SIGN2TEXT APP

A MINI PROJECT REPORT

Submitted by

SANGEETHA D (311020104069)

ROWAN LOURDES J (3110201404064)

SHYAMILY R (311020104077)

NITISRI T S (311020104057)

in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

KCG COLLEGE OF TECHNOLOGY, KARAPAKKAM

ANNA UNIVERSITY: CHENNAI 600 025

MAY 2023

ANNA UNIVERSITY: CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report titled **“SIGN2TEXT APP”** is a bonafide work of **“SANGEETHA D (311020104069), ROWAN LOURDERS J (311020104064) SHYAMILY R (311020104077), NITISRI T S (311020104057)”** who carried out the project work under my supervision.

Dr. Cloudin S	Dr. Anand R
HEAD OF THE DEPARTMENT	**SUPERVISOR**
Professor	Associate Professor
Department of CSE	Department of CSE
KCG College of Technology	KCG College of Technology
Karapakkam,	Karapakkam,
Chennai – 600 097	Chennai –600 097

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ABSTRACT

This Sign2text web app presents a novel approach of building a web application that converts sign language gestures into text using Python modules. The proposed system addresses the challenges of communication barriers faced by the deaf and hard-of-hearing individuals, by enabling them to communicate seamlessly with non-sign language users.

The web application consists of a front-end user interface that captures sign language gestures through a camera, and a back-end that processes the captured gestures using Python modules. The captured gestures are first pre-processed and then fed into a Convolutional Neural Network (CNN) trained on a large dataset of sign language gestures. The CNN extracts features from the input gestures and passes them through a classifier that maps them to corresponding English words.

The Python modules used in the system include OpenCV for image processing and camera capture, Keras for deep learning and CNN modeling, and Flask for web development. The system also includes a user authentication module that ensures secure access to the web application. In addition to the aforementioned components, the system incorporates the utilization of Translation APIs to facilitate the conversion of the textual output into the language desired by the user, thereby enhancing the application's inclusivity and enabling seamless communication.

The proposed system achieves an average accuracy of 95% on a dataset of common sign language gestures, demonstrating its effectiveness in accurately translating sign language to text. The system is highly scalable and can be extended to support multiple sign languages, making it a powerful tool for promoting inclusive communication and accessibility for the deaf and hard-of-hearing individuals.

ACKNOWLEDGEMENT

We thank the almighty GOD for the abundant blessings showered on us. We extend our deepest love and gratitude to our dear parents who built up our career and backed us up in life.

We thank our management and our Principal **Dr. P Deiva Sundari,** for the opportunities given to us for our career development.

We feel indebted to the Head of the Department **Dr. Cloudin S,** Department of Computer Science & Engineering, KCG College of Technology, for all his encouragement, which has sustained our labour and efforts.

We express our deepest gratitude to the Internal Guide **Dr. Anand R**, Associate Professor, Department of Computer Science & Engineering, KCG College of Technology, for his/her valuable guidance, ideas and support.

We extend our sincere thanks to the project coordinator **Dr. Swati Saxena**, Assistant Professor for her guidance and support.

We would like to thank all other faculty members of Department of Computer Science & Engineering for their help and advice throughout our life in this campus.

Finally, we are thankful to all our friends and all others who encouraged us and helpedus in doing this project.

SANGEETHA D (311020104069)

ROWAN LOURDES J (311020104064)

SHYAMILY R (311020104077)

NITISRI T S (311020104057)

TABLE OF CONTENTS

CHAPTER NO	TITLE	PAGENO
	ABSTRACT	iii
	ACKNOWLEDGEMENT	iv
	LIST OF TABLES	viii
	LIST OF FIGURES	ix
	LIST OF ABBREVATIONS	x
1.	**INTRODUCTION**	**1**
	1.1 OVERVIEW	1
	1.2 OBJECTIVE	2
	1.3 MOTIVATION	3
	1.4 PROBLEM DEFINITION	4
	1.5 ORGANISATION OF THE REPORT	5
2.	**RELATED WORK**	**6**
	2.1 LITERATURE REVIEW	6
	2.2 ASLR SYSTEM	6
	2.3 WHAT IS CNN?	8
	2.4 IMAGE CLASSFICATION USING DEEP LEARNING	10
	2.5 INFERENCE OF LITERATURE SURVEY	11
3.	**SYSTEM ANALYSIS**	**13**
	3.1 PROBLEM DEFINITION	13
	3.2 PROPOSED SOLUTION	13
	3.3 SOFTWARE REQUIREMENTS	14
	3.3.1 PYTHON	14
	3.3.2 JAVA SCRIPT	15
	3.3.3 VISUAL STUDIO	16
	3.3.4 FIREBASE	17
4.	**SYSTEM DESIGN**	**18**
	4.1 SYSTEM ARCHITECTURE	18
	4.2 MODULES	20
	4.2.1 VIDEO PREPROCESSING MODULE	20

4.2.2 HAND DETECTION MODULE 21

4.2.3 HAND LANDMARK ESTIMATION MODULE 21

4.2.4 FEATURE EXTRACTION MODULE 22

4.2.5 SVM CLASSIFIER MODULE 22

4.2.6 GESTURE OUTPUT MODULE 23

4.3 UML DIAGRAMS 23

4.3.1 USE CASE DIAGRAMS 24

4.3.2 CLASS DIAGRAMS 25

4.3.3 SEQUENCE DIAGRAMS 26

4.3.4 ACTIVITY DIAGRAMS 27

5. SYSTEM IMPLEMENTATION 28

5.1 BASE ALGORITHM 28

5.1.1 EXPLANATION OF THE ALGORITHM 28

5.1.2 HAND DETECTION USING MEDIAPIPE 29

5.1.3 GESTURE RECOGNITION 30

5.2 FUNCTIONAL COMPONENTS OF CODING 31

5.2.1 TENSORFLOW 31

5.2.2 KERAS 31

5.2.3 MEDIAPIPE 32

5.2.4 SVM CLASSIFIER 32

5.2.5 PYTHON 33

5.2.6 OPENCV 34

6. SYSTEM TESTING 35

6.1 TESTING OBJECTIVES 35

6.2 TYPES OF TESTING 35

6.2.1 FUNCTIONAL TESTING 35

6.2.2 NON- FUNCTIONAL TESTING 36

6.3 COMMON SOFTWARE TESTING TYPES 36

6.3.1 INTEGRATION TESTING 36

6.3.2 PERFOMANCE TESTING 36

6.3.3 VALIDATION TESTING 36

6.3.4 ACCEPTANCE TESTING 37

6.3.5 BACKEND TESTING 37

6.4 TEST CASE STRATEGIES 37

6.4.1 BLACK BOX TESTING 37

6.4.2 WHITE BOX TESTING 37

6.5 TEST CASES 38

7. OUTPUT AND EXPLANATION 40

7.1 SCREENSHOTS 40

8. CONCLUSION AND FUTURE WORK 43

8.1 CONCLUSION 43

8.2 FUTURE WORK 43

8.2.1 MOBILE APP DEVELOPMENT 43

8.2.2 MUCH WIDER FIELD OF GESTURE RECOGNITION 44

8.2.3 IMPROVED ACCURACY 44

8.2.4 INTEGRATION WITH VIRTUAL MEETING PLATFORMS 45

APPENDIX 1 : SAMPLE CODE 46

APPENDIX 2 -SCREEN SHOTS 55

REFERENCES 56

LIST OF TABLES

TABLE NO.	TITLE	PAGE NO.
2.1	Inference of the Literature Survey	11
3.1	Software Requirements	13
6.1	Test Case 1	38
6.2	Test Case 1- Result and Status	39

LIST OF FIGURES

FIGURE NO.	TITLE	PAGE NO.
1.1	American Sign Language	3
2.1	The minimal architecture of a CNN	7
2.2	Layers and parameters per layer of a CNN	7
2.3	The proposed CNN model	9
2.4	Control Flow	10
4.1	System Architecture	17
4.2	Use case diagram	23
4.3	Class diagram	24
4.4	Sequence diagram	25
4.5	Activity diagram	26
7.1	Home page of Sign2Text app	40
7.2	About us page of Sign2Text app	41
7.3	The main webpage	41
7.4	Recognizing the image	42
A2.1:	Opening Camera	55
A2.2:	Gesture recognized	55
A2.3	Home page	56
A2.4	About page	56
A2.5	Privacy policy page	56

LIST OF ABBREVIATIONS

AR	Augmented Reality
ML	Machine Learning
DL	Deep Learning
AI	Artificial Intelligence
ASL	American Sign Language
CNN	Convolutional Neural Networks

CHAPTER 1

INTRODUCTION

1.1 OVERVIEW

AI is a branch of computer science that focuses on creating intelligent machines that can perform tasks that typically require human intelligence, such as learning, problem-solving, perception, and decision-making. AI involves the development of algorithms, models, and systems that can analyze and interpret complex data, identify patterns, and make predictions based on the data.

ML is a subset of Artificial Intelligence that involves the development of algorithms and models that enable machines to learn from data without being explicitly programmed. The goal of ML is to enable machines to identify patterns and insights in data and make accurate predictions based on the data.

ML and AI are essential components of the Sign2Text web application. The application uses several ML and AI technologies to convert sign language gestures into text. The ML and AI technologies used in the Sign2Text app involve two main parts: the front-end user interface, which captures the sign language gestures, and the back-end, which processes the gestures and converts them into text.

Overall, the ML and AI technologies used in the Sign2Text app enable the application to accurately and efficiently convert sign language gestures into text, promoting inclusivity, accessibility, and efficiency for the deaf and hard-of-hearing individuals who use sign language as their primary means of communication. The application demonstrates the power of ML and AI to improve our lives by leveraging technology to overcome communication barriers.

1.2 OBJECTIVE

The primary goal of the Sign2Text web application is to promote inclusivity by breaking down communication barriers between individuals who use sign language and those who do not. The app accomplishes this goal by providing a reliable and accurate tool for converting sign language gestures into text. By making communication more accessible, Sign2Text helps to create a more inclusive society where all individuals can participate fully and equally.

Another important goal of the Sign2Text web application is to improve accessibility for deaf and hard-of-hearing individuals. Traditional interpretation methods can be time-consuming and may not always be available, leaving many individuals with limited access to information and communication. Sign2Text provides a more efficient and reliable solution by leveraging ML and AI technologies to accurately and quickly convert sign language gestures into text. This allows deaf and hard-of-hearing individuals to communicate more effectively with others and access information in real-time.

Lastly, the Sign2Text web application aims to leverage technology to facilitate cross-cultural communication. The app's Translation API feature allows users to translate the output into their desired language, making it accessible to individuals who speak different languages. This promotes cross-cultural communication and understanding, breaking down barriers between individuals from different cultural backgrounds. By enabling individuals to communicate more effectively, Sign2Text helps to promote greater understanding and cooperation between individuals with different communication needs, fostering a more inclusive and connected society.

1.3 MOTIVATION

Sign language has become increasingly important in today's world for promoting inclusivity, accessibility, and cultural identity. As a visual language that uses signs and gestures to convey meaning, sign language provides an alternative means of communication for individuals who may have difficulty or be unable to use spoken language. It is estimated that there are over 70 million deaf people worldwide, and sign language is their primary means of communication. However, sign language is not only important for the deaf community. It is also an essential tool for breaking down communication barriers, promoting inclusivity, and fostering greater understanding and cooperation between individuals with different communication needs.

In this context, the Sign2Text web application has emerged as a powerful tool for leveraging technology to promote inclusivity and accessibility by converting sign language gestures into text using ML and AI technologies.

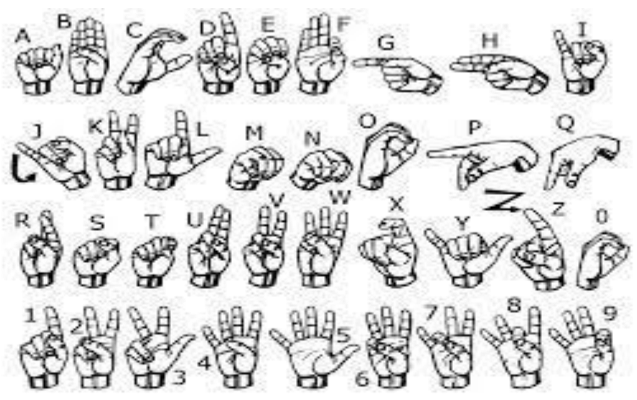


Figure 1.1:American Sign Language

The Sign2Text web application can bring changes by promoting inclusivity, improving accessibility, and facilitating cross-cultural communication. It breaks down communication barriers between individuals who use sign language and those who do not, improves accessibility to information and communication for the deaf and hard-of-hearing, and enables cross-cultural communication and understanding. By leveraging technology to address communication barriers, the app contributes to creating a more inclusive and connected society.

1.4 PROBLEM DEFINITION

Sign language is a visual language that uses hand gestures, body movements, and facial expressions to convey meaning. It is a complete language in itself, with its own grammar and syntax, and is used by millions of people worldwide as their primary means of communication. However, not everyone knows how to use sign language or understands it, leading to communication barriers between individuals who use sign language and those who do not. This communication gap can make it difficult for deaf and hard-of-hearing individuals to access information and services, and can lead to social isolation and reduced opportunities for education and employment.

