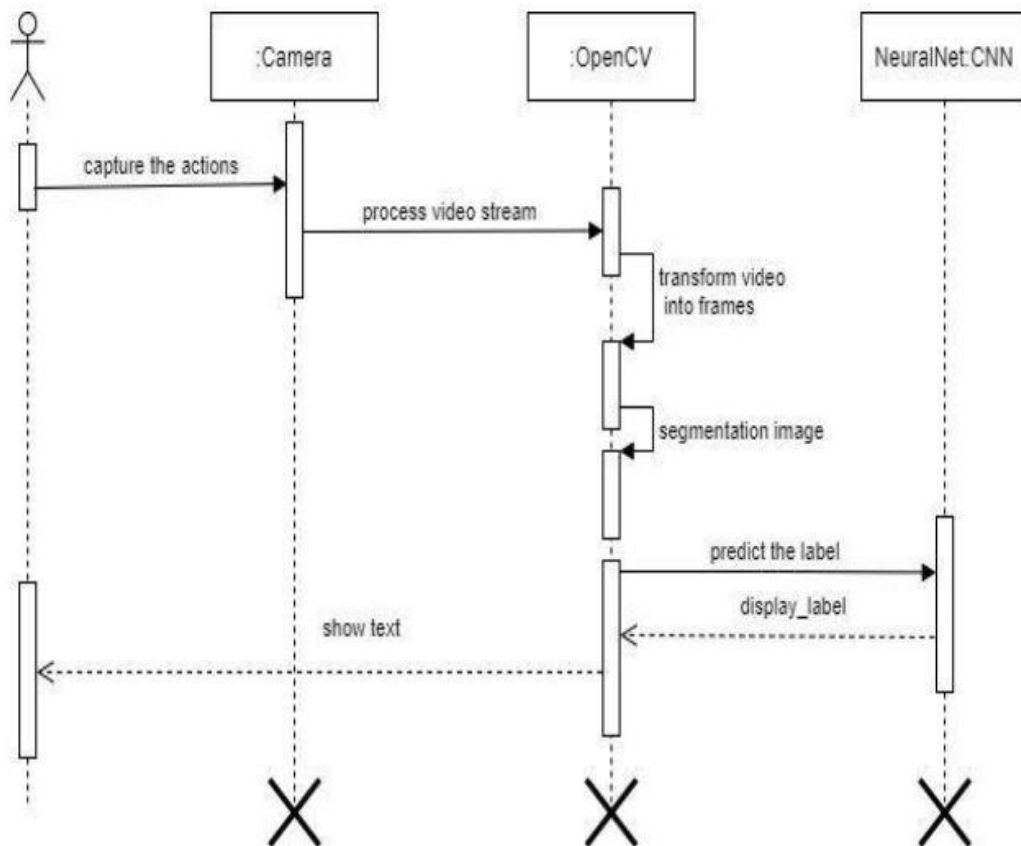


Fig 4.2.3 Sequence Diagram

4.2.4 Activity Diagram

Activity diagram is a graphical representation of workflows of stepwise activities and actions with support for choice, iteration and concurrency. An activity diagram shows the overall flow of control.

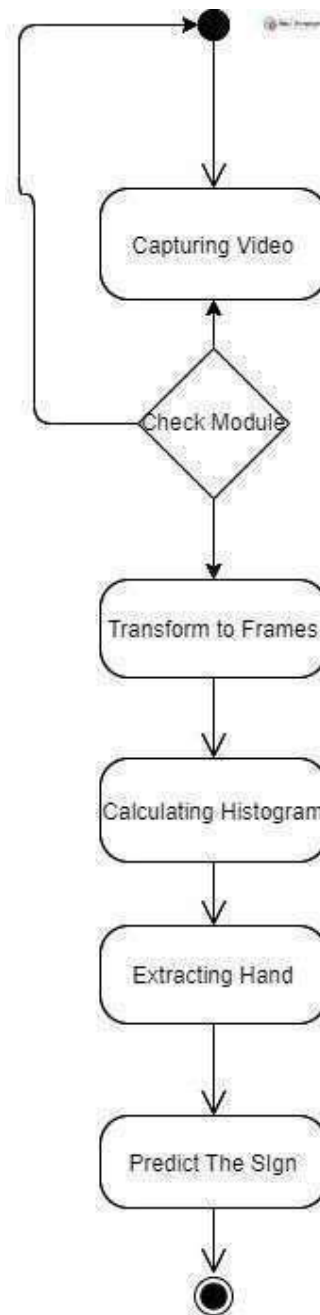


Fig 4.2.4 Activity Diagram

4.2.5 State Chart Diagram

A state chart diagram describes a state machine which shows the behavior of classes. It shows the actual changes in state not processes or commands that create those changes and is the dynamic behavior of objects over time by modeling the life cycle of objects of each class. It describes how an object is changing from one state to another state

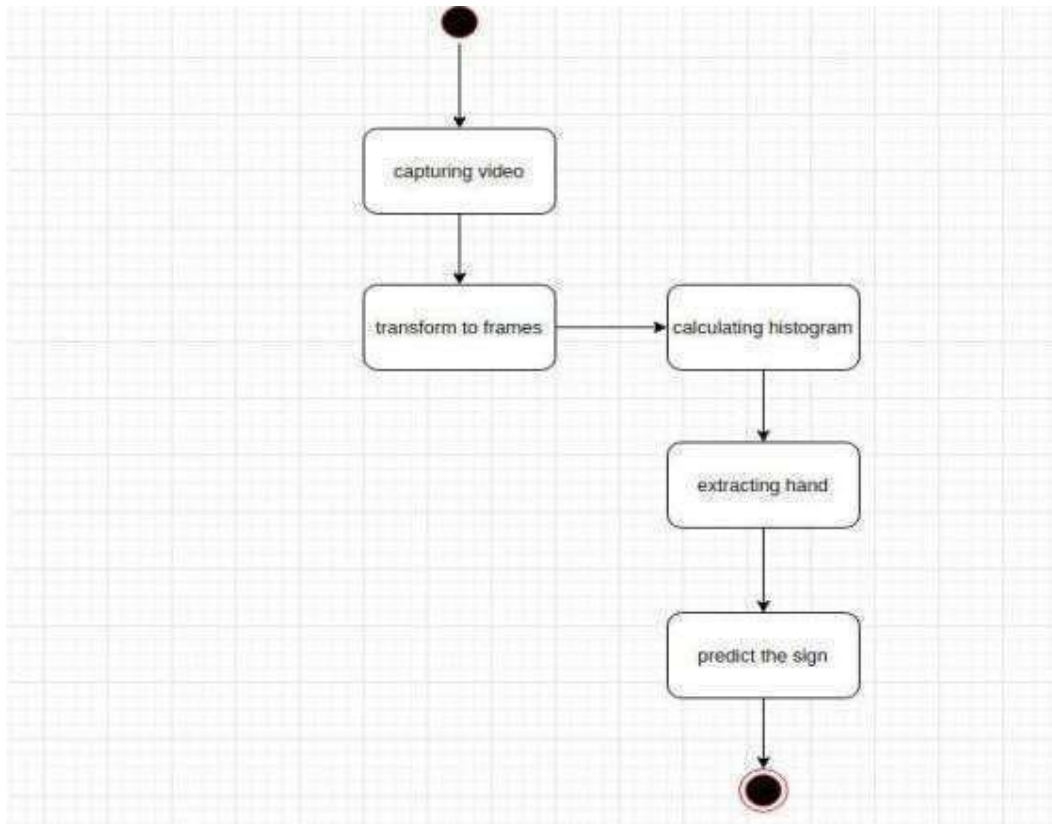


Fig 4.2.5 State Chart Diagram

CHAPTER 5

SYSTEM ARCHITECTURE

5. SYSTEM ARCHITECTURE

5.1 ARCHITECTURE DIAGRAM

An architecture diagram is a graphical representation of a set of concepts that are part of an architecture, including their principles, elements and components. It is also defined as a visual representation that maps out the physical implementation for components of a software system. It shows the general structure of the software system and the associations, limitations, and boundaries between each element.

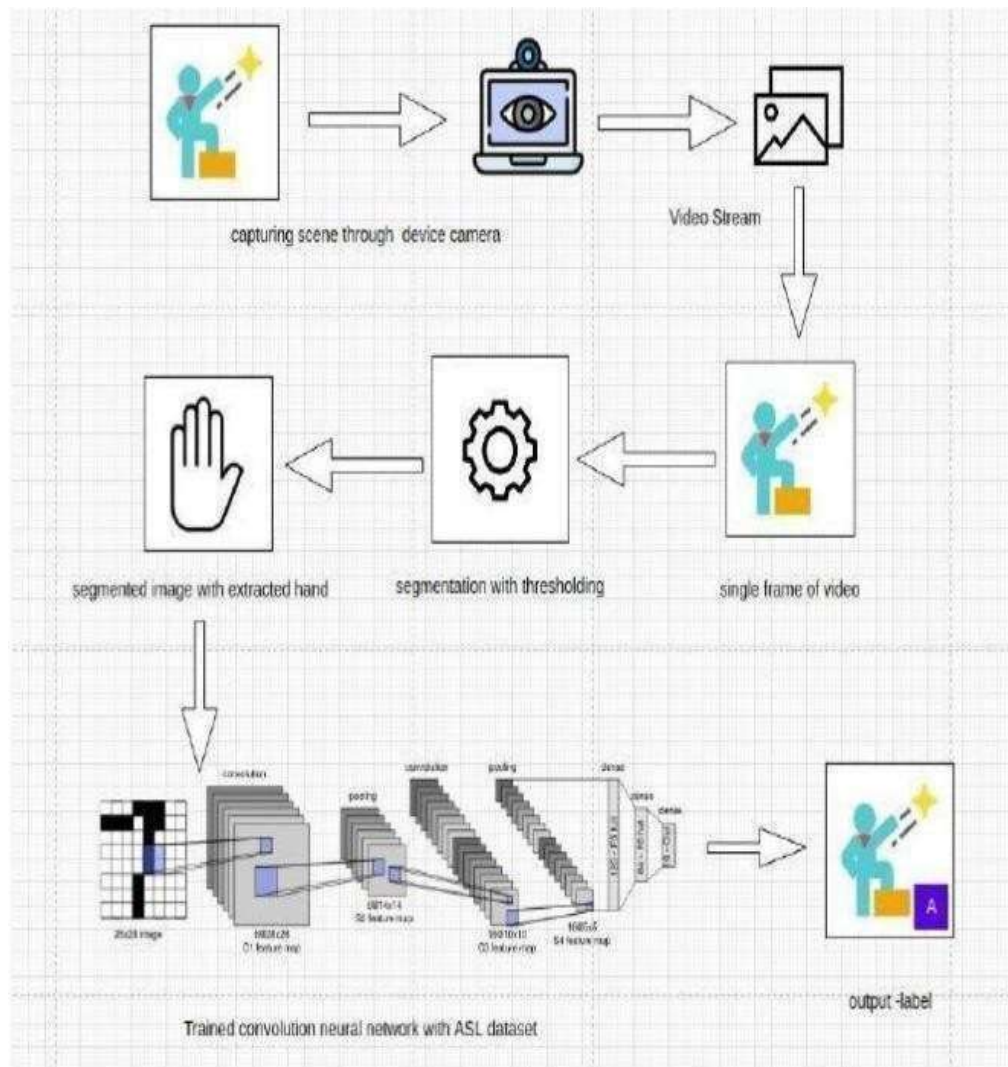


Fig 5.1 Architecture Diagram

This graphic provides a concise and understandable description of all the entities currently integrated into the system. The diagram shows how the many actions and choices are linked together. You might say that the whole process and how it was carried out is a picture. The figure below shows the functional connections between various entities.

5.2 ALGORITHMS

5.2.1 Random Forest

Random forests are an ensemble learning technique that combines multiple decision trees into a forest or final model of decision trees that ultimately produces more accurate and stable predictions.

Random forests operate on the principle that a large number of trees operating as a committee (forming a strong learner) will outperform a single constituent tree (a weak learner). This is akin to the requirement in statistics to have a sample size large enough to be statistically relevant. Some individual trees may be wrong but as long as the individual trees are not making completely random predictions, their aggregate will form an approximation of the underlying data.

As part of their construction, random forest predictors naturally lead to a dissimilarity measure among the observations. One can also define a random forest dissimilarity measure between unlabeled data: the idea is to construct a random forest predictor that distinguishes the "observed" data from suitably generated synthetic data. The observed data are the original unlabeled data and the synthetic data are drawn from a reference distribution. A random forest dissimilarity can be attractive because it handles mixed variable types very well, is invariant to monotonic transformations of the input variables, and is robust to outlying observations.

The random forest dissimilarity easily deals with a large number of semi-continuous variables due to its intrinsic variable selection; for example, the "Add cl 1" random forest dissimilarity weighs the contribution of each variable according to how dependent it is on other variables. The random forest dissimilarity has been used in a variety of applications, e.g. to find clusters of patients based on tissue marker data.