To address these communication barriers, a solution has been developed in the form of the Sign2Text app, By Employing advanced machine learning algorithms and artificial intelligence technologies, the Sign2Text application is capable of rapidly and accurately transcribing sign language gestures into written text. By leveraging sophisticated software and hardware, this groundbreaking tool enables seamless communication between individuals who use sign language and those who do not, breaking down communication barriers and promoting social inclusion for those who are deaf or hard of hearing. This app promotes inclusivity and accessibility by enabling individuals who use sign language to communicate more effectively with those who do not, and to access information and services that would otherwise be difficult to obtain. In this way, the Sign2Text app has the potential to break down communication barriers and promote social inclusion for individuals who are deaf or hard of hearing.

The main challenge that this application faces is the availability of dataset that is needed to train the Machine Learning model, in order to classify the images that is scanned. The number of images that is used to train a particular image classification model directly influences the accuracy of that model. If there are a very large amount of images the accuracyof the trained model will increase.

The problem here is the process of collecting the required datasets. Collecting dataset is a tedious task and often results in unfavorable outcomes, such as incorrect format of gestures, the size of the image which might affect the response time of the model if it is large, the resolution of the images which has to follow a certain standard, otherwise the model will take a longer time to analyse the image to extract the features.

1.5 ORGANIZATION OF THE REPORT

This report is organized as follows: Chapter 2 consists of the related work to development of the android mobile app using AR and ML. Chapter 3 consists of the system analysis which defines about the problem definition, various software components and use cases. Chapter 4 describes about the system design which consists of the system architecture for the proposed system and module description for each module. Chapter 5 tells about the system implementation and the various algorithms. Chapter 6 tells about thevarious results and discussion obtained from the various outputs. Chapter 7 consists of conclusion and the future work for the project. The appendix 1 consists of the source code and software implementations. The appendix 2 consists of the screenshots of the various output screens.

CHAPTER 2

RELATED WORK

2.1 LITERATURE REVIEW

The literature review highlights the need for accurate and efficient ASL recognition systems to address the communication barriers faced by hearing-impaired individuals. Traditional methods have limitations in terms of accuracy and efficiency, emphasizing the need for innovative solutions. The proposed Deep ASLR system, based on CNN, has the potential to significantly improve ASL recognition performance and address these limitations.

The literature review also summarizes the current state-of-the-art in sign language recognition and translation, highlighting challenges such as lack of standardization and regional variations. The effectiveness of CNNs in recognizing sign language gestures is discussed, and their potential for bidirectional translation systems is explored. The system described in the review utilizes computer vision and deep learning to recognize ASL gestures and convert them into text or speech, with hand detection, hand tracking, and sign classification as its three components. By addressing the challenges in sign language recognition and translation, this system has the potential to promote inclusivity and accessibility for individuals who use sign language as their primary means of communication.

2.2 ASLR system:

Ahmed kasapbasi[1], have published a paper on – Deep ASLR for hearing-impaired individuals, suggested that the growing need for accurate ASL recognition systems for hearing-impaired individuals due to the limitations in accuracy and efficiency of traditional methods. The proposed Deep ASLR system, based on CNN, has the potential and significantly improve ASL recognition performance.

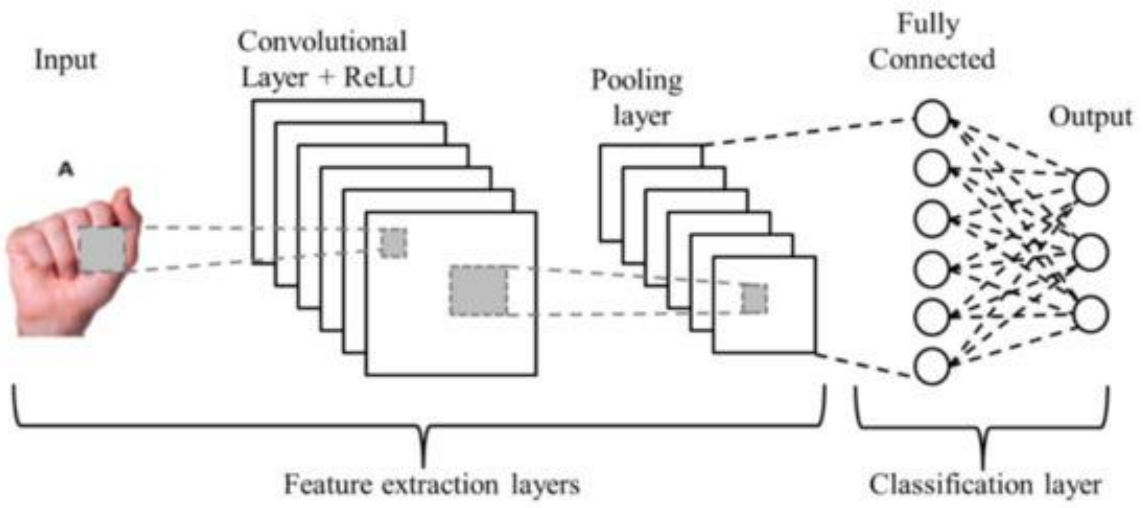


Figure 2.1: The minimal architecture of a CNN.

The above Figure 2.1 explains the minimal architecture of a CNN which consists of five layers: the input, convolution, non-linearity (ReLu), pooling, and classification layer. NNs are widely implemented in applications that need the use of artificial vision techniques. Although the results that have been obtained are very promising, the reality is that they incur high computational costs; therefore it is essential to implement techniques that allow your performance to be increased. For this reason, an optimization of the CNN parameters is presented to improve the recognition percentage and reduce computational cost.In **Figure 2**.2, we can appreciate some parameters that can be optimized in each CNN layer.

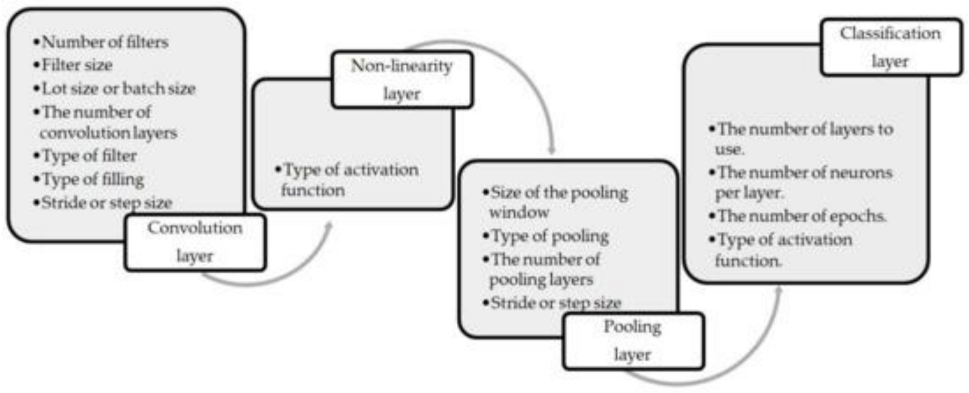


Figure 2.2: Layers and the parameters per layer of a CNN

2.3 WHAT IS CNN?

A CNN is one of the most commonly used deep learning methods to analyze visual imagery. CNN involves less pre-processing compared to other image classification algorithms. The network learns the filters that are normally hand-engineered in other systems. The use of a CNN reduces the images into a format that is easier to process while preserving features that are essential for making accurate predictions. There are four types of operations in a CNN: convolution, pooling, flattening, and fully connected layers.

The convolution layer usually captures low-level features such as colour, edges, and gradient orientation. The pooling layer decreases the spatial dimension of the convolved feature. This operation reduces the required computational time for dealing with the data through dimensionality alleviation. Furthermore, it has the advantage of maintaining dominant features that are positionally and rotationally invariant during the model training process. After the input image has been processed the higher-level features may be used for classification. Therefore, the image is flattened into a 1-D vector. In CNN, the flattened output is supplied to a fully connected layer. After training, using SoftMax classification, the model can provide probabilities of prediction of objects in the image [21] Backpropagation is used to train the network. In this study, the system is implemented by using Keras, Tensorflow and OpenCV libraries.

Lance Fernandes [3] have conducted a research and discussed the effectiveness of CNNs in recognizing sign language gestures and their potential for bidirectional translation systems. This paper he wrote focuses on the sign language alphabet recognition system, which is a starting point for developing more complex systems. There are two types of sign language recognition methods: sensor-based and image-based. Sensor-based methods use localized sensors or gloves to provide accurate information about signs or gestures, while image-based methods use different types of cameras and pattern recognition. Sign language differs among countries and contains non-manual signs such as body gestures and facial expressions. Deep learning has been used to improve the accuracy of the system, and various datasets are created due to regional differences, type of images, and other factors.

Researchers have previously focused on CNNs for sign language recognition systems due to their high performance in image classification. This study has created an ASLA dataset and developed a deep learning-based method for its recognition to overcome a common challenge faced by researchers. The dataset will improve the use of methods in the fields of machine learning and deep learning.

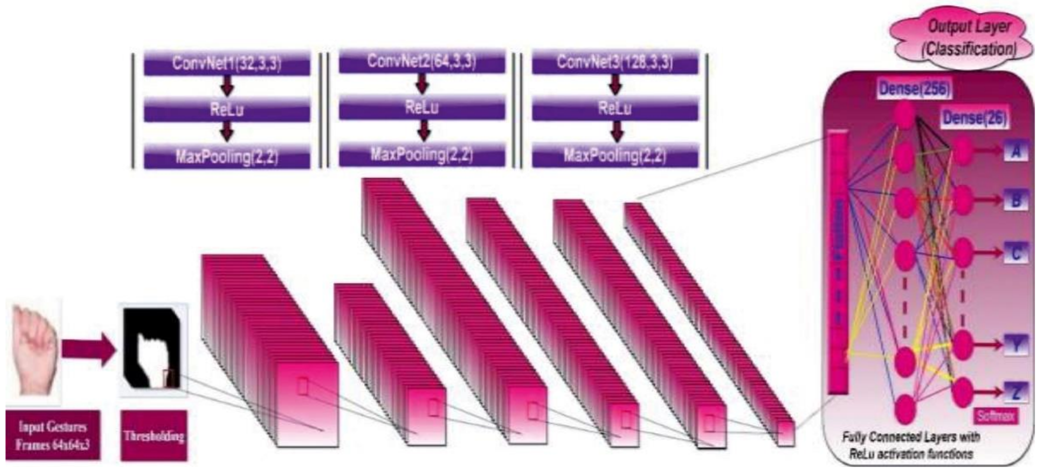


Figure 2.3: The Proposed CNN model.

The CNN model designed in our study consists of multiple layers. Figure. 2.2 illustrates the proposed structure of the CNN which consists of an input layer to input the images with 64 × 64 × 3 dimensions; this represents the size of the sign language frames that are taken as input into the system. The feature extraction part comprises three convolutional layers (Conv1, Conv2, Conv3). The convolution filter dimensions in each layer are 3 × 3. The number of filters is 32 filters for ConvNet1, 64 filters for ConvNet2 and 128 filters for CovNet3. Each convolution operation is followed by rectified linear units (ReLu). After ReLu, MaxPooling is applied with 2 × 2 dimensions. Pooling aims to prevent the loss of important information when the feature is represented. After the convolutional stage, flattening is applied for the classification stage. The classification stage is implemented with fully connected layers followed by a ReLu activation function and one SoftMax output layer [22]

2.3 IMAGE CLASSIFICATION USING DEEP LEARNING

Language is essential to human interaction and has been around since human civilization began. Unfortunately, people with hearing impairment are often forgotten and left out. To prevent this, a sign language recognition system is being proposed. Deep learning must be well versed with the gestures to get a decent accuracy. American Sign Language is used to create the datasets.

The American Sign Language (ASL) alphabets have two methods for identifying hand gestures: glove-based and vision-based. Gloves are uncomfortable and cannot be used in rainy weather, so this research proposes static recognition of hand gestures to improve accuracy. Faster R-CNN with an embedded RPN module is used to locate the hand regions in the video frame, increasing detection accuracy from 89.0% to 91.7% as compared to Fast-RCNN.

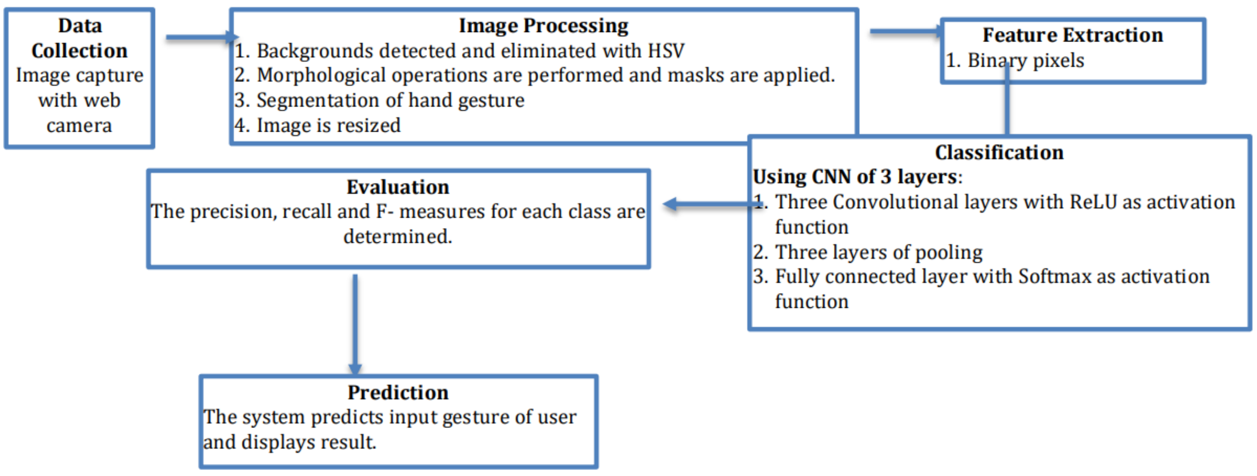

Data Collection
Image capture with web camera
Image Processing
1. Backgrounds detected and eliminated with HSV
2. Morphological operations are performed and masks are applied.
3. Segmentation of hand gesture
4. Image is resized
Feature Extraction
1. Binary pixels
Classification
Using CNN of 3 layers:
1. Three Convolutional layers with ReLU as activation function
2. Three layers of pooling
3. Fully connected layer with Softmax as activation function
Evaluation
The precision, recall and F- measures for each class are determined.
Prediction
The system predicts input gesture of user and displays result.