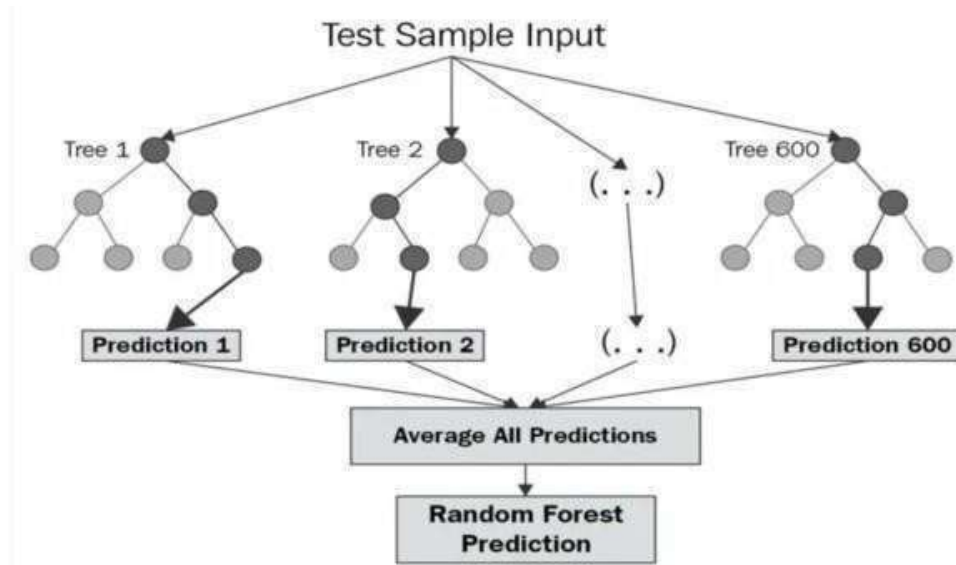


Fig 5.2.1 : Random Forest Classifier

5.2.2 Convolutional Neural Network (CNN)

Convolutional Neural Networks (CNN), are deep neural networks used to process data that have a grid-like topology, e.g images that can be represented as a 2-D array of pixels. A CNN model consists of four main operations: Convolution, Non-Linearity, Pooling and Classification (Fully-connected layer). Convolution: The purpose of convolution is to extract features from the input image. It preserves the spatial relationship between pixels by learning image features using small squares of input data. It is usually followed by ReLU.

- **ReLU:** It is an element-wise operation that replaces all negative pixel values in the feature map by zero. Its purpose is to introduce non-linearity in a convolution network.
- **Pooling:** Pooling (also called down sampling) reduces the dimensionality of each feature map but retains important data.
- **Fully-connected layer:** It is a multi layer perceptron that uses soft max function in the output layer. Its purpose is to use features from previous layers for classifying the input image into various classes based on training data.

The combination of these layers is used to create a CNN model. The last layer is a fully connected layer.

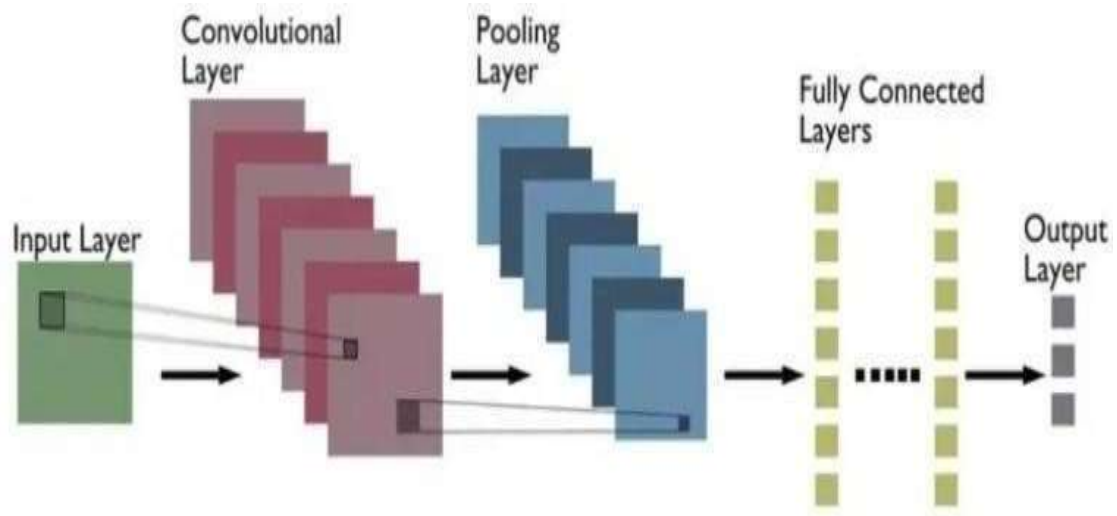


Fig 5.2.2 : Layers Architecture

CHAPTER 6

SYSTEM IMPLEMENTATION

6. SYSTEM IMPLEMENTATION

6.1 DATA PRE-PROCESSING

Data preprocessing is a process of preparing the raw data and making it suitable for a machine learning model. It is the important process in machine learning model. When creating a machine learning project, it is not always a case that we come across the clean and formatted data so to verify that and to get a clean dataset we do data pre processing so after that we will be able to obtain clean and formatted dataset and then we can proceed further with our project implementation.

-In this the input values will be in the form of raw dataset and after pre processing the output will be clean and formatted dataset.

-To get pre processed data first we imported libraries like pandas and numpy where pandas is used to analyze big data and to make conclusions and numpy is used for working with arrays. After importing the libraries we will link the dataset with the notebook for processing with the help of df and read function. Then we use shape , size and column function to know more about the dataset for processing . is null function is used to return a data frame object where values are replaced with true for null values and false for non-null values. We use dropna function to remove rows which contain null values. And also we use group by and duplicate function for aggregation , analysis and remove duplicate values.

6.1.1 Classification

In a random forest classification, multiple decision trees are created using different random subsets of the data and features.

Each decision tree is like an expert, providing its opinion on how to classify the data. Predictions are made by calculating the prediction for each decision tree, then taking the most popular result. The random forest would count the number of predictions from decision trees for Cat and for Dog, and choose the most popular prediction.

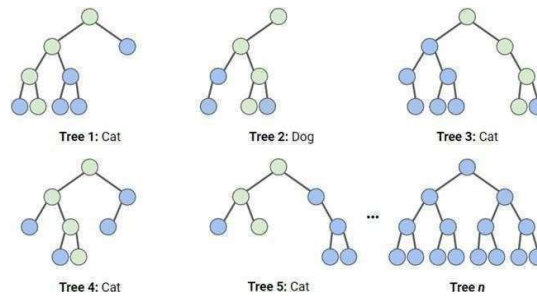


Fig 6.1.1 : Random Forest Classifier

6.1.2 Normalization

Normalization is a rescaling of the data from the original range so that all values are within the new range of 0 and 1.

Normalization requires that you know or are able to accurately estimate the minimum and maximum observable values.

Formula : $y = (x - \min) / (\max - \min)$

You can normalize your dataset using the scikit-learn object `MinMaxScaler`.

6.1.3 Autocorrect Module

Some words will sometimes have mistakes, especially when someone is learning the language for the first time.

We would hate to require the person to completely rewrite their word from scratch every time they do a mistake.

So, much like when we are typing using a keyboard, we will correct the word they were trying to spell to the closest estimated word.

6.1.4 Detection Module

We utilized the mediapipe library to find hands in the frame since it not only finds hands but also provides 22 3D coordinates of the complete hand, providing us a wealth of information to work with.

We then used this hand-detection model to extract the coordinate data from an ASL alphabet dataset.

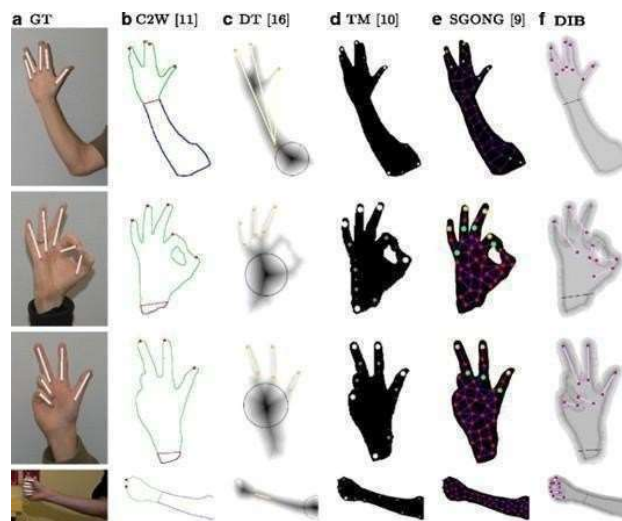


Fig 6.1.4 : Detection Module

CHAPTER 7

SYSTEM TESTING

7. SYSTEM TESTING

7.1 BLACK BOX TESTING

Black box testing is a software testing technique that focuses on testing the functionality of a software system without knowing the internal workings of the system. In this type of testing, the tester is not concerned with the code, architecture, or design of the system but instead focuses on the inputs and outputs of the system. The goal of black box testing is to determine whether the software system behaves correctly based on its specifications, requirements, and business logic. The tester typically creates test cases that simulate different scenarios and inputs to verify the expected results. Black box testing can be performed at different levels of software testing, including unit testing, integration testing, system testing, and acceptance testing. It can be automated or performed manually. The advantages of black box testing include the ability to test the software system from the end user's perspective, the ability to detect defects that may be missed in other types of testing, and the fact that it does not require knowledge of the internal workings of the system. The disadvantages of black box testing include the possibility of incomplete testing due to the limited knowledge of the system, and the fact that it may be difficult to determine the root cause of defects.

7.2 WHITE BOX TESTING

White box testing is a software testing technique that focuses on testing the internal workings of a software system. In this type of testing, the tester has access to the source code, architecture, and design of the system and uses this knowledge to create test cases that verify the correctness and quality of the system's implementation. The goal of white box testing is to ensure that the code is written correctly, follows best practices and coding standards, and meets the design specifications. The tester typically creates test cases that target specific areas of the code, such as loops, conditionals, and error handling, to ensure that all possible scenarios have been tested. White box testing can be performed at different levels of software testing, including unit testing, integration testing, and system testing. It can be automated or performed manually. The advantages of white box testing include the ability to test the system thoroughly and ensure that all code paths have been exercised, the ability to detect

defects that may be missed in other types of testing, and the ability to optimize the code for performance and efficiency.

7.3 TEST CASES

TEST REPORT: 01

PRODUCT : Doing Hand Signs To Recognize The Word

USE CASE : Do Hand Signs

TEST CASE ID	TESTCASE / ACTION TO BE PERFORMED	EXPECTED RESULT	ACTUAL RESULT	PASS/FAIL
1	Do Hand Signs	Letters Showing For The Hand Signs	Letters Showing For The Hand Signs	PASS
2	Do Hand Signs	Extra Words Are Printed	Extra Words Are Printed	PASS

Table-7.3.1 Test Case For Hand Signs

TEST REPORT 02

PRODUCT : Doing Hand Signs To Recognize The Word

USE CASE : Do Hand Signs

TEST CASE ID	TESTCASE/ ACTION TO BE PERFORMED	EXPECTED RESULT	ACTUAL RESULT	PASS/FAIL
1	Do Some Hand Signs	Word Pop Ups	Word Pop Ups	PASS
2	Do Hand Signs	Two Hands Detected Multiple Words	Two Hands Detected Multiple Words	PASS
3	Do Hand Signs	Wrong Word Detected	Wrong Word Detected	PASS
4	Do Hand Signs	AutoCorrected Word Diplayed	AutoCorrect ed Word Diplayed	PASS

Table-7.3.2 Test Case For Hand Signs

CHAPTER 8

CONCLUSION & FUTURE ENHANCEMENT

8. CONCLUSION & FUTURE ENHANCEMENT

8.1 CONCLUSION

Nowadays, applications need several kinds of images as sources of information for elucidation and analysis. Several features are to be extracted so as to perform various applications. When an image is transformed from one form to another such as digitizing, scanning, and communicating, storing, etc. degradation occurs. Therefore, the output image has to undertake a process called image enhancement, which contains of a group of methods that seek to develop the visual presence of an image. Image enhancement is fundamentally enlightening the interpretability or awareness of information in images for human listeners and providing better input for other automatic image processing systems. Image then undergoes feature extraction using various methods to make the image more readable by the computer. Sign language recognition system is a powerful tool to prepare an expert knowledge, edge detect and the combination of inaccurate information from different sources. The intend of convolution neural network is to get the appropriate classification

8.2 FUTURE ENHANCEMENT

The proposed sign language recognition system used to recognize sign language letters can be further extended to recognize gestures facial expressions. Instead of displaying letter labels it will be more appropriate to display sentences as more appropriate translation of language. This also increases readability. The scope of different sign languages can be increased. More training data can be added to detect the letter with more accuracy. This project can further be extended to convert the signs to speech

APPENDICES

APPENDICES

A.1 CODING

test.py

```
import cv2
import mediapipe as mp
import joblib
from sklearn.preprocessing import MinMaxScaler
ml_model = joblib.load('models/rf_model.joblib')
mp_drawing = mp.solutions.drawing_utils
mp_drawing_styles = mp.solutions.drawing_styles
mp_hands = mp.solutions.hands
avg = 0

count = 1
# For webcam input:

cap = cv2.VideoCapture(1)
with mp_hands.Hands(
    min_detection_confidence=0.5,
    min_tracking_confidence=0.5) as hands:
    while cap.isOpened():
        success, image = cap.read()
        if not success:
            print("Ignoring empty camera frame.")

        # If loading a video, use 'break' instead of 'continue'.
        continue

        # Flip the image horizontally for a later selfie-view display, and convert
        # the BGR image to RGB.
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
```

```

# To improve performance, optionally mark the image as not writeable to #
pass byreference.
image.flags.writeable = False results = hands.process(image)

# Draw the hand annotations on the image. image.flags.writeable = True
image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR) 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_styl
e(),#
    mp_drawing_styles.get_default_hand_connections_style())

scaler = MinMaxScaler()

originX = hand_landmarks.landmark[0].x originY = hand_landmarks.landmark[0].y originZ
= hand_landmarks.landmark[0].z

landmarkers = []

for landmarker in hand_landmarks.landmark: landmarker.x -= originX
landmarker.y -= originY landmarker.z -= originZ
landmarkers.append([landmarker.x, landmarker.y, landmarker.z])

scaled = scaler.fit_transform(landmarkers)