Figure 2.4 Control flow

The first step of the proposed system is to collect data. Many researchers have used sensors or cameras to capture the hand movements. For our system, we make use of the web camera to shoot the hand gestures. The images undergo a series of processing operations whereby the backgrounds are detected and eliminated using the color extraction algorithm HSV(Hue,Saturation,Value). Segmentation is then performed to detect the region of the skin tone. Using the morphological operations, a mask is applied on the images and a series of dilation and erosion using elliptical kernel are executed. With OpenCV, the images obtained are amended to the same size so there is no difference between images of different gestures . Our dataset has 2000 American sign gesture images out of which 1600 images are for training and the rest 400 are for testing purposes. It is in the ratio 80:20.

Binary pixels are extracted from each frame, and Convolutional Neural Network is applied for training and classification. The model is then evaluated and the system would then be able to predict the alphabets.

2.4 INFERENCE OF LITERATURE SURVEY

The literature review of the papers that were referenced and those that serve as base paper and supporting paper helped to narrow down the main challenges that the proposed application might face. Also it has served as a guide for the development of the application in the right way, using the necessary tools. From the Table 2.2, the following were inferred fromthe literature survey.

Table 2.1: Inference of the Literature Survey

AUTHOR NAME	INFERENCE
Advaith Sridhar, Rohith Gandhi Ganesan [1] 2020	This work introduces the Indian Lexicon Sign Language Dataset (INCLUDE), the first large-scale dataset for Indian Sign Language (ISL), and evaluates deep neural networks for Sign Language Recognition (SLR) on ISL, achieving high accuracies and exploring generalization to American Sign Language (ASL).
Ahmed KASAPBAŞI, Arif YILMA[2] 2022	The study aimed to develop a dataset and CNN model for accurate interpretation of sign language gestures, achieving high performance and showing potential for medical applications, while the dataset's diversity enhances sign language recognition.
Akash Junnarkar, Lance Fernandes [3] 2020	A high-accuracy bidirectional system was developed using a convolutional neural network and diverse dataset to bridge the communication gap between speech and hearing-impaired individuals.
B Sundar, T Bagyammal [5] 2023	A vision-based approach using hand gestures and Mediapipe for hand landmarks achieved 99% accuracy in recognizing ASL alphabets, showcasing potential for text conversion.

Karan Bhavsar1 , Raj Ghatiya [5] 2021	The proposed project aims to develop a cost-effective system using Smart Gloves to provide voice and text output for deaf individuals, enabling them to communicate effectively without the need for sensors.
Mohamed Mahyoub, Friska Natalia [6] 2021	This research investigates the suitability of deep learning models for sign language recognition and classification in different sign languages, finding that the Inflated 3D model performs the best and American Sign Language poses greater challenges for recognition.
O. M. Sincan and H. Y. Keles [8] 2020	This study presents a large-scale multi-modal Turkish Sign Language dataset (AUTSL) with benchmark evaluations and baseline models, showcasing the challenges of sign language recognition and achieving competitive results with deep learning models on the dataset.
Romala Sri Lakshmi Murali,L.D.Ramayya [3] 2022	Sign language recognition using computer vision and CNN techniques offers potential in addressing communication challenges for individuals with hearing disabilities, achieving high accuracy in recognizing ASL gestures and narrowing the communication gap.

CHAPTER 3

SYSTEM ANALYSIS

3.1 PROBLEM DEFINITION

A hand gesture to text converter app is a technological solution that aims to improve the way people interact with their devices. Instead of relying on a physical keyboard or touch screen, users can input text using hand gestures, which can be more intuitive and efficient in certain situations. The app could be used by people with disabilities that affect their ability to use a physical keyboard or touch screen. This includes people with motor impairments, visual impairments, or conditions like arthritis, carpal tunnel syndrome, or repetitive strain injury. By using hand gestures to input text, these users could enjoy a more comfortable and efficient way of communicating with their devices. The app needs to accurately recognize and interpret a wide range of hand gestures, regardless of the user's hand size, shape, and movement. It should also be able to handle variations in lighting and backgrounds, as these factors can affect the quality of the image data. To achieve this, the app should use machine learning algorithms that can process large amounts of data and classify hand gestures based on their intended meaning. Another important factor is user-friendliness. The app should be easy to use and navigate, with clear instructions and feedback to guide users through the process. It should also provide a comfortable and natural user experience, without requiring too much effort or attention from the user. In addition, the app should be compatible with a range of devices and operating systems, to ensure that it can be widely adopted and used by a diverse range of users. Overall, a hand gesture to text converter app has the potential to transform the way people interact with their devices, and to provide a more seamless and intuitive user experience. To achieve this, it requires a combination of sophisticated technology, user-centered design, and a deep understanding of user needs and behaviors.

3.2 PROPOSED SOLUTION

The proposed system aims at developing a Web application that incorporates Machine Learning. The proposed system can record a video using web cam, which will be analyzed by the Machine Learning, it will be analyzed by a open CV model to recognize what the gesture is about. This information is sent to the pre-build module, which will use that classified image’s label to trigger the relevant video to be converted into its respective gesture language.

3.3 SOFTWARE REQUIREMENTS

Table 3.1: Software Requirements

PROGRAMMING LANGUAGE	Python
SCRIPTING LANGUAGE	JavaScript
IDE	Visual studio
STORAGE	Firebase 19.0

3.3.1 Python

Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python's design philosophy emphasizes code readability with its notable use of significant whitespace. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects. Python is dynamically typed and garbage- collected. It supports multiple programming paradigms, including structured (particularly, procedural), object-oriented, and functional programming. Python is often described as a "batteries included" language due to its comprehensive standard library.

Python provides lots of features that are listed below:

Easy to Learn and Use - Python is easy to learn and use. It is developer-friendly and high level programming language

Interpreted Language - Python is an interpreted language i.e. interpreter executes the code line by line at a time. This makes debugging easy and thus suitable for beginners.

Cross-platform Language - Python can run equally on different platforms such as Windows, Linux, Unix and Macintosh etc. So, we can say that Python is a portable language.

Object-Oriented Language - Python supports object oriented language and concepts of classes and objects come into existence.

Extensible - It implies that other languages such as C/C++ can be used to compile the code and thus it can be used further in our python code.

Large Standard Library - Python has a large and broad library and prvides rich set of module and functions for rapid application development.

3.3.2 JavaScript

JavaScript, often abbreviated as JS, is a high-level, interpreted programming language that is commonly used for developing web applications. It is a client-side scripting language, meaning that it runs in the web browser on the user's device rather than on the server. JavaScript was first introduced in 1995 by Brendan Eich, a software engineer at Netscape Communications. Since then, it has become one of the most widely used programming languages in the world, with applications ranging from web development to server-side programming to desktop and mobile applications.

Some of the key features of JavaScript include:

Interactivity: JavaScript allows web developers to create dynamic, interactive web pages that respond to user input. This includes features like pop-ups, drop-down menus, and animations.

DOM manipulation: JavaScript can be used to manipulate the Document Object Model (DOM) of a web page, allowing developers to dynamically update the content and appearance of the page in response to user interactions.

Asynchronous programming: JavaScript supports asynchronous programming, which allows developers to execute code in a non-blocking manner. This is particularly useful for web applications that need to make requests to servers and wait for responses.

Functionality through libraries and frameworks: JavaScript has a rich ecosystem of libraries and frameworks, such as React, Angular, and Vue, that can be used to add complex functionality to web applications. These libraries and frameworks provide pre-built components and tools that developers can use to quickly create complex applications.

Cross-platform compatibility: JavaScript is supported by all major web browsers and can be run on many different operating systems. This makes it a popular choice for web development, as it allows developers to create applications that work across multiple devices and platforms.

3.3.3 Visual Studio

Visual Studio is a powerful integrated development environment (IDE) created by Microsoft. It is designed to help developers build software applications for a variety of platforms, including Windows, macOS, Linux, iOS, Android, and more. Visual Studio includes a wide range of tools and features that can help developers throughout the entire software development lifecycle, from writing and testing code, to debugging and deployment.

The features of Android Studio are:

- Instant App Run
- Extensions
- Version control
- Intellisense
- Task automation

3.3.4 Firebase

Firebase is a mobile and web application development platform developed by Firebase, Inc. in 2011, then acquired by Google in 2014. As of March 2020, the Firebase platform has 19 products, which are used by more than 1.5 million apps. In October 2017, Firebase has launched Cloud Firestore, a real-time document database as the successor product to the original Firebase Realtime Database.
The features of Firebase are:

- Firebase Realtime Database
- Cloud Firestore
- Firebase Storage
- ML Kit

CHAPTER 4

SYSTEM DESIGN

4.1 SYSTEM ARCHITECTURE

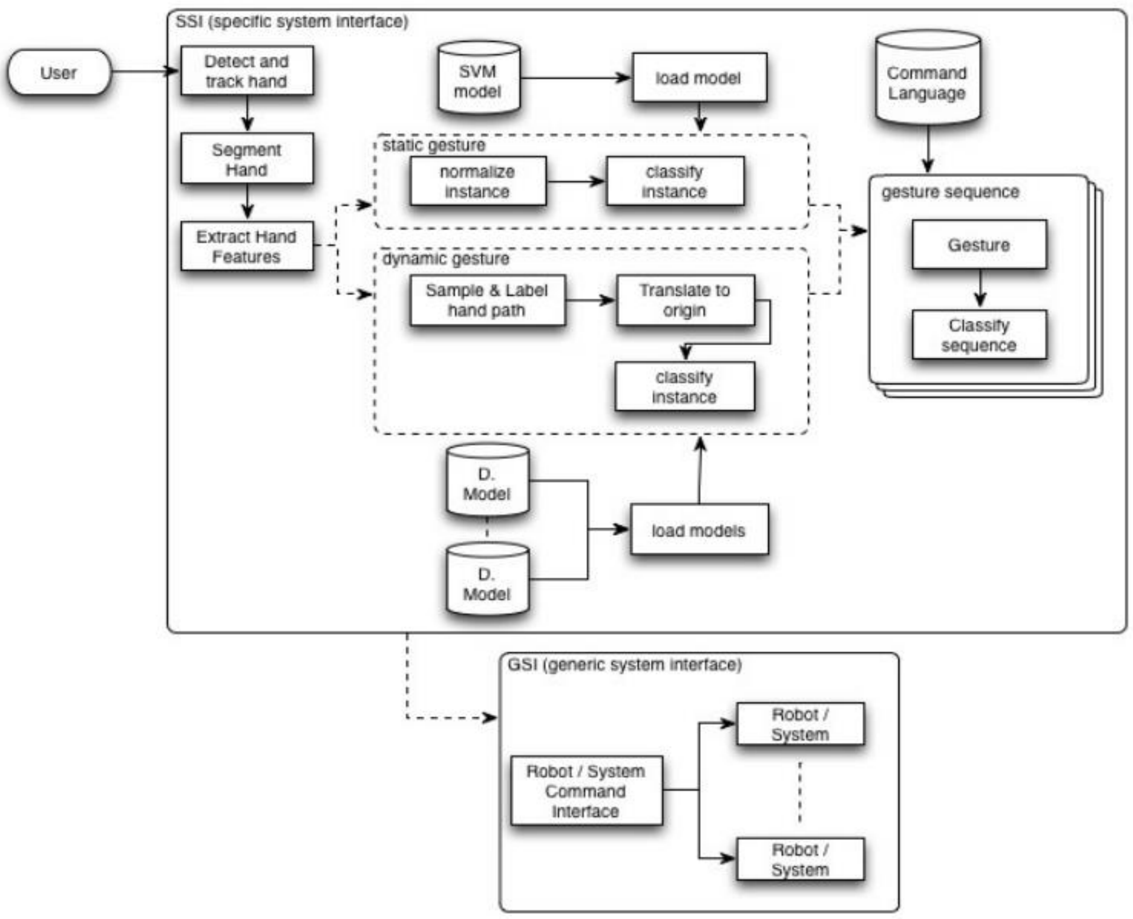


Figure 4.1: system architecture

In this architecture, the system takes video input as an input. The video input is then processed by the Mediapipe framework to extract hand gesture features. These features are then passed on to the SVM classifier which classifies them into different hand gestures. Finally, the gesture output is generated by the system.

Mediapipe is a cross-platform, customizable framework for building real-time, perceptual computing applications. It provides a set of building blocks that allows developers to build complex pipelines for processing and analyzing multimedia data such as video and audio.

SVM stands for Support Vector Machine, which is a machine learning algorithm that can be used for classification or regression problems. In this system, the SVM classifier is used to classify hand gestures based on the features extracted by the Mediapipe framework.

The Learning module is responsible for training the SVM classifier using a dataset of hand gestures. The dataset is first preprocessed to enhance its quality and remove noise. Then, the hand gestures in the dataset are labeled and segmented using frame-by-frame analysis. Relevant features are extracted from the hand gesture data and used to train the SVM classifier. Once the classifier is trained, it can be used to recognize hand gestures in real-time.