```

```

output = []

for coord in scaled: output.append(coord[0]) output.append(coord[1])
output.append(coord[2])

letter = ml_model.predict([output])[0] print(letter)

cv2.imshow('MediaPipe Hands', image) if cv2.waitKey(5) & 0xFF
== 27:break cap.release()

```

web.py

```

from sklearn.preprocessing import MinMaxScaler from spellchecker import
SpellCheckerfrom demos import orchestrator, fig from pathlib import Path
import streamlit as st import mediapipe as mp import joblib
import random import cv2 import sys

spell = SpellChecker()

ml_model = joblib.load('rf_model.joblib') # Load image model

# Mediapipe hand utils

mp_drawing = mp.solutions.drawing_utils
mp_drawing_styles =mp.solutions.drawing_styles
mp_hands = mp.solutions.hands

```

```

file = Path( file ).resolve()

parent, root = file.parent, file.parents[1] sys.path.append(str(root))

try:

sys.path.remove(str(parent))

except ValueError: # Already removed pass


VERSION = ".".join(st. version .split(".")[2:]) demo_pages = {

"Method overview": orchestrator.show_examples, #"ASL table": fig.show_examples
}

avg = 0

count = 1

def draw_main_page():
st.write(f"""

# LiveSigns  """)
)

current_cam = 1

# st.write("Try switching to `cam 0` if you're on mac, and `cam 1` if windows.")

cam_switch = st.checkbox('Switch camera input: (try using "Camera #1" on Mac,
"Camera#2" on Windows)')

if cam_switch:

current_cam = 1-current_cam

```

```

st.text('Current Camera: #' + str(current_cam + 1))

st.write('Click `Run` to try out the live sign language translation. The ASL sign table
can be found on the left; the `del` sign is used before entering a duplicate symbol;
`space` is used to end a word.')

st.write('(Hold up each symbol for at least one second for best results.)')

run = st.checkbox('Run')
landmark_on = st.checkbox('Overlay landmarks')


FRAME_WINDOW = st.image([]) cap = cv2.VideoCapture(current_cam)


hand_word = "" prev_letter = "" freq_letter = 0 displayed = False start_new_word =
False auto_corrected = 0


while run:

    with mp_hands.Hands( min_detection_confidence=0.5,
min_tracking_confidence=0.5) as hands:
        while cap.isOpened(): success, image = cap.read() if not
success: print("Ignoring empty camera frame.") continue

        width, height = int(cap.get(3)), int(cap.get(4))

        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

        # image = cv2.cvtColor(cv2.flip(image, 1), cv2.COLOR_BGR2RGB)


        image.flags.writeable = False

        font = cv2.FONT_HERSHEY_SIMPLEX # type: ignore results = hands.process(image)

```

```

image.flags.writeable = True

image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR) if
results.multi_hand_landmarks:hand_landmarks, handedness =
results.multi_hand_landmarks[0], results.multi_handedness[0]

if(landmark_on): 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())

# add text

textX = int(hand_landmarks.landmark[0].x * width) - 150 # type: ignore
textY =int(hand_landmarks.landmark[0].y * height) - 100 # type: ignore

# calculate offset for right hand

if handedness.classification[0].label == 'Right': textX += 250

scaler = MinMaxScaler()

originX = hand_landmarks.landmark[0].x originY = hand_landmarks.landmark[0].y originZ
= hand_landmarks.landmark[0].z

landmarkers = []

for landmarker in hand_landmarks.landmark: landmarker.x -= originX
landmarker.y -= originY landmarker.z -= originZ

```



```

landmarkers.append([landmarker.x, landmarker.y, landmarker.z])
scaled=scaler.fit_transform(landmarkers)
output = []

for coord in scaled: output.append(coord[0]) output.append(coord[1])
output.append(coord[2])

letter = ml_model.predict([output])[0]

if letter == 'del': prev_letter = "
elif letter == prev_letter: freq_letter
+= 1 if freq_letter > 5 and not
displayed:

if letter == 'space' and not start_new_word: start_new_word =
True
misspelled = list(spell.unknown([hand_word]))

if len(misspelled) != 0:

hand_word = spell.correction(misspelled[0]) auto_corrected
= 1 else:

auto_corrected = 2

elif start_new_word: start_new_word = False auto_corrected = 0 hand_word =
letter
else:

hand_word += letter

```

```

displayed = not displayed elif letter != prev_letter:
# hand_word += letter freq_letter = 0 displayed = False

prev_letter = letter

# if letter == 'space':

# image = cv2.putText(image, " ", # type: ignore # (textX,
textY),# font, 3, (0, 0, 0), 5, cv2.LINE_AA # type: ignore
# )

# else:

display_letter = letter if letter != 'del' else 'next' image = cv2.putText(image, letter, #
type:ignore
(textX, textY),
font, 3, (0, 0, 0), 5, cv2.LINE_AA # type: ignore
)

text_size, _ = cv2.getTextSize(hand_word, font, 3, 2) # type: ignore image =
cv2.rectangle( #type: ignore
image,
(int((width - text_size[0]) / 2), height - 100 - text_size[1] - 30), (int((width +
text_size[0]) / 2), height - 130 + text_size[1]),
(0, 0, 0), -1)
display_word = hand_word if auto_corrected == 1:
cv2.putText( # type: ignore
image,display_word,
(int((width - text_size[0]) / 2), height - 100),

```

```

font, 3, (0, 255, 255), 2, cv2.LINE_AA) # type: ignore elif auto_corrected == 2: # if
correct!cv2.putText( # type: ignore image,
display_word,

(int((width - text_size[0]) / 2), height - 100),

font, 3, (0, 255, 0), 2, cv2.LINE_AA) # type:

ignoreelse:

cv2.putText( # type: ignore
image,display_word,

(int((width - text_size[0]) / 2), height - 100),

font, 3, (255, 255, 255), 2, cv2.LINE_AA) # type: ignore

FRAME_WINDOW.image(cv2.cvtColor(image,
cv2.COLOR_RGB2BGR)) # ifcv2.waitKey(5) & 0xFF == 27:
#
break
else:

cap.release()
cv2.destroyAllWindows()
#st.write(release_notes)
# Draw sidebar

pages = list(demo_pages.keys())

if len(pages):

pages.insert(0, "LiveSigns")
st.sidebar.title(f"ASLRecognition ")
query_params =
st.experimental_get_query_params()