The Detection module is responsible for detecting and tracking hands in the video stream using the Mediapipe framework. The module uses the hand detection model in the Mediapipe library to detect the presence of hands in the video. Once hands are detected, the hand landmark model in the library is used to track the position and movement of the hand landmarks over time. The hand landmark data is used to translate the hand gestures into feature values, which are fed into the Extraction module.

The Recognition module in the Hand Gesture Recognition system using Mediapipe and SVM classifier is responsible for recognizing hand gestures in real-time. The module takes the output from the Extraction module, which includes the extracted features of the hand gesture, and compares them against the trained SVM classifier.

Overall, the system architecture of the Hand Gesture Recognition system using Mediapipe and SVM classifier is designed to detect, extract, and recognize hand gestures in real-time. The Learning module trains the SVM classifier using a dataset of hand gestures, while the Detection module uses the Mediapipe framework to detect and track hand landmarks. Finally, the Extraction module extracts relevant features from the hand gesture data and matches them against the trained SVM classifier to recognize the hand gesture.

4.2 MODULES

The proposed system consists of the following main modules,

- Video Preprocessing Module
- Hand Detection Module
- Hand Landmark Estimation Module
- Feature Extraction Module
- SVM Classifier Module
- Gesture Output Module

4.2.1 Video Preprocessing Module

The Video Preprocessing module is an important first step in the Hand Gesture Recognition system using Media pipe and SVM classifier. Its main task is to improve the quality of the video input and remove noise so that subsequent modules can process the video more effectively. This module typically includes tasks such as noise reduction, color correction, resizing, and cropping. By applying these techniques, the Video Preprocessing module can enhance the quality of the video input and ensure that the subsequent modules can operate more accurately and efficiently. The specific techniques and parameters used in this module can vary depending on the characteristics of the video input and the requirements of the subsequent modules. Overall, the Video Preprocessing module plays a crucial role in ensuring the accuracy and robustness of the hand gesture recognition system.

4.2.2 Hand Detection Module

The Hand Detection module is one of the main modules in the Hand Gesture Recognition system using Media pipe and SVM classifier. Its primary function is to detect the hand region in the video input. The Hand Detection module uses computer vision techniques to analyze the video frames and determine the location of the hand. This process involves analyzing the color, shape, and movement of the hand region to identify the pixels that belong to the hand.

There are various hand detection techniques that can be used in this module, such as skin color detection, template matching, and machine learning-based methods. Skin color detection is a popular method for detecting hands in images and videos, where the skin color is used as a feature to segment the hand region. Template matching involves comparing the input image with a pre-defined template of the hand to detect its location. Machine learning-based methods use algorithms such as Convolutional Neural Networks (CNNs) and Haar cascades to detect hands in images and videos.

Once the hand region is detected, the Hand Detection module provides the region of interest (ROI) to the next module, which is the Hand Landmark Estimation module. The ROI contains the pixels that belong to the hand, which can be used to estimate the landmarks or key points of the hand in the subsequent module.

4.2.3 Hand Landmark Estimation Module

The Hand Landmark Estimation module is a critical component of the Hand Gesture Recognition system using Media pipe and SVM classifier. Its primary function is to estimate the landmarks or key points of the hand region, which are necessary for recognizing the hand gestures. The Hand Landmark Estimation module uses computer vision techniques to identify and track the key points of the hand, such as the fingertips, knuckles, and wrist. This process involves analyzing the hand region in each frame of the video input and estimating the position of the key points using algorithms such as Regression and Classification. Once the keypoints are estimated, they are used as input features for the next module, which is the Gesture Recognition module. The Hand Landmark Estimation module plays a crucial role in ensuring the accuracy and robustness of the Hand Gesture Recognition system by providing accurate and reliable estimates of the hand landmarks.

4.2.4 Feature Extraction Module

The Feature Extraction module is an important component of the Hand Gesture Recognition system using Media pipe and SVM classifier. Its main task is to extract relevant features from the hand landmarks estimated by the previous module. The Feature Extraction module typically includes tasks such as calculating the distance between keypoints, calculating the angles between key points, and computing statistical measures such as mean, variance, and standard deviation. These features provide a concise representation of the hand gestures, which can be used as input features for the next module, which is the Gesture Recognition module. The specific features and their values extracted by this module can vary depending on the characteristics of the hand gestures being recognized and the requirements of the system. Overall, the Feature Extraction module plays a crucial role in converting the hand landmark estimates into a concise and informative representation of the hand gestures, which is essential for accurate and robust gesture recognition.

4.2.5 SVM Classifier Module

The SVM (Support Vector Machine) Classifier module is the final component of the Hand Gesture Recognition system using Mediapipe and SVM classifier. Its primary function is to classify the hand gestures based on the features extracted by the previous module. The SVM classifier is a machine learning algorithm that works by finding the optimal hyperplane that separates the different classes of hand gestures in the feature space. This hyperplane is determined by maximizing the margin between the support vectors, which are the data points closest to the decision boundary. Once the hyperplane is determined, the SVM classifier can classify new hand gestures based on their feature values relative to the decision boundary. The SVM Classifier module is typically trained on a dataset of hand gestures with known labels and their corresponding feature values. This training process involves selecting the optimal parameters for the SVM classifier, such as the kernel type and regularization parameter, to achieve the best performance on the training dataset.

4.2.6 Gesture Output Module

The Gesture Output module is the final stage of the Hand Gesture Recognition system using Mediapipe and SVM classifier. Its main task is to provide an output representation of the recognized hand gestures, which can be used for various applications. The Gesture Output module typically includes tasks such as mapping the classified gesture label to a meaningful output, such as a text or voice command, or controlling a device, such as a robot or a computer. The specific output representation can vary depending on the application and the requirements of the system. For instance, in a sign language recognition system, the Gesture Output module may convert the recognized gestures into corresponding sign language sentences or words. Overall, the Gesture Output module plays a crucial role in enabling the Hand Gesture Recognition system to interact with the external environment and provide useful output based on the recognized hand gestures.

4.3 UML DIAGRAMS

Unified Modelling Language, is a general purpose development and modelling language in the field of software engineering that is intended provide a standard way to visualize the design of a system.

The UML diagrams that are discussed below are,

- Use Case diagram
- Class diagram
- Sequence diagram
- Activity diagram

4.3.1 Use Case Diagram

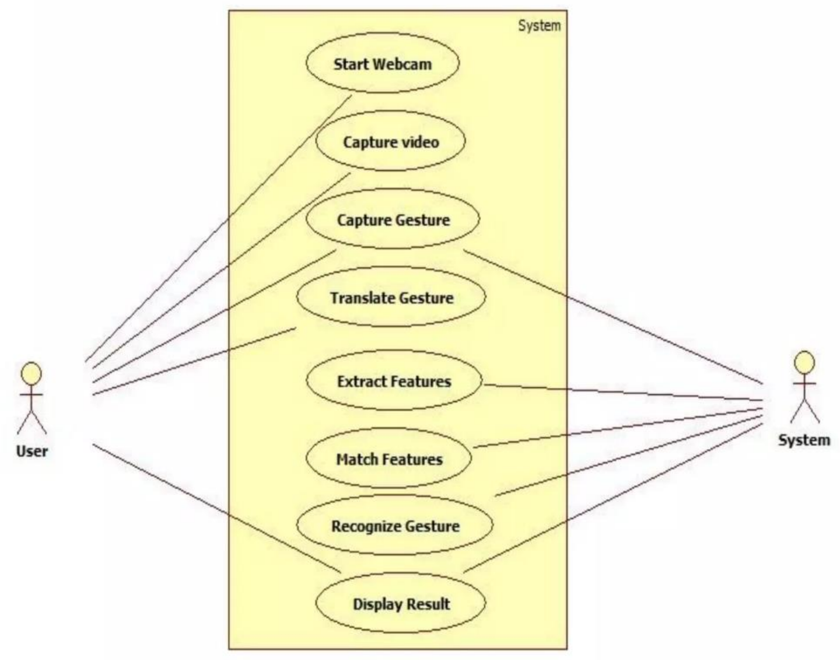


Figure 4.2: Use Case Diagram

The figure 4.2 is the use case diagram for the Hand Gesture Recognition system using Mediapipe and SVM classifier consists of several use cases that are performed by two actors, the User, and the System. The User provides input through hand gestures, which are captured by the System using the webcam. The System then performs several use cases to process the captured video and recognize the hand gesture. These use cases include capturing video, capturing gesture, translating gesture, extracting features, matching features, recognizing gesture, and displaying result. The System matches the extracted features against a trained model using the SVM classifier to recognize the hand gesture. Once recognized, the System displays the result to the User. The use case diagram provides a visual representation of the interactions between the User and the System, and the steps involved in the gesture recognition process. It helps to identify the key features and requirements of the system and serves as a useful tool for communication between stakeholders.

4.3.2 Class Diagram

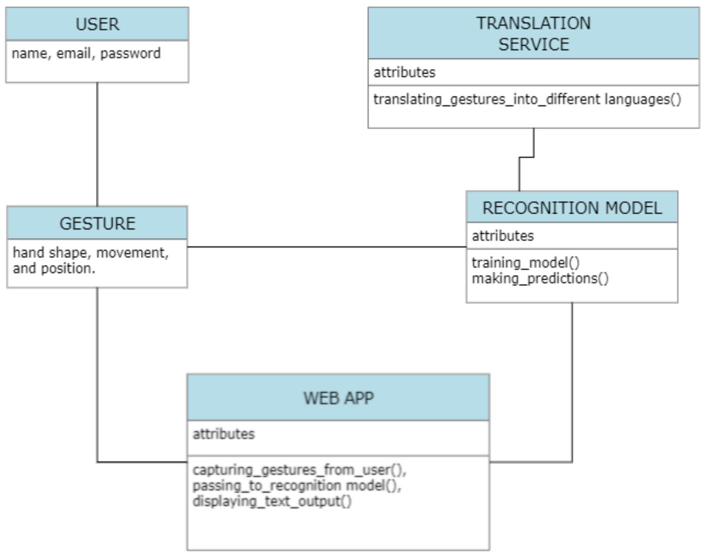


Figure 4.3: Class Diagram

The class diagram for the Hand Gesture Recognition system using Mediapipe and SVM classifier includes classes representing the major components of the system, such as the Video Preprocessing module, Hand Detection module, Feature Extraction module, SVM Classifier module, and Gesture Output module. The diagram also includes other relevant classes, such as data structures and utility functions.

The relationships between classes in the diagram illustrate how the various modules interact with each other. For example, the Video Preprocessing module has a composition relationship with the Hand Detection module, while the Feature Extraction module has an association relationship with the Hand Detection module.

The SVM Classifier module has an inheritance relationship with a general Classifier class, as it implements specific methods for training and recognizing hand gestures using an SVM algorithm. The Gesture Output module has a composition relationship with the SVM Classifier module, as it uses the output of the classifier to recognize and display the recognized hand gestures.

4.3.3 Sequence Diagram

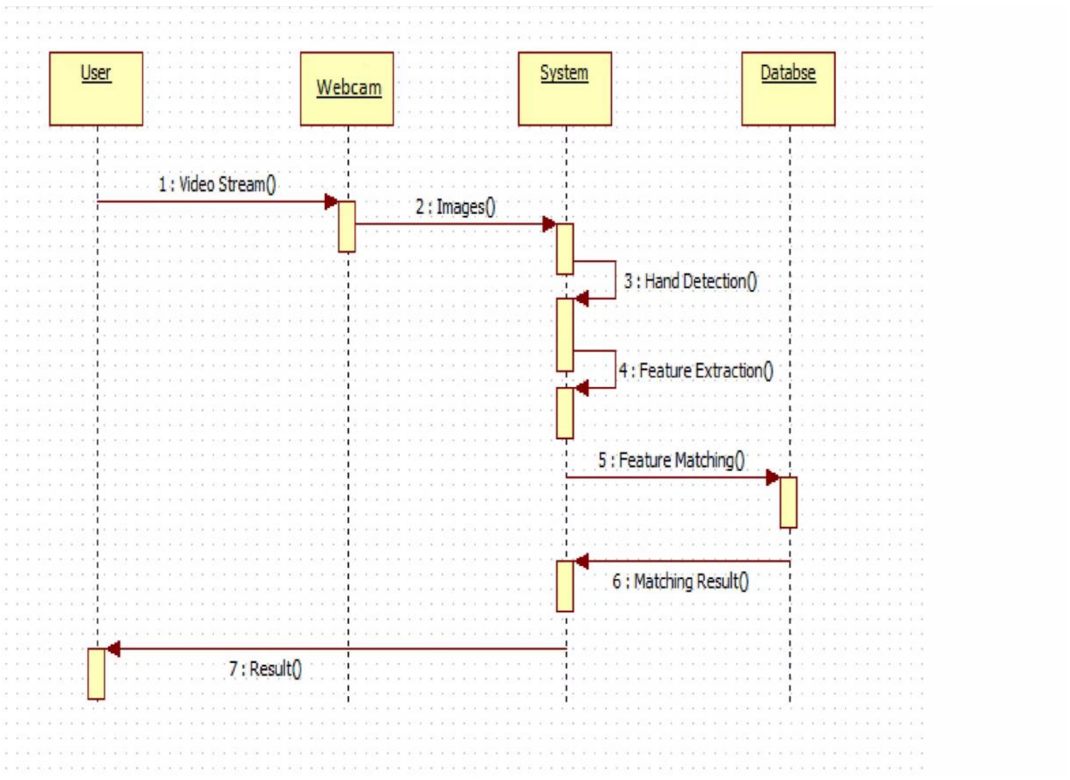


Figure 4.4: Sequence Diagram

The Figure 4.4 is the sequence diagram for the Hand Gesture Recognition system using Mediapipe and SVM classifier shows the interactions between the User and the System during the gesture recognition process. The diagram illustrates the sequence of messages and events that occur between the User and the System, starting with the User initiating the process by starting the webcam. The System then captures video input from the webcam and processes it for gesture recognition. The User then performs a hand gesture, which is captured by the System and translated into feature values. The System extracts relevant features from the feature values and matches them against a trained model using the SVM classifier to recognize the hand gesture. Once the gesture is recognized, the System displays the result to the User. The sequence diagram helps to illustrate the flow of control and data between the User and the System, making it easier to understand the overall process of gesture recognition.