```

```

if "page" in query_params and query_params["page"][0] == "headliner": index
= 1 else:

index = 0

selected_demo = st.sidebar.radio("", pages, index,
key="pages") else:selected_demo = ""

st.sidebar.image("pic/asl.png", use_column_width=True)


# Draw main page

if selected_demo in demo_pages:
demo_pages[selected_demo]()else:

draw_main_page()

```

web2.py

```

from sklearn.preprocessing import MinMaxScaler from spellchecker import
SpellCheckerfrom demos import orchestrator, fig from pathlib import Path
import streamlit as st import mediapipe as mp import joblib
import random import cv2 import sys


from streamlit_webrtc import VideoTransformerBase, webrtc_streamer


spell = SpellChecker()

ml_model = joblib.load('rf_model.joblib') # Load image model


st.set_page_config(layout='wide')

```

```

# Mediapipe hand utils

mp_drawing = mp.solutions.drawing_utils
mp_drawing_styles = mp.solutions.drawing_styles
mp_hands = mp.solutions.hands

file = Path(file_path).resolve()

parent, root = file.parent, file.parents[1]
sys.path.append(str(root))

try:
    sys.path.remove(str(parent))
except ValueError: # Already removed pass

VERSION = ".".join(sys.version.split(".")[2:])
demo_pages = {
    "Method overview": orchestrator.show_examples,
    #"ASL table": fig.show_examples
}

avg = 0

count = 1

class VideoTransformer(VideoTransformerBase):
    def __init__(self):
        self.hand_word = ""
        self.prev_letter = ""
        self.freq_letter = 0
        self.displayed = False
        self.start_new_word = False
        self.auto_corrected = 0
        # self.hands = mp_hands.Hands(

```

```

# min_detection_confidence=0.5, # min_tracking_confidence=0.5)

def transform(self, frame):

    with mp_hands.Hands(min_detection_confidence=0.5,
        min_tracking_confidence=0.5) as hands: image =
        frame.to_ndarray(format="bgr24")
        width, height, _ = image.shape

    # image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

    image = cv2.cvtColor(cv2.flip(image, 1), cv2.COLOR_BGR2RGB)

    image.flags.writeable = False

    font = cv2.FONT_HERSHEY_SIMPLEX # type: ignore results = hands.process(image)
    image.flags.writeable = True

    image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR) if
    results.multi_hand_landmarks:hand_landmarks, handedness =
    results.multi_hand_landmarks[0], results.multi_handedness[0]

    # 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()

    # add text

    textX = int(hand_landmarks.landmark[0].x * width) + 150 # type: ignore
    textY =int(hand_landmarks.landmark[0].y * height) - 150 # type: ignore

    # calculate offset for right hand

    if handedness.classification[0].label == 'Right':

```

```
textX += 100
```

```
scaler = MinMaxScaler()
```

```
originX = hand_landmarks.landmark[0].x originY = hand_landmarks.landmark[0].y originZ  
= hand_landmarks.landmark[0].z
```

```
landmarkers = []
```

```
for landmarker in hand_landmarks.landmark: landmarker.x -=
```

```
originX landmarker.y -= originY landmarker.z -= originZ
```

```
landmarkers.append([landmarker.x, landmarker.y, landmarker.z])
```

```
scaled = scaler.fit_transform(landmarkers)
```

```
output = []
```

```
for coord in scaled: output.append(coord[0]) output.append(coord[1])
```

```
output.append(coord[2])
```

```
letter = ml_model.predict([output])[0]
```

```
if letter == 'del': self.prev_letter = "
```

```
elif letter == self.prev_letter: self.freq_letter
```

```
+= 1 if self.freq_letter > 10 and not
```

```
self.displayed:
```

```
if letter == 'space' and not self.start_new_word: self.start_new_word = True
```

```
misspelled = list(spell.unknown([self.hand_word]))
```

```

if len(misspelled) != 0:

    self.hand_word = spell.correction(misspelled[0]) self.auto_corrected
    = 1 else:

    self.auto_corrected = 2


elif self.start_new_word: self.start_new_word = False self.auto_corrected = 0 self.hand_word
=
letter
else:

self.hand_word += letter


self.displayed = not self.displayed elif letter !=
self.prev_letter: # hand_word += letter self.freq_letter = 0
self.displayed = False
self.prev_letter = letter

# if letter == 'space':

# image = cv2.putText(image, " ", # type: ignore # (textX,
textY),# font, 3, (0, 0, 0), 5, cv2.LINE_AA # type: ignore
# )

# else:

display_letter = letter if letter != 'del' else 'next' image = cv2.putText(image, letter, #
type:ignore
(textX, textY),

font, 2, (0, 0, 0), 5, cv2.LINE_AA # type: ignore
)

text_size, _ = cv2.getTextSize(self.hand_word, font, 2, 2) # type: ignore

```



```

image = cv2.rectangle( # type: ignore image,
(int((height - text_size[0]) / 2), width - 60 - text_size[1] - 10), (int((height +
text_size[0]) / 2),width - 60 + 10),
(0, 0, 0), -1)
display_word = self.hand_word if self.auto_corrected == 1:

cv2.putText( # type: ignore
image,display_word,

(int((height - text_size[0]) / 2), width - 60),

font, 2, (0, 255, 255), 2, cv2.LINE_AA) # type: ignore elif self.auto_corrected == 2: #
ifcorrect!
cv2.putText( # type: ignore
image,display_word,

(int((height - text_size[0]) / 2), width - 60),

font, 2, (0, 255, 0), 2, cv2.LINE_AA) # type:
ignoreelse:


cv2.putText( # type: ignore
image,display_word,

(int((height - text_size[0]) / 2), width - 60),

font, 2, (255, 255, 255), 2, cv2.LINE_AA) # type: ignore

return image
def draw_main_page():

st.write( f"""
# LiveSigns



st.write('Click `Run` to try out the live sign language translation. The ASL sign table
can be found on the left; the `del` sign is used before entering a duplicate symbol;
`space` is used to end a word.')
```

```

st.write('(Hold up each symbol for at least one second for best results.)') run
=st.checkbox('Run')

# FRAME_WINDOW = st.image([])

# cap = cv2.VideoCapture(current_cam)

if run:

    webrtc_streamer(key="special_key_or_somethin
g",
video_transformer_factory=VideoTransformer)
#st.write(release_notes)# Draw sidebar

pages = list(demo_pages.keys())

if len(pages):

    pages.insert(0, "LiveSigns")

    st.sidebar.title(f'ASLRecognition ")

    query_params =
st.experimental_get_query_params()

    if "page" in query_params and query_params["page"][0] == "headliner": index
= 1 else:

        index = 0

    selected_demo = st.sidebar.radio("", pages, index,
key="pages") else:selected_demo = ""

    st.sidebar.image("pic/asl.png", use_column_width=True)

# Draw main page
if selected_demo in demo_pages:
    demo_pages[selected_demo]()else:
    draw_main_page(