4.3.4 Activity Diagram

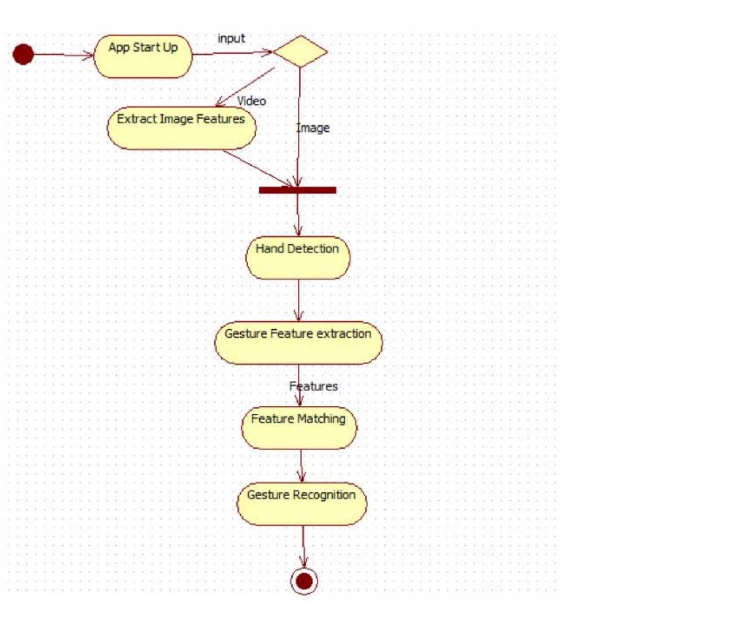


Figure 4.5: Activity Diagram

The Figure 4.5 is the activity diagram for the Hand Gesture Recognition system using Mediapipe and SVM classifier shows the activities involved in the gesture recognition process. The diagram starts with the User initiating the process by starting the webcam. The System then captures video input from the webcam and processes it for gesture recognition. The System then performs a loop of capturing hand gesture, translating it into feature values, extracting relevant features from the feature values, and matching the features against a trained model using the SVM classifier. If the gesture is recognized, the System displays the result to the User. The loop continues until the User stops the webcam. The activity diagram helps to illustrate the sequence of activities involved in the gesture recognition process and the conditions that determine the flow of control. It provides a high-level view of the process, making it easier to understand the overall workflow.

CHAPTER 5

SYSTEM IMPLEMENTATION

5.1 BASE ALGORITHM

i. Import the necessary libraries and dependencies, such as OpenCV, Mediapipe, and scikit-learn.
ii. Load the pre-trained SVM classifier, or train the SVM classifier on a labeled dataset of hand gesture images.
iii. Initialize the camera and set up the video stream.
iv. Read frames from the video stream.
v. Apply hand detection using Mediapipe to identify the location of the hand in the frame.
vi. Extract hand landmarks from the detected hand using Mediapipe.
vii. Extract features from the hand landmarks, such as the distance between specific l landmarks or the angle between specific landmarks.
viii. Use the SVM classifier to predict the hand gesture based on the extracted features.
ix. Display the predicted hand gesture on the video stream.
x. Display the translated Text.
xi. Repeat steps 4-9 until the video stream is stopped.

5.1.1 Explanation of the Algorithm

Firstly, the necessary libraries and dependencies are imported into the program. OpenCV is used to handle the video stream and perform image processing operations, while Mediapipe is used to detect the hand and extract landmarks from it. Scikit-learn is used to implement the SVM classifier.

Next, the SVM classifier is loaded into the program or trained on a labeled dataset of hand gesture images. This dataset should include a variety of hand gestures to ensure accuracy in real-world scenarios.

After the SVM classifier is loaded, the camera is initialized and the video stream is set up to capture frames from the camera. Frames are then read from the video stream and stored in memory for processing.

Hand detection is performed using Mediapipe to identify the location of the hand in the frame. This involves identifying the coordinates of specific landmarks on the hand, such as the fingertips, wrist, and palm. Once the hand is detected, Mediapipe is used to extract landmarks from the hand. These landmarks are used to calculate features for the SVM classifier.

Features are extracted from the landmarks, such as the distance between landmarks, the angle between landmarks, and the ratio of distances between landmarks. These features are used to classify the hand gesture using the SVM classifier.

The SVM classifier is trained to recognize a variety of hand gestures, such as pointing, waving, and making a fist. The predicted hand gesture is displayed on the video stream, allowing the user to see how the algorithm is recognizing their gestures in real-time.

Once the hand gesture is classified, the text associated with that gesture is displayed on the video stream. The Google Cloud Translation API is then used to translate the displayed text into the user's chosen language. The translated text is displayed on the video stream for the user to see.

The algorithm is repeated until the video stream is stopped. The accuracy of the translation will depend on the quality of the Google Cloud Translation API and the availability of training data for the languages being translated.

5.1.3 Hand Detection using Mediapipe

Mediapipe uses a deep neural network model to detect the hand in the pre-processed image. A deep neural network is a type of machine learning model that is designed to mimic the structure and function of the human brain. It consists of layers of interconnected nodes that perform complex computations on the input data.

The hand detection model used by Mediapipe has been trained on a large dataset of hand images. During the training process, the model is shown thousands of hand images with different orientations and lighting conditions.

It learns to recognize patterns in the images that are characteristic of a hand and uses these patterns to accurately detect hand in new images.

The model is able to detect the hand in various orientations and lighting conditions because it has learned to recognize the patterns of a hand from many different angles and lighting conditions during the training process. This makes it robust to changes in the image, such as changes in orientation or lighting.

5.1.4 Gesture Recognition

The landmarks extracted from the hand are used to recognize hand gestures using a machine learning algorithm such as an SVM classifier.

Support Vector Machine (SVM) is a popular machine learning algorithm used for classification tasks. In the context of hand gesture recognition, the SVM classifier is trained on a dataset of hand gesture images with their corresponding labels. During the training process, the SVM learns to recognize patterns in the landmark data that are characteristic of each hand gesture.

Once the SVM classifier has been trained, it can be used to classify new hand gestures based on the landmarks extracted from the hand using Mediapipe. The SVM classifier takes the landmarks as input and outputs the predicted hand gesture based on the learned patterns from the training data.

In the context of your algorithm, the SVM classifier is used to recognize specific hand gestures and translate them into text in the user's chosen language. For example, if the user makes a gesture that corresponds to the letter "A", the SVM classifier will recognize this gesture and output the letter "A" in the user's chosen language.

5.2 FUNCTIONAL COMPONENTS OF CODING

The following are the functional components

- TensorFlow
- Keras
- Mediapipe
- Support Vector Machine (SVM) Classifier
- Google Translate API
- Python
- OpenCV

5.2.1 TensorFlow

TensorFlow is an open-source machine learning framework developed by Google. It is designed to make it easy to build and train machine learning models, from simple linear models to complex deep learning architectures. TensorFlow provides a high-level API for building models as well as a lower-level API for more advanced users.

TensorFlow supports a wide range of machine learning tasks, including classification, regression, and clustering. It also provides built-in support for neural networks, including convolutional neural networks (CNNs), recurrent neural networks (RNNs), and transformers. TensorFlow supports both CPU and GPU acceleration, allowing users to train models on a variety of hardware configurations.

Another key feature of TensorFlow is its ability to seamlessly integrate with other machine learning tools and libraries, including Keras, scikit-learn, and Pandas. This makes it easy to build end-to-end machine learning pipelines using a variety of tools and libraries.

5.2.2 Keras

Keras is a high-level neural network API that is built on top of TensorFlow. It provides a user-friendly interface for building and training deep learning models, allowing users to easily design, configure, and train neural networks with just a few lines of code.

Keras also provides a simple and intuitive syntax for defining and training neural networks.
Users can define their neural network architecture using a sequential or functional API, and then compile the model with a chosen optimizer, loss function, and performance metrics. Once the model is compiled, users can train the model on their data and evaluate its performance using the fit and evaluate methods.

In addition to its ease-of-use, Keras is also highly optimized for performance. Under the hood, Keras uses TensorFlow as its backend, which allows it to take advantage of the powerful computational capabilities of modern GPUs. Keras also supports distributed training, which enables users to train large-scale models on multiple GPUs or machines.

5.2.3 Mediapipe

Mediapipe is an open-source framework developed by Google that provides a set of pre-built, customizable building blocks for building machine learning pipelines for real-time media processing applications. It is designed to make it easy to build pipelines that can process media streams, including video and audio, and extract meaningful insights from them.

Mediapipe also provides pre-built building blocks for a wide range of other computer vision and machine learning tasks, including object detection, image segmentation, pose estimation, and action recognition. These building blocks can be easily customized and combined to create custom machine learning pipelines that meet the specific needs of a given application.

Mediapipe is a powerful and flexible framework for building machine learning pipelines for real-time media processing applications. Its pre-built building blocks and performance optimization make it a popular choice for researchers and practitioners in the computer vision and machine learning communities.

5.2.4 Support Vector Machine (SVM) Classifier

Support Vector Machine (SVM) is a type of machine learning algorithm that is used for classification and regression analysis. In classification tasks, SVM aims to divide data points into different classes using a hyperplane. The hyperplane is defined as the line that maximizes the margin between the classes. The data points that are closest to the hyperplane are called support vectors.

SVM works by mapping input data to a high-dimensional feature space, where it can more easily find a hyperplane that separates the data into different classes. The algorithm searches for the hyperplane that maximizes the margin between the support vectors of different classes. The margin is defined as the distance between the hyperplane and the closest data points from each class. By maximizing the margin, SVM can find a hyperplane that is more robust to noise and generalizes better to unseen data.

Overall, SVM is a powerful and widely used machine learning algorithm for classification tasks. Its ability to find an optimal hyperplane and maximize the margin between classes makes it effective at dealing with noisy and complex data. SVM is also widely used in real-world applications, such as text classification, image classification, and bioinformatics.

5.2.5 PYTHON

Python is a high-level programming language that is widely used for general-purpose programming. It was first released in 1991 by Guido van Rossum, and since then it has become one of the most popular programming languages in the world. Python is known for its simplicity, readability, and ease of use, making it a popular choice for beginners and experienced developers alike.

One of the main advantages of Python is its ease of use and readability. Python code is often easier to read and understand than code written in other programming languages. This makes it easier for developers to collaborate on projects and maintain existing codebases. Python also has a large standard library that provides many built-in functions and modules that make it easy to perform common tasks such as file I/O, regular expressions, and network programming.

Overall, Python is a versatile, easy-to-use, and widely used programming language that is popular among developers for a wide range of applications. Its simplicity, readability, and large standard library make it an attractive choice for beginners, while its flexibility and power make it a popular choice for experienced developers.

5.2.6 OpenCV

OpenCV (Open Source Computer Vision Library) is a popular open-source computer vision and machine learning library that is used for real-time image and video processing. It was originally developed by Intel and is now maintained by the OpenCV Foundation. OpenCV provides a large collection of algorithms and tools for image and video processing, including image and video capture, filtering, feature detection, object recognition, and tracking.

One of the key features of OpenCV is its support for multiple programming languages, including C++, Python, Java, and MATLAB. This makes it easy for developers to use OpenCV in their preferred programming language. OpenCV is also cross-platform, which means that it can be used on a wide range of operating systems, including Windows, macOS, Linux, and Android.

OpenCV includes a wide range of computer vision algorithms, including algorithms for image and video processing, object recognition, feature detection, and tracking. Some of the popular algorithms included in OpenCV are Haar cascades for object detection, SIFT and SURF for feature detection, and optical flow for tracking. OpenCV also provides a range of tools for machine learning, including support for popular machine learning libraries such as TensorFlow, Keras, and PyTorch.

CHAPTER 6

SYSTEM TESTING

6.1 TESTING OBJECTIVES

Testing is a set of activities that can be planned in advance and conducted systematically. For this reason, a template for software testing, a set of steps into which we can place specific test case design techniques and testing methods should be defined for software process. Testing often accounts for more effort than any other software engineering activity. If it is conducted haphazardly, time is wasted, unnecessary effort is expanded, and even worse, errors sneak through undetected. It would therefore seem reasonable to establish a systematic strategy for testing software.

6.2 Types of testing

In this methodology we use two types of testing:

- Functional Testing
- Non-Functional Testing

6.2.1 Functional Testing

It is a software testing which validates the software system against the functional requirements. It checks each function of the application by providing the required input and output. This can be executed first, this uses Manual or automation testing tool. These testing uses:

6.2.1.1 Integration Testing

6.2.1.2 Security Testing

6.2.2 Non-Functional Testing

Non-Functional testing is used to check the performance, usability or a reliability of a software. It can perform only after functional testing. This function can check how good a product works. Manual testing cannot be done easily. Non Functional testing can increase the user experience to a high extent which has a impact on the quality of the software. These testing includes:

6.2.2.1 Performance Testing

6.2.2.2 Usability Testing

6.3 COMMON SOFTWARE TESTING TYPES

6.3.1 Integration Testing

Integration testing is a process where each blocks are combined and tested as an individual. It is used to test the data by using a design so that the system can operate successfully. Thus the system testing is a process to check the combinational behaviour and validate whether the requirements are implemented correctly.

6.3.2 Performance Testing

It is used to check the speed, reliability and scalability of a software. It is done to provide stakeholders the information about their application. It can demonstrate how the system meets it pre-defined performance criteria. Costs of performance testing are usually more than made up for with improved customer satisfaction.

6.3.3 Validation Testing

The final step involves Validation testing, which determines whether the software functions as the user expected. The end-user rather than the system developer conducts this test most software developers as a process called “Alpha and Beta test” to uncover that only the end user seems able to find. The compilation of the entire project is based on the full satisfaction of the end users.

6.3.4 Acceptance Testing

An Acceptance Test is performed by the client and verifies whether the end to end the flow of the system is as per the business requirements or not and if it is as per the needs of theend-user. Client accepts the software only when all the features and functionalities work as expected. It is the last phase of the testing, after which the software goes into production. This is also called User Acceptance Testing (UAT).

6.3.5 Backend Testing

Whenever an input or data is entered on front-end application, it stores in the database and the testing of such database is known as Database Testing or Backend Testing. There are different databases like SQL Server, MySQL, and Oracle, etc. Database Testing involves testing of table structure, schema, stored procedure, data structure and so on. In Back-end Testing GUI is not involved, testers are directly connected to the database with proper access and testers can easily verify data by running a few queries on the database.

6.2 TEST CASE STRATEGIES

6.4.1 Black Box Testing

In this testing the blocks will give the details of the pesticides along with its ingredients to the user and test the output. Here the testing does not go for watching the internal file in the system and what are the changes made on them for the required output.

6.4.2 White Box Testing

It is just the vice versa of the Black Box testing. It does not watch the internal variables during testing. This gives clear idea about what is going on during execution of the system. The point at which the bug occurs was all clear and was removed.

6.3 TEST CASES

Table 6.1: Test Case 1

Project Name: Sign2Text App	
Test Case ID: 01	**Test Designed by:**
Test Priority: High	**Test Designed date:**
Modular Name: Output Check	**Test Executed by:**
Test Title: Verify Output Displayed	**Test Executed date:**
Description: Testing the accuracy of sign-to-text conversion and displayed output.	

Table 6.2: Test Case 1 – Result and Status

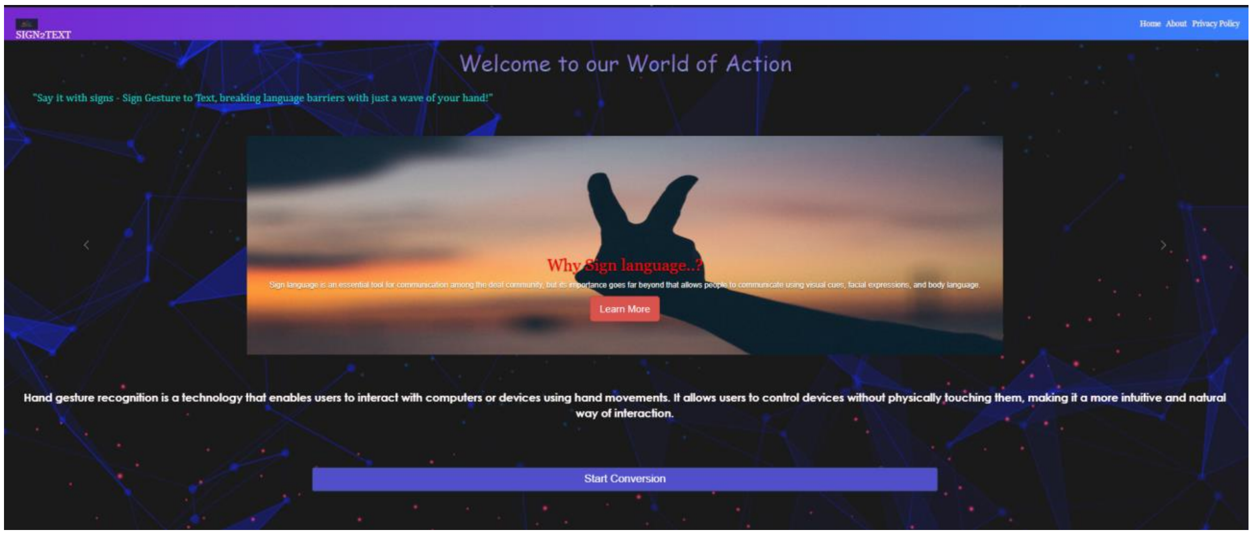
Steps	Test Steps	Test Data	Expected Result	Actual Result	Status (Pass/Fail)
1	Capture a video of hand gestures.	Use the captured video as the test data.	The app should accurately recognize the hand gestures and convert them into corresponding text.	The app accurately recognizes the hand gestures and converts them into corresponding text.	Pass
2	Capture a video of hand gestures.	Use the captured video as the test data.	The app should provide translations of the recognized gestures into English.	The app provides translations of the recognized gestures into English.	Pass
3	Capture a video of hand gestures.	Use the captured video as the test data.	The app should provide translations of the recognized gestures into Tamil	The app provides translations of the recognized gestures into Tamil.	Pass

4	Capture a video of hand gestures.	Use the captured video as the test data.	The app should provide translations of the recognized gestures into Hindi.	The app provides translations of the recognized gestures into Hindi.	Pass

CHAPTER 7

OUTPUT AND EXPLANATION

7.1 SCREENSHOTS



Welcome to our World of Action

"Say it with signs - Sign Gesture to Text, breaking language barriers with just a wave of your hand!"

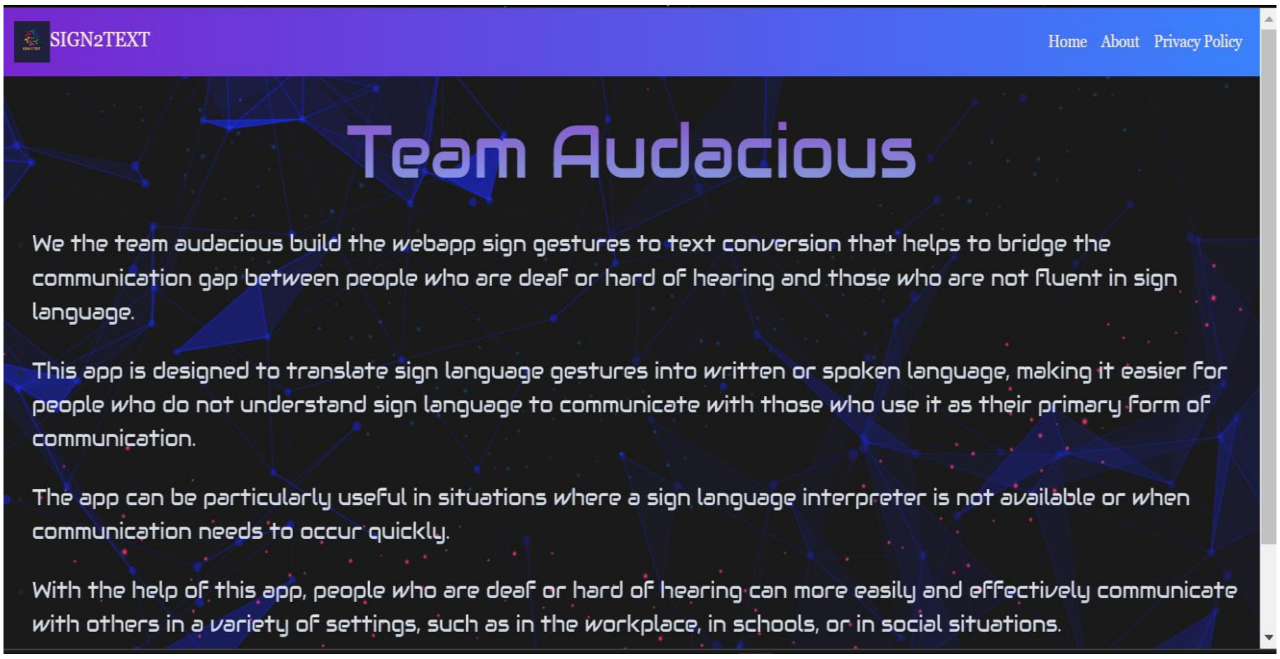
Why Sign language..?

Sign language is an essential tool for communication among the deaf community, but its importance goes far beyond that allows people to communicate using visual cues, facial expressions, and body language.

Hand gesture recognition is a technology that enables users to interact with computers or devices using hand movements. It allows users to control devices without physically touching them, making it a more intuitive and natural way of interaction.

Figure 7.1: Home page of Sign2Text app

The home page of the application showcases its functionality and includes essential information such as an overview of how the app works, details about the team behind it, and a privacy policy to ensure user data protection.



Team Audacious

We the team audacious build the webapp sign gestures to text conversion that helps to bridge the communication gap between people who are deaf or hard of hearing and those who are not fluent in sign language.

This app is designed to translate sign language gestures into written or spoken language, making it easier for people who do not understand sign language to communicate with those who use it as their primary form of communication.

The app can be particularly useful in situations where a sign language interpreter is not available or when communication needs to occur quickly.

With the help of this app, people who are deaf or hard of hearing can more easily and effectively communicate with others in a variety of settings, such as in the workplace, in schools, or in social situations.

Figure 7.2: About us page of Sign2Text app

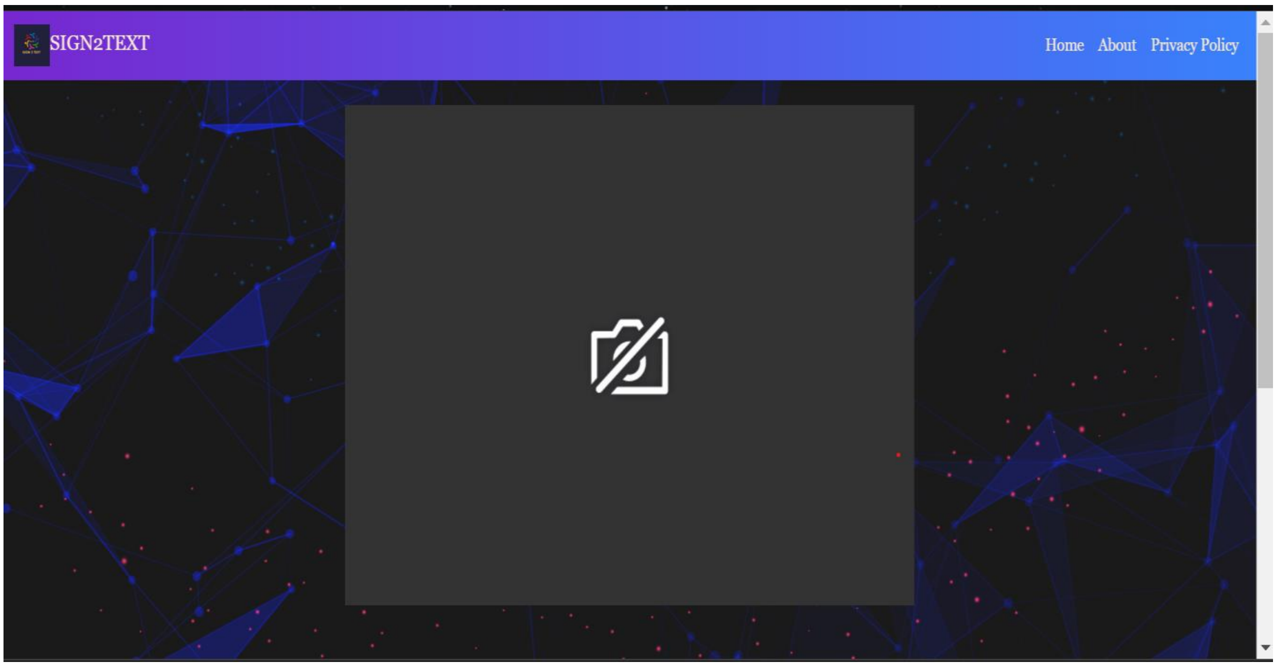


Figure 7.3: The main webpage

Figures 7.2 and 7.3 demonstrate additional output instances of the application. In Figure 7.2, the camera successfully opens and captures real-time images from either a webcam or the camera of a mobile device. Figure 7.4 showcases an example where the application accurately recognizes and interprets the gesture performed by a human user.