```

A.2 SAMPLE SCREENS

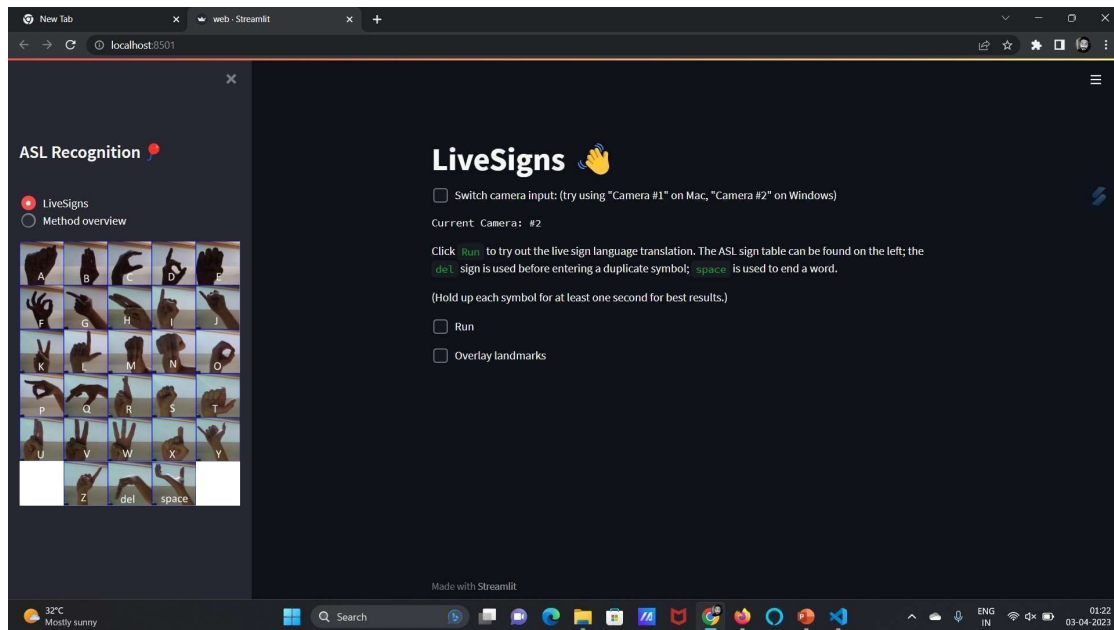


Fig A2.1 : Home Screen Of The Application

In Fig A2.1 The Home Page of The Screen is displayed and then we should enable the checkbox to run and show hand signs.

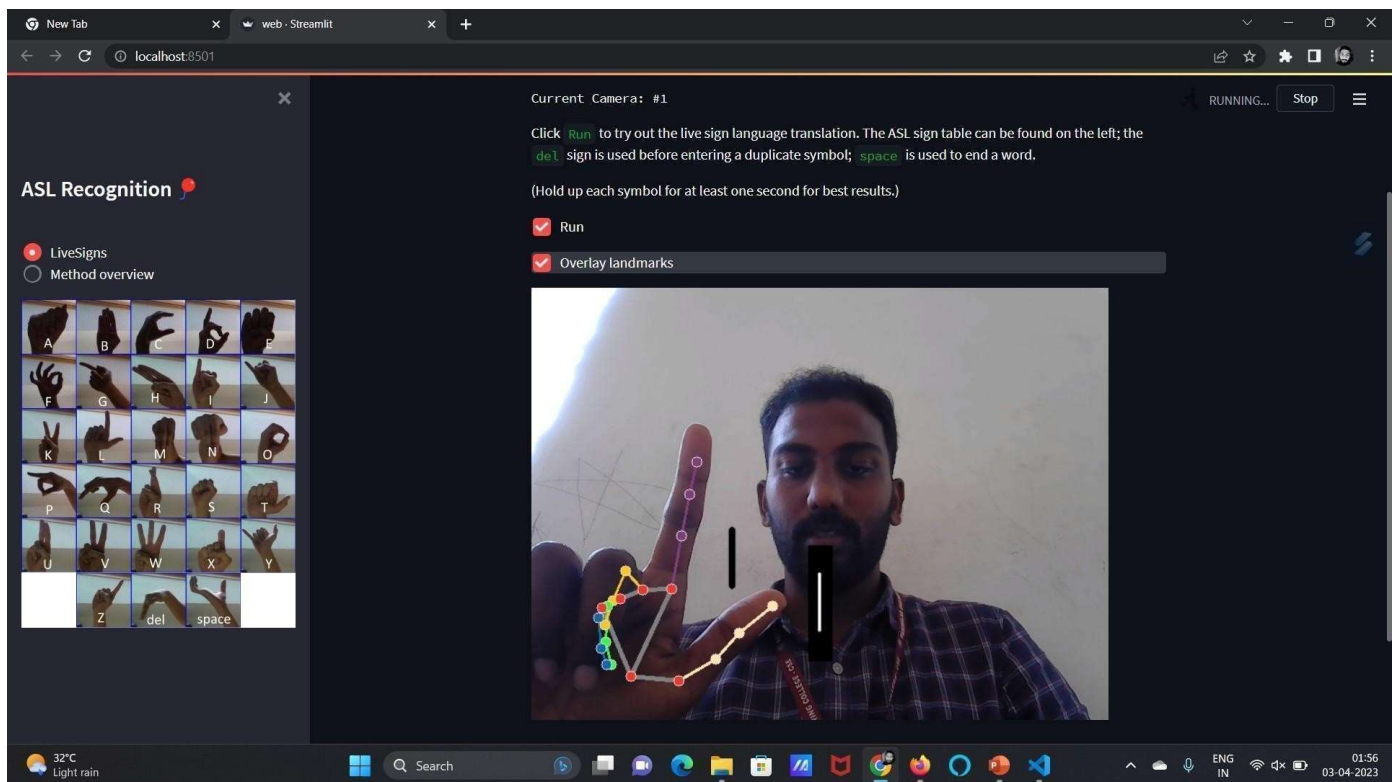


Fig A2.2 : Overlay Landmarks On Hand Signs

In Fig A2.2 The camera is Turned On and When we give the Hand Sign for Specific Character and it Recognizes and Shows the Output Character