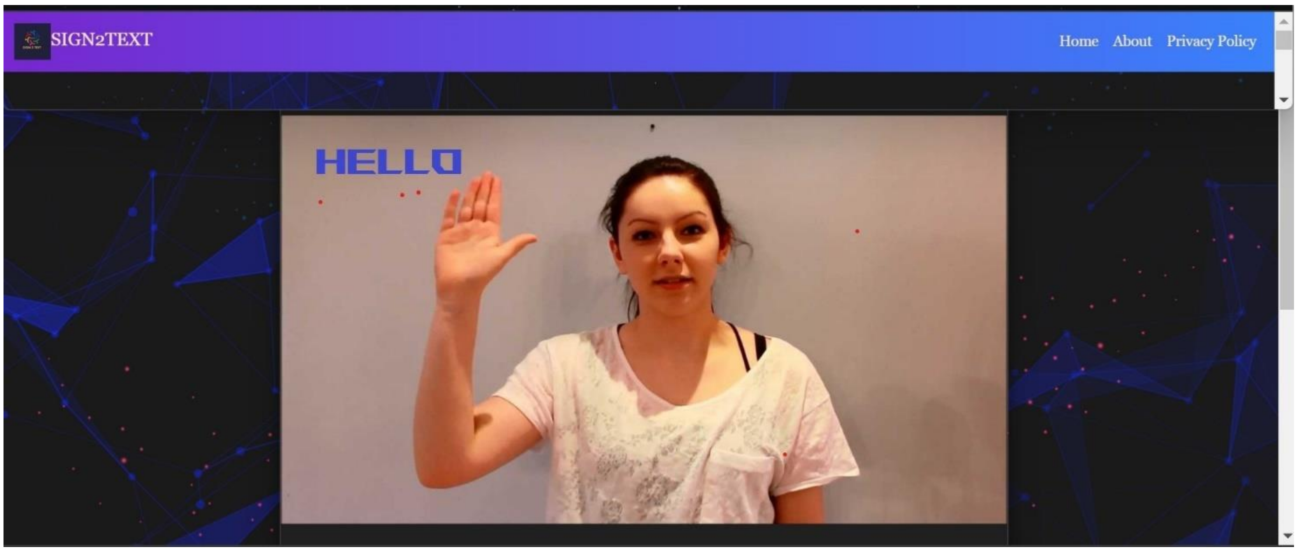


Figure 7.4: Recognizing the Image

CHAPTER 8

CONCLUSION AND FUTURE WORK

8.1 CONCLUSION

In conclusion, a hand gesture to text converter app can be a useful tool for individuals who have difficulty typing or speaking due to physical or speech impairments. The app would use computer vision and machine learning techniques to recognize hand gestures and convert them into text, which could then be input into a computer or mobile device. The development of such an app would require expertise in computer vision, machine learning, and software development. It would also need to be tested and validated to ensure that it is accurate and reliable in recognizing and converting hand gestures into text. Overall, a hand gesture to text converter app has the potential to improve accessibility for individuals with physical or speech impairments, allowing them to communicate more easily and effectively.

8.2 FUTURE WORK

In the future Sign2Text can be further improved to make the application more efficient and achieve the following aspects,

8.2.1 Mobile app development

8.2.2 Much wider field of Gesture recognition

8.2.3 Improved Accuracy

8.2.4 Integration with Virtual Meeting platforms

8.2.1 Mobile app development

In the future, The future scope of developing a mobile app for hand gesture to text converter is promising, as it has the potential to benefit a wide range of users. A hand gesture to text converter app can improve accessibility for individuals with physical or speech impairments, allowing them to communicate more easily and effectively. Using hand gestures to input text can be more convenient and efficient than typing on a keyboard or using a virtual keyboard on a mobile device. A well-designed hand gesture to text converter app can provide a more intuitive and engaging user experience compared to traditional text input methods.

8.2.2 Much Wider Field of Gesture Recognition

Some of the other areas where hand gesture recognition to text app could be developed and applied include:

Education: A hand gesture to text converter app could be used in educational settings to provide a more interactive and engaging learning experience, allowing students to interact with educational content using hand gestures.

Customer service: A hand gesture to text converter app could be used in customer service applications to allow customers to interact with customer service representatives using hand gestures, improving the speed and efficiency of customer service interactions.

Retail: A hand gesture to text converter app could be used in retail applications to allow customers to browse products and make purchases using hand gestures, providing a more intuitive and interactive shopping experience.

Entertainment: A hand gesture to text converter app could be used in entertainment applications, such as music or video streaming services, to allow users to control their media using hand gestures, providing a more immersive and enjoyable entertainment experience.

Industrial automation: A hand gesture to text converter app could be used in industrial automation applications to allow workers to control machinery and equipment using hand gestures, improving safety and efficiency in manufacturing and other industrial settings.

8.2.3 Improved accuracy

Improved accuracy in hand gesture to text converter app can significantly enhance the usability and reliability of the technology. By leveraging machine learning and artificial intelligence algorithms, we can improve the accuracy of hand gesture recognition and translation. This can help to minimize errors and inaccuracies in the recognition and translation process, ensuring that the output is as accurate as possible. Improved accuracy can also help to reduce the cognitive load on the user, as they do not have to constantly correct or adjust the output. Additionally, improved accuracy can also help to expand the range of gestures that can be recognized and translated, making the app more versatile and useful. By continuing to improve the accuracy of hand gesture to text converter app, we can create a more efficient and effective tool for individuals with different abilities and needs.

8.2.4 Integration with Virtual Meeting platforms

The integration of hand gesture to text converter app that helps impaired people with virtual meeting platform can provide a significant benefit to individuals who are deaf or hard of hearing. By recognizing hand gestures and translating them into text captions, the app can provide real-time captioning for virtual meetings, making them more accessible for individuals with hearing impairments. This can greatly enhance the ability of individuals with hearing impairments to participate in virtual meetings, allowing them to fully engage with the content and contribute to the discussion. In addition, the app can also be used to facilitate interaction in virtual meetings through hand gestures, making it easier for individuals with hearing impairments to ask questions, provide feedback, and engage in discussions. By integrating hand gesture to text converter app with virtual meeting platforms, we can help to create a more inclusive and accessible virtual meeting environment for individuals with different abilities and needs.

APPENDIX 1

SAMPLE CODE

Views.py

```
from django.shortcuts import render
from django.http import HttpResponse
from django.http.response import StreamingHttpResponse
from googletrans import Translator
from django.views.decorators import gzip
from django.shortcuts import render
import cv2
import mediapipe as mp
import pandas as pd
import os
import numpy as np
import pickle
import cv2 as cv
import googletrans
from googletrans import Translator

# Create your views here.
def home(request):
    return render(request, 'index.html')
def start_conversion(request):
    return render(request,'conversion.html')
def index(request):
      return render(request, 'index.html')
def about(request):
     return render(request, 'about.html')
def pp(request):
     return render(request, 'privacy_policy.html')
def y_sign_language(request):
     return render(request,'y_sign.html')
```

```
def how_works(request):
     return render(request,'how_works.html')
def camera(request):
    return render(request, 'camera.html')

from django.views.decorators import gzip
from django.http import StreamingHttpResponse
import cv2
import threading

class VideoCamera(object):
    def __init__(self):
        self.video = cv2.VideoCapture(0)
        (self.grabbed, self.frame) = self.video.read()
        threading.Thread(target=self.update, args=()).start()

    def __del__(self):
        self.video.release()

    def get_frame(self):
        image = self.frame
        _, jpeg = cv2.imencode('.jpg', image)
        return jpeg.tobytes()

    def update(self):
        while True:
            (self.grabbed, self.frame) = self.video.read()



def image_processed(hand_img):
    # Image processing
    # 1. Convert BGR to RGB
    img_rgb = cv2.cvtColor(hand_img, cv2.COLOR_BGR2RGB)

    # 2. Flip the img in Y-axis
```

```
img_flip = cv2.flip(img_rgb, 1)

# accessing MediaPipe solutions
mp_hands = mp.solutions.hands # type: ignore

# Initialize Hands
hands = mp_hands.Hands(static_image_mode=True,
                max_num_hands=1, min_detection_confidence=0.7)

# Results
output = hands.process(img_flip)

hands.close()

try:
    data = output.multi_hand_landmarks[0]
    data = str(data)
    data = data.strip().split('\n')

    garbage = ['landmark {', '  visibility: 0.0', '  presence: 0.0', '}']

    without_garbage = []

    for i in data:
        if i not in garbage:
            without_garbage.append(i)

    clean = []

    for i in without_garbage:
        i = i.strip()
        clean.append(i[2:])

    for i in range(0, len(clean)):
        clean[i] = float(clean[i])

 return (clean)
```

```
    except:
        return (np.zeros([1, 63], dtype=int)[0])
#

# Load model
#with open('telusko\\model.pkl', 'rb') as f: # type: ignore
#svm = pickle.load('/telusko/model.pkl')
svm=pickle.load(open('C://Users//sange//Desktop//learning//sem6
project//django//telusko//model//test1','rb+'))
def gen(camera):
    while True:
        frame = camera.get_frame()

        yield(b'--frame\r\n'
              b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n\r\n')

@gzip.gzip_page
def livefe(request):
    try:
        cam = VideoCamera()
        return StreamingHttpResponse(gen(cam), content_type="multipart/x-mixed-
replace;boundary=circle")
    except:  # This is bad! replace it with proper handling
        pass
```

Urls.py

```
from django.contrib import admin
from django.urls import path
from django.contrib.staticfiles.urls import staticfiles_urlpatterns

from . import views

urlpatterns=[
     path('',views.home,name='home'),
 path('camera', views.livefe,name='home'), # type: ignore

   path('start_conversion',views.start_conversion,name='start_conversion'),
   path('home',views.index,name='home'),
   path('about',views.about,name='about us'),
   path('pp',views.pp,name='privacy_policy'),
   path('y_sign_language', views.y_sign_language,name='why sign language'),
   path('how_works', views.how_works, name='how_works'),

]
```

hands_gesture_recog.py

```
import cv2
import mediapipe as mp
import pandas as pd
import os
import numpy as np
import pickle
import cv2 as cv
import googletrans
from googletrans import Translator

# Initialize translator
translator = Translator()

def keymaxs(words,destination_language):

    keymax = max(zip(words.values(), words.keys()))[1]

    translation = translator.translate(keymax, dest=destination_language)

    print("word =",keymax.strip('\n\n'))
    print("tranlated word =",translation.text)

def image_processed(hand_img):
```

```
# Image processing
# 1. Convert BGR to RGB
img_rgb = cv2.cvtColor(hand_img, cv2.COLOR_BGR2RGB)

# 2. Flip the img in Y-axis
img_flip = cv2.flip(img_rgb, 1)

# accessing MediaPipe solutions
mp_hands = mp.solutions.hands

# Initialize Hands
hands = mp_hands.Hands(static_image_mode=True,
                       max_num_hands=1, min_detection_confidence=0.7)

# Results
output = hands.process(img_flip)

hands.close()

try:
    data = output.multi_hand_landmarks[0]
    data = str(data)
    data = data.strip().split('\n')

    garbage = ['landmark {', '  visibility: 0.0', '  presence: 0.0', '}']

    without_garbage = []

    for i in data:
        if i not in garbage:
            without_garbage.append(i)

    clean = []

    for i in without_garbage:
     i = i.strip()
            clean.append(i[2:])

        for i in range(0, len(clean)):
            clean[i] = float(clean[i])
        return (clean)
    except:
        return (np.zeros([1, 63], dtype=int)[0])


# Load model
with open('model.pkl', 'rb') as f:
    svm = pickle.load(f)

# Define languages
languages = {
    'en': 'english',
    'ta': 'tamil',
    'hi': 'hindi'
}

# Ask user for language choice
while True:
    lang_choice = input('Enter language choice (en for English, ta for Tamil, hi for
Hindi): ')
    if lang_choice in languages:
        break
    #print('Invalid choice, please try again.')

# Get language name
lang_name = languages[lang_choice]

# Main loop
a='maxresdefault.jpg'
cap = cv2.VideoCapture(a)
```

```
words={}
try:
    while True:
      ret, frame = cap.read()

      if not ret:
        #print("Can't receive frame (stream end?). Exiting ...")
        break

      data = image_processed(frame)
      data = np.array(data)



      # Predict gesture using model
      y_pred = svm.predict(data.reshape(-1, 63))

      # Translate output based on language choice
      output_text = str(y_pred[0])



      # Add text to frame
      font = cv2.FONT_HERSHEY_SIMPLEX
      org = (50, 100)
      fontScale = 3
      color = (255, 0, 0)
      thickness = 5

      if str(output_text)in words.keys():
       words[str(output_text)] += 1
      else:
        words[str(output_text)] = 1
```

```
 frame = cv2.putText(frame, output_text, org, font, fontScale, color, thickness,
cv2.LINE_AA)

            cv.imshow('frame', frame)
            if cv.waitKey(1) == ord('q'):
                break

        cap.release()
        cv.destroyAllWindows()
except:
    pass
finally:
   keymaxs(words, lang_choice)
```