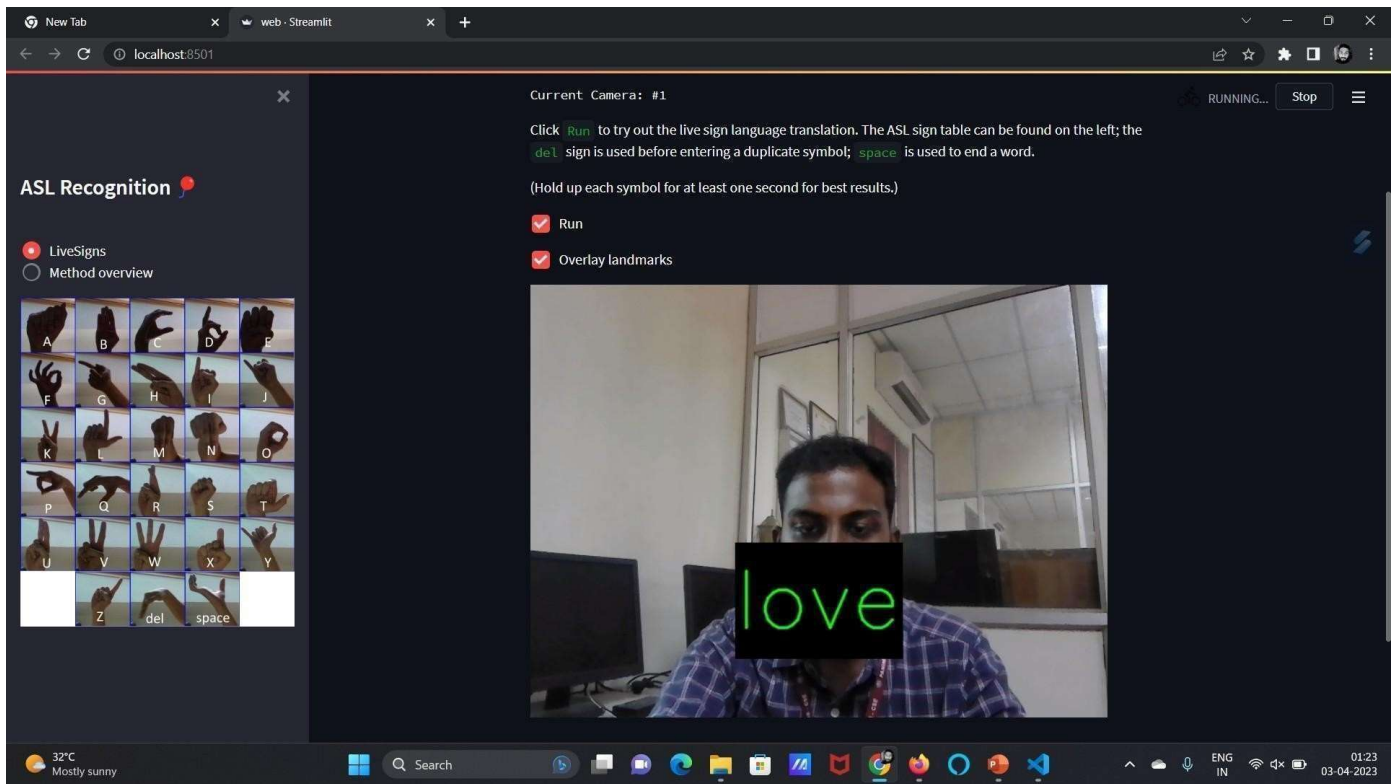


Fig A2.3 : Generated Word From Hand Sign

In Fig A2.3 It Shows the Letter Love and Each Hand Sign is Made for Each Character and Made into a Word for the output.

REFERENCES

REFERENCES

- [1] A. Z. Shukor, M. F. Miskon, M. H. Jamaluddin, F. bin Ali, M. F. Asyraf, M. B. bin Bahar and others. (2015) "A new data glove approach for Malaysian sign language detection,"
Procedia Computer Science, vol. 76, p. 60–67
- [2] M. Mohandes, M. Deriche and J. Liu. (2014) "Image-based and sensor-based approaches to Arabic sign language recognition," IEEE transactions on human-machine systems, vol. 44, p. 551–557.
- [3] N. M. Kakoty and M. D. Sharma. (2018) "Recognition of sign language alphabets and numbers based on hand kinematics using a data glove," Procedia Computer Science, vol.133, p. 55–62.
- [4] Continuous dynamic Indian Sign Language gesture recognition with invariant backgrounds by Kumud Tripathi, Neha Baranwal, G. C. Nandi at 2015 Conference on Advances in Computing, Communications and Informatics (ICACCI)
- [5] Bernard Boulay, Francois Bremond, Monique Thonat, Human Posture Recognition in Video Sequence. IEEE International Workshop on VSPETS, Visual Surveillance and Performance Evaluation of Tracking and Surveillance, 2003, Nice, France.
- [6] Z. A. Ansari and G. Harit. (2016) "Nearest neighbour classification of Indian sign language gestures using kinect camera," Sadhana, vol. 41, p. 161–182.
- [7] A. Das, S. Gawde, K. Suratwala and D. Kalbande. (2018) "Sign language recognition using deep learning on custom processed static gesture images," in International Conference on Smart City and Emerging Technology (ICSCET).
- [8] C. Lugaresi, J. Tang, H. Nash, C. McClanahan, E. Uboweja, M. Hays, F. Zhang, C.-L. Chang, M. G. Yong, J. Lee and others. (2019) "Mediapipe: A framework for building perception pipelines," arXiv preprint arXiv:1906.08172.
- [9] Long Short-Term Memory, Sepp Hochreiter et al., Neural Computation 9(8): 1735- 1780, 1997
- [10] A. K. Sahoo. (2021) "Indian sign language recognition using machine learning techniques," in Macromolecular Symposia.

- [11] Cui, R., Liu, H., Zhang, C.: A deep neural framework for continuous sign language recognition by iterative training. *IEEE Trans.Multimedia* 21(7), 1880–1891 (2019). <https://doi.org/10.1109/TMM.2018.2889563>
- [12] Sharma, S., Gupta, R., Kumar, A.: Continuous sign language recognition using isolated signs data and deep transfer learning. *J.Ambient Intell. Humaniz. Comput.* (2021). <https://doi.org/10.1007/s12652-021-03418-z>
- [13] Yin, K., Read, J.: Attention is all you sign: sign language translation with transformers. In: *European Conference on Computer Vision Work on Sign Language Recognition, Translation and Production*, pp. 1–4 (2020)
- [14] Camgoz, N.C., Hadfield, S., Koller, O., Ney, H., Bowden, R.: Neural sign language translation. In: *Proceedings of IEEE Computer Society Conference on Computer Vision Pattern Recognition*, pp. 7784–7793 (2018). <https://doi.org/10.1109/CVPR.2018.00812>
- [15] Forster, J., et al.: RWTH-PHOENIX-weather: a large vocabulary sign language recognition and translation corpus. In: *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC' 2012)*, pp. 3785–3789 (2012)