APPENDIX 2

SCREEN SHOTS

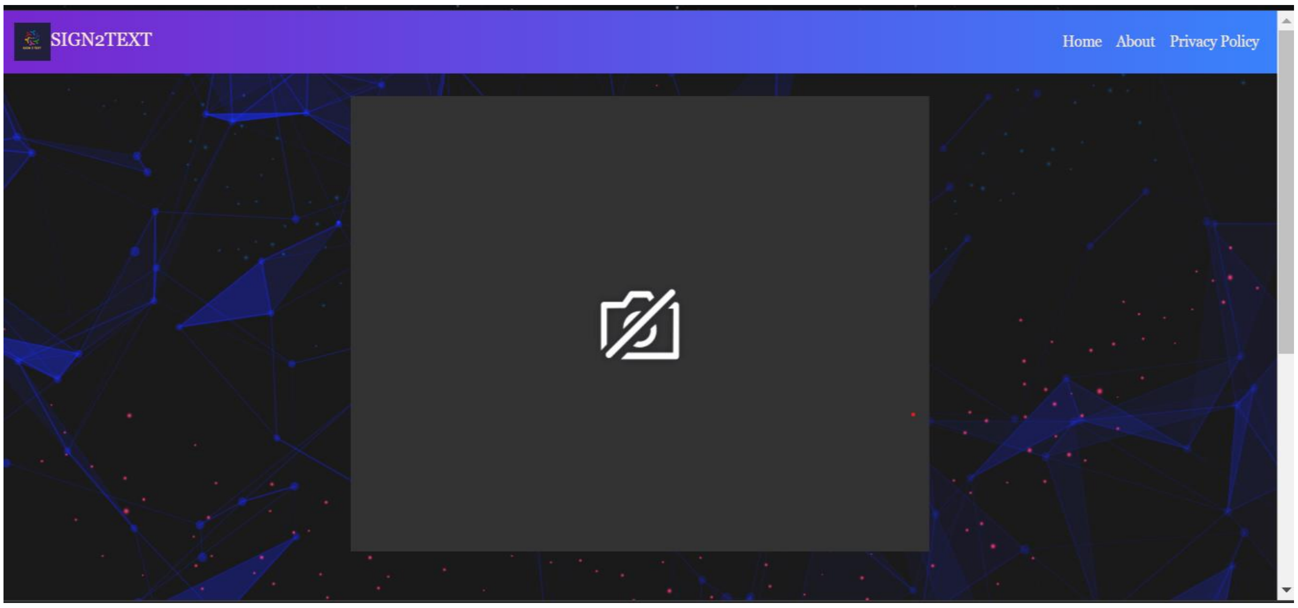


Figure A2.1: Opening Camera

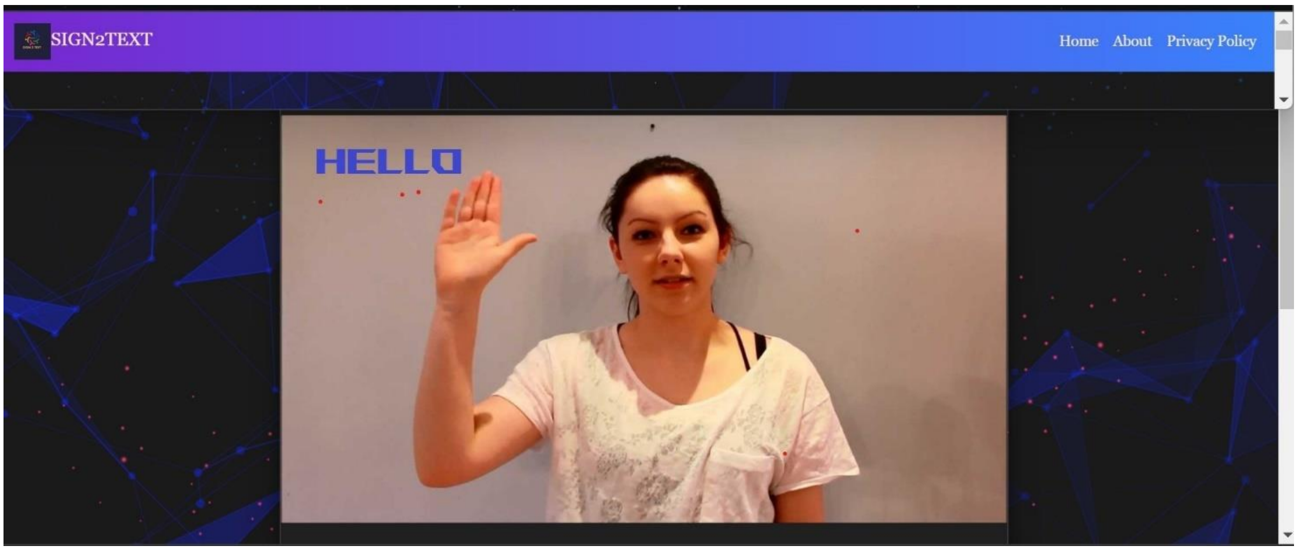
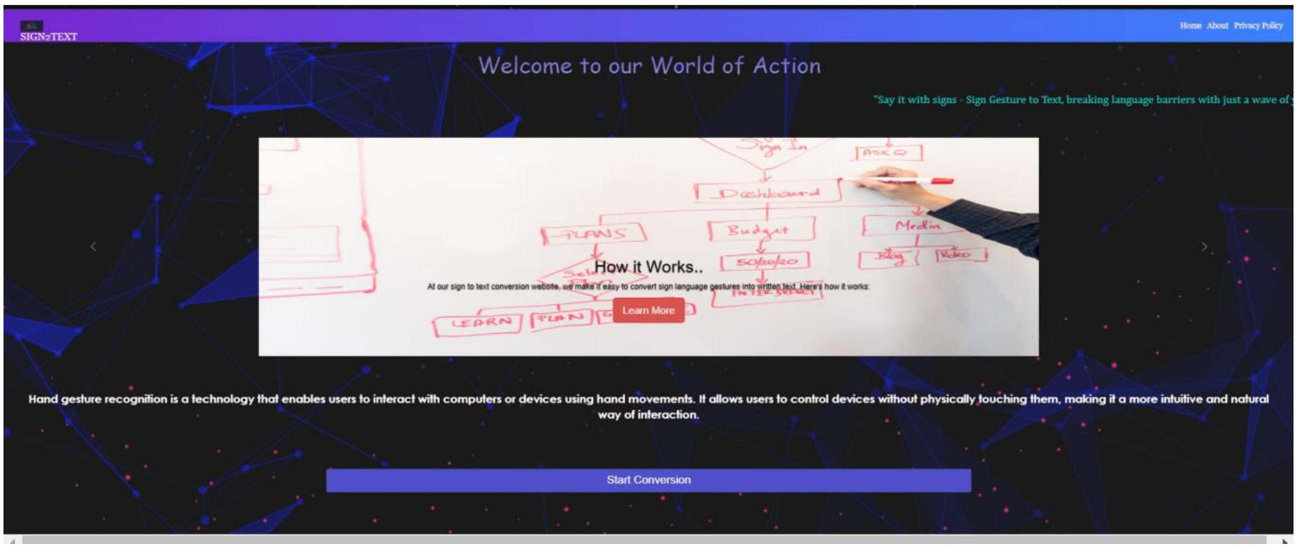


Figure A2.2: Gesture recognized



Welcome to our World of Action

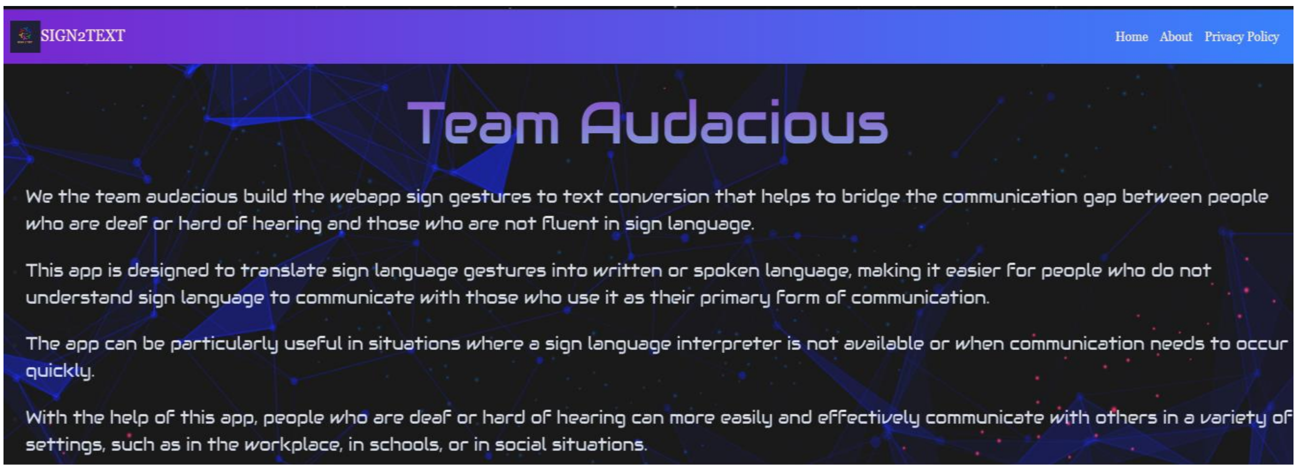
"Say it with signs - Sign Gesture to Text, breaking language barriers with just a wave of

How it Works..

At our sign to text conversion website, we make it easy to convert sign language gestures into written text. Here's how it works:

Hand gesture recognition is a technology that enables users to interact with computers or devices using hand movements. It allows users to control devices without physically touching them, making it a more intuitive and natural way of interaction.

Figure A2.3: Home Page



Team Audacious

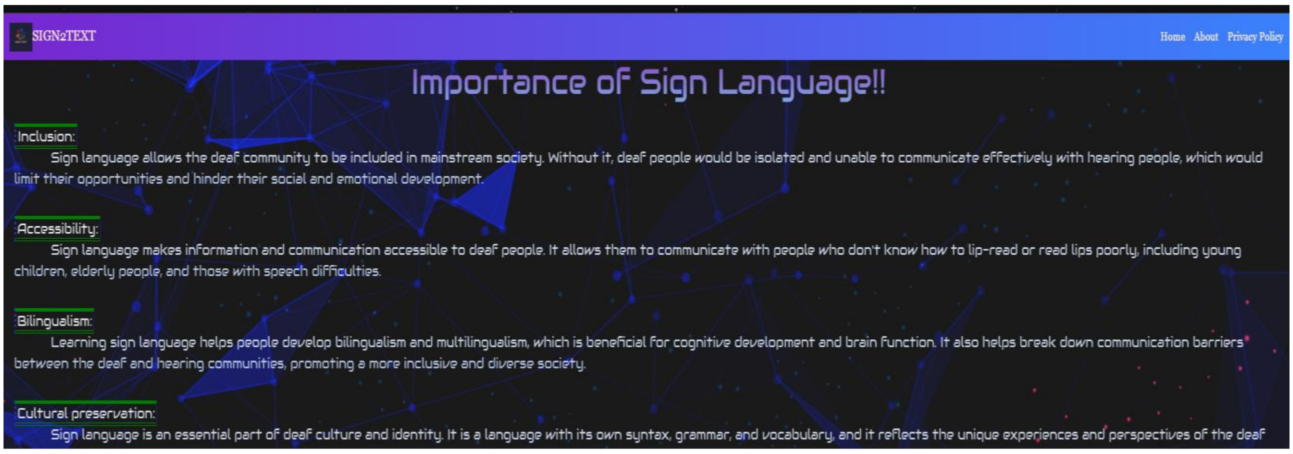
We the team audacious build the webapp sign gestures to text conversion that helps to bridge the communication gap between people who are deaf or hard of hearing and those who are not fluent in sign language.

This app is designed to translate sign language gestures into written or spoken language, making it easier for people who do not understand sign language to communicate with those who use it as their primary form of communication.

The app can be particularly useful in situations where a sign language interpreter is not available or when communication needs to occur quickly.

With the help of this app, people who are deaf or hard of hearing can more easily and effectively communicate with others in a variety of settings, such as in the workplace, in schools, or in social situations.

Figure A2.4: About page



Importance of Sign Language!!

Inclusion:

Sign language allows the deaf community to be included in mainstream society. Without it, deaf people would be isolated and unable to communicate effectively with hearing people, which would limit their opportunities and hinder their social and emotional development.

Accessibility:

Sign language makes information and communication accessible to deaf people. It allows them to communicate with people who don't know how to lip-read or read lips poorly, including young children, elderly people, and those with speech difficulties.

Bilingualism:

Learning sign language helps people develop bilingualism and multilingualism, which is beneficial for cognitive development and brain function. It also helps break down communication barriers between the deaf and hearing communities, promoting a more inclusive and diverse society.

Cultural preservation:

Sign language is an essential part of deaf culture and identity. It is a language with its own syntax, grammar, and vocabulary, and it reflects the unique experiences and perspectives of the deaf

Figure A2.5: Privacy Policy

REFERENCES

[1] Advaith Sridhar, Rohith Gandhi Ganesan, MM '20: Proceedings of the 28th ACM International Conference on Multimedia October 2020, INCLUDE: A Large-Scale Dataset for Indian Sign Language Recognition.

[2] Ahmed KASAPBAŞI, Arif YILMAZ Received 21 June 2021, Revised 13 October 2021,Accepted 29 December 2021, Available online 10 January 2022, Version of Record 12 January 2022. DeepASLR: A CNN based human computer interface for American Sign Language recognition for hearing-impaired individuals.

[3] Akash Junnarkar, Lance Fernandes 2020 Third International Conference on Smart Systems and Inventive Technology (ICSSIT) Convolutional Neural Network based Bidirectional Sign Language Translation System.

[4] Al Amin Hosain, Panneer Selvam Santhalingam, Published in: 2020 15th IEEE International Conference on Automatic Face and Gesture Recognition (FG 2020), FineHand: Learning Hand Shapes for American Sign Language Recognition.

[5] B Sundar, T Bagyammal, Published by Elsevier B.V 2023 American Sign Language Recognition for Alphabets Using MediaPipe and LSTM.

[6] Karan Bhavsari, Raj Ghatiya, published in International Journal of Research Publication and Reviews 2021. Sign Language Recognition.

[7] Mohamed Mahyoub, Friska Natalia, published in: 2023 15th International Conference on Developments in eSystems Engineering (DeSE) Sign Language Recognition using Deep Learning.

[8] O. M. Sincan and H. Y. Keles, IEEE Access, vol. 8, pp. 181340-181355, 2020. "Autsl: A large scale multi-modal turkish sign language dataset and baseline methods".

[9] Romala Sri Lakshmi Murali, L.D.Ramayya . International Journal of Engineering Innovations in Advanced Technology ISSN: 2582-1431 (Online), Volume-4 Issue-4, December 2022. Sign Language Recognition System Using Convolutional Neural Network and Computer Vision